THE UNIVERSITY OF ADELAIDE

DEPARTMENT OF MECHANICAL ENGINEERING

A Methodology for Modelling the Steady-State Thermal Performance of Air Conditioning Systems

submitted by

Patrick George Marshallsay, M.Sc. (Flin)., B.Tech. (Adel).

Thesis for the Degree of Doctor of Philosophy

January, 1996

Table of Contents

Abstract

In meeting new standards relating to human thermal comfort and indoor air quality, while minimizing energy consumption and capital costs, the air conditioning designer is faced with a particularly challenging task. Selection of a suitable design solution requires the evaluation of the performance of a series of candidate solutions not only at peak load, but also over a representative range of possibly more demanding part-load conditions. The ability to undertake such an evaluation programme is a necessary first step in seeking a design solution which is not only satisfactory, but is also near-optimal. Unfortunately, the design tools currently available to the practising designer are inadequate for the task at hand, and severely restrain the designer in his evaluation of alternative design solutions. Further, the existing design tools tend to accent historical precedent at the expense of fundamental physics, and hence the quest for an optimal solution is compromised by inadequacies in the problem formulation.

In the current study a thorough analysis of the modelling strategies required to simulate a broad range of air conditioning systems of both conventional and innovative design has been undertaken. This has led to the development of a hierarchical set of data structures and associated algorithms which will permit such systems to be modelled with a high degree of generality. These structures and algorithms form the basis for a computational code known as ZEBRA, which in the longer term is intended to form the basic building block for a comprehensive computer-aided approach to air conditioning system design. The validity and scope of the modelling methodology developed in this investigation is demonstrated by its application to number of case studies, some of which refer to installed systems, and some of which are conceptual in nature. Since the ZEBRA code is based rigorously on the component fundamental sciences which form the foundations of air conditioning practice, new insights emerge from the case studies which enhance the disciplined understanding of air conditioning. This disciplined understanding provides a firm foundation for new, more logical design procedures.

Statement of Originality

The material contained in this thesis is original and has not been submitted or accepted for the award of a degree or diploma at any other university. To the best of the author's knowledge and belief, no material which has been published or written by another person is included, except when due reference is made in the text of the thesis.

Patrick George Marshallsay
(8th January, 1996)

I give consent to this copy of my thesis, when deposited in the University Library, being available for loan and photocopying.

Signed Patrick George MARSHALLSAY

Date 18th March, 1996

Acknowledgements

I would like to acknowledge the advice, support and assistance of the many individuals with whom I have had dealings in the course of the work described in this thesis. I am especially indebted to my supervisor, Professor R.E. Luxton, for his friendship and professional guidance over many years, for his encouragement to embark on my present course of study, and for his unfailing support and invaluable advice during the term of my PhD candidature.

Thanks are also due to the staff and postgraduate students of the Department of Mechanical Engineering at the University of Adelaide who contributed to the work undertaken during the course of my candidature, either directly by way of comments or assistance, or indirectly by creating a pleasant working environment. In this respect I am especially grateful to Dr. Alan Shaw and Dr. Chandra Sekhar for much fruitful discussion during the early stages of the work.

I am also indebted to my wife Zai, my mother, and our friends outside the University for their encouragement and understanding during the preparation of this thesis. Zai deserves special thanks for proof-reading the thesis.

Abbreviations.

AFV	Air Face Velocity
AHU	Air Handling Unit
APD	Air Pressure Drop
ARI	Air Conditioning and Refrigeration Institute
ASHRAE	American Society of Heating, Refrigerating and Air-Conditioning Engineers
AU	Adelaide University
CAV	Constant Air Volume
CHI	Comfort Health Index
CLI	Command Line Interpreter
CPU	Central Processing Unit
CW	Chilled Water
DX	Direct Expansion
HCV	High Coolant Velocity
HDP	High Driving Potential
LFV	Low Face Velocity
LMHD	Log Mean Enthalpy Difference
LMTD	Log Mean Temperature Difference
LMWD	Log Mean Humidity Ratio Difference
PC	Personal Computer
PDL	Programme Description Language
SSR	Sum of Squares of Residuals
TX	Thermostatic Expansion
VAV	Variable Air Volume
WPD	Water Pressure Drop
WTR	Water Temperature Rise
WV	Water Velocity
ZK	Zebra Kernel
ZPL	Zebra Programming Language
ZUI	Zebra User Interface

Notation.

A

a	Internal pipe radius (m).	Eq. (7.6.15)
A	Cross-sectional area of a conduit (m²).	Eq. (9.3.2)
A_d	Dry surface area for coil (m²).	Eq. (7.3.23)
A_f	Coil face area (m²).	Eq. (7.1.8)
A_i	Total internal surface area for coil (m²).	Eq. (7.1.13)
A_m	Minimum free-flow area for a coil (m²).	Eq. (7.1.9)
A_o	Total external surface area for coil (m²).	Eq. (7.1.12)
A_p	Primary surface area for a coil (m²).	Eq. (7.1.10)
A_s	Secondary surface are for coil (m²).	(Eq. 7.1.11)
A_w	Wet surface area for coil (m²).	Eq. (7.3.23)

B

B	Ratio of total external to total internal surface area for a coil.	Eq. (7.1.14)

C

c	Constant (dimensionless).	Eq. (7.3.17)
C	Coil characteristic (°C.kg/kJ).	Eq. (7.3.8)
C	Local friction loss coefficient (dimensionless).	Eq. (7.7.1)
C_a	Air-side capacity rate for a heat exchanger (kW).	Eq. (8.4.7)
C_p	Isobaric specific heat (kJ/Kg.K).	Eq. (4.1.26)
C_p^0	Ideal gas isobaric specific heat (kJ/Kg.K).	Eq. (11.1.11)
C_{pa}	Isobaric specific heat of dry air (kJ/kg.K).	Eq (4.1.26)
$C_{p,a}$	Isobaric specific heat of moist air (kJ/kg.K).	Eq. (5.1.1)
$C_{p,l}$	Isobaric specific heat of a liquid (kJ/kg.K).	Sec. (11.1.1.1)
C_{pw}	Isobaric specific heat of water vapour (kJ/kg.K).	Eq. (4.1.26)
$C_{p,w}$	Isobaric specific heat of liquid water (kJ/kg.K).	Eq. (7.3.6)
C_v	Isochoric specific heat (kJ/Kg.K).	Sec. (11.1.1.1)
C_v^0	Ideal gas isochoric specific heat (kJ/Kg.K).	Eq. (11.1.11)
C_w	Water-side capacity rate for a heat exchanger (kW).	Eq. (8.4.6)
C_0	Fan system effect loss coefficient (dimensionless).	Eq. (10.2.13)

D

d	Fan impeller diameter (m).	Eq. (10.2.2)
D	Dean number (dimensionless).	Eq. (7.6.15)
dh_a	Change of enthalpy undergone by moist air flowing over an elemental heat transfer area (kJ/kg).	Eq. (7.3.1)
dq_s	Rate of heat transfer across an elemental area (kW).	Eq. (7.3.18)
dq_t	Rate of combined heat and mass transfer across an elemental area (kW).	Eq. (7.3.1)

dt_w	Change of temperature undergone by water flowing through an elemental length of tube (°C).	Eq. (7.3.6)
dW_a	Change of humidity undergone by moist air flowing over an elemental heat transfer area (kg water vapour per kg dry air).	Eq. (7.3.1)
D_{AB}	Mass diffusivity (m²/s).	Eq. (4.2.6)
D_{crit}	Critical Dean number (dimensionless).	Eq. (7.6.16)
D_f	Equivalent by friction loss duct diameter (m).	Eq. (9.3.1)
D_i	Internal diameter of cooling coil tube (mm).	Eq. (7.4.5)
D_h	Hydraulic diameter (m).	Eq. (7.2.1)
D_o	External diameter of cooling coil tube (mm).	Eq. (7.1.5)
D_v	Equivalent by velocity duct diameter (m).	Eq. (9.3.6)

E

ET^*	Effective temperature (°C).	Sec. (3.1)

F

f	Friction factor (dimensionless).	Eq. (7.6.6)
F	Correction factor for log-mean temperature difference (dimensionless).	Eq. (7.4.3)
f_{aD}	Effective air-side heat transfer coefficient for a dry surface (W/m².°C).	Sec. (7.4.4)
f_{aW}	Effective air-side heat transfer coefficient for a wet surface (W/m².°C).	Eq. (7.4.30)
f_c	Friction coefficient for fully developed curved flow (dimensionless).	Eq. (7.6.14)
f_d	Friction factor for a dry coil surface (dimensionless).	Eq. (7.7.9)
F_h	Height of free-flow passage between adjacent tubes in a cooling coil (mm).	Eq. (7.1.6)
f_s	Enhancement factor (dimensionless).	Eq. (4.1.10)
FSE_i	Fan system effect pressure loss caused by fan inlet conditions (Pa).	Eq. (10.1.4)
FSE_o	Fan system effect pressure loss caused by fan outlet conditions (Pa).	Eq. (10.1.4)
f_w	Friction factor for a wet coil surface (dimensionless).	Eq. (7.7.13)
$f_\eta^{(2)}$	Function for calculating the higher approximations to the coefficient of diffusion.	Eq. (4.2.6)

G

G	Mass flux (kg/m².s).	Eq. (7.5.14)
g_c	Gravitational constant (kg.m/N.s²).	Eq. (9.3.1)

H

h	Specific enthalpy of moist air (kJ/kg).	Eq. (4.1.23)
h	Height of a duct section (m).	Eq. (9.3.2)
h	Enthalpy (kJ/kg).	Sec. (11.1.1.1)
h_a	Specific enthalpy of dry air (kJ/kg).	Eq. (4.1.23)

h_a Specific enthalpy of moist air in free stream (kJ/kg). Eq. (7.3.4)
h_{ab} Specific enthalpy of free-stream air at dry-wet boundary (kJ/kg). Eq. (7.3.10)
h_{am} Mean specific enthalpy of free-stream air (kJ/kg). Eq. (7.3.10)
h_{a1} Specific enthalpy of free-stream air entering cooling coil (kJ/kg). Eq. (7.3.9)
h_{a1}'' Specific enthalpy of air at entering dew-point temperature (kJ/kg). Eq. (7.3.10)
h_{a2} Specific enthalpy of free-stream air leaving cooling coil (kJ/kg). Eq. (7.3.9)
h_c Convection heat transfer coefficient (W/m²K). Eq. (4.2.11)
$h_{c,ow}$ Convective heat transfer coefficient for outside wetted surface (W/m²K). Eq. (7.3.2)
h_f Saturated liquid enthalpy (kJ/kg). Sec. (11.1.1.1)
$h_{f,w}$ Specific enthalpy of saturated liquid water at temperature t_s (kJ/kg). Eq. (7.3.2)
h_{fg} Latent heat of vapourization of water (kJ/kg). Eq. (4.1.32)
h_{fg} Latent heat of vapourization (kJ/kg). Sec. (11.1.1.1)
h_g Specific enthalpy of saturated water (kJ/kg). Eq. (4.1.23)
h_g Saturated vapour enthalpy (kJ/kg). Sec. (11.1.1.1)
$h_{g,t}$ Specific enthalpy of saturated water vapour at air dry-bulb temperature (kJ/kg). Eq. (7.3.2)
h_i Specific enthalpy of air entering zone (kJ/kg). Eq. (6.1.4)
h_t Combined coefficient for heat transfer through the condensate layer, tube fins and walls, and coolant boundary layer or a wet heat transfer surface (W/m²K). (Eq. 7.3.7)
h_t Coolant-side film heat-transfer coefficient (W/m²K). (Eq. 7.4.5)
h_o Specific enthalpy of air leaving zone (kJ/kg). Eq. (6.1.3)
h_s Enthalpy of air saturated at an air-water interface (kJ/kg). Eq. (7.3.4)
h_{sb} Specific enthalpy of air saturated at an air-water interface at dry-wet boundary (kJ/kg). Sec. (6.3.3)
h_{sm} Mean specific enthalpy of air saturated at an air-water interface (kJ/kg). Eq. (7.4.30)
h_{s1} Specific enthalpy of air saturated at an air-water interface at entry to cooling coil (kJ/kg). Sec. (6.3.3)
h_{s2} Specific enthalpy of air saturated at an air-water interface at exit from cooling coil (kJ/kg). Sec. (6.3.3)
h_{s_*} Effective surface enthalpy for a cooling coil (kJ/kg). Eq. (7.3.14)
h_s^* Specific enthalpy of moist air saturated at specified wet bulb temperature (kJ/kg). Eq. (4.1.28)
h_w^* Specific enthalpy of water saturated at specified wet bulb temperture (kJ/kg). Eq. (4.1.28)

I

I_{cl} Degree of insulation provided by the clothing of the occupants of a space (clo). Eq. (3.1.1)
i_m Permeability index of clothing. Sec. (3.1)

J

j	Colburn j-factor for heat transfer (dimensionless).	Eq. (7.5.13)
j_m	Colburn j-factor for mass transfer (dimensionless).	Eq. (7.5.15)

K

k	Thermal conductivity (W/m.K).	Eq. (4.2.4)
K	Mass transfer coefficient (kg/m²s).	Eq. (7.2.1)
K	Pressure drop coefficient for a pipe bend (dimensionless).	Eq. (7.6.13)
K_G	Gross pressure drop coefficient for a pipe bend (dimensionless).	Eq. (7.6.14)
k_l	Thermal conductivity for a liquid (W/m.K).	Sec. (11.1.1.1)
K_s	Duct section conductance ($m^{7/2}/kg^{1/2}$).	Eq. (9.3.9)
k_t	Thermal conductivity of tube walls (W.mm/m².°C).	Eq. (7.4.7)
K_{ts}	System characteristic ($m^{7/2}/kg^{1/2}$).	Eq. (9.4.10)
K_W	Mass transfer coefficient based on specific humidity as the driving potential (kg/m²s).	Eq. (4.2.11)
K_y	Mass transfer coefficient based on mass fraction y as the driving potential (kg/m²s).	Sec. (4.2.4)
k^*	Thermal conductivity of vapour at atmospheric pressure (W/m.K).	Eq. (11.1.34)

L

L	Length of a duct section (m).	Eq. (9.3.1)
L_c	Width of the passage between adjacent fins in a cooling coil (mm).	Eq. (7.1.7)
L_d	Depth of a cooling coil (mm).	Eq. (7.1.4)
L_f	Face height of a cooling coil (mm).	Eq. (7.1.3)
L_t	Width of cooling coil face (exposed length of tubes; mm).	Eq. (7.1.2)
Le	Lewis number.	Eq. (4.2.11)

M

m	A constant.	Eq. (7.2.2)
m_a	Mass of dry air (kg).	Eq. (4.1.1)
M_a	Molccular weight of air.	Eq. (4.1.3)
M_s	Activity level of the occupants of a space (W/m²).	Eq. (3.1.1)
m_w	Mass of water vapour (kg).	Eq. (4.1.1)
M_w	Molecular weight (kg/kmol).	Eq. (4.1.3)
$\dot{m}$	Mass flow rate (kg/s).	Eq. (4.3.1)
$\dot{m}_a$	Mass flow rate of air (kg/s).	Eq. (7.3.1)
$\dot{m}_f$	Mass flow rate of air through a fan (kg/s).	Eq. (6.3.18)
$\dot{m}_{OA}$	Mass flow rate of air through an outdoor air connector (kg/s).	Eq. (6.3.19)
$\dot{m}_{RA}$	Mass flow rate of air through a return air connector (kg/s).	Eq. (6.3.19)
$\dot{m}_{SA}$	Mass flow rate of air through a supply air connector (kg/s).	Eq. (6.3.18)
$\dot{m}_w$	Mass flow rate of water (kg/s).	Eq. (7.3.6)

$\underline{m''}$	dh_s/dt_s (kJ/kg.°C).	Eq. (7.4.27)
$\underline{m}$	Multiplicity of an air handling unit (the number of times an identical air handling unit occurs in a defined situation)	Sec. (6.4.1)

N

N	Total number of tubes in a cooling coil.	Eq. (7.1.1)
N	Fan rotational speed (rev/s).	Eq. (10.2.5)
n_a	Number of moles of dry air in a sample.	Eq. (4.1.19)
N_{ct}	Number of parallel circuits in a coil.	Eq. (7.6.11)
N_d	Number of circuits to be deactivated in a coil.	Sec. (8.3.3)
N_e	Total number of circuits (active and inactive) in a portion of a coil in which circuits have been deactivated.	Sec. (8.3.3)
N_f	Total number of fins in a cooling coil.	Eq. (7.1.2)
$\dot{N}_f$	Fin density (fins per inch).	Eq. (7.1.2)
N_{pc}	Number of passes per circuit for a coil.	Eq. (7.6.11)
N_r	Number of rows in a cooling coil.	Eq. (7.1.1)
N_{rm}	Number of rows remaining in a portion of a coil in which circuits have been deactivated.	Sec. (8.3.3)
N_t	Face height of a cooling coil (tubes per row).	Eq. (7.1.1)
Nu	Nusselt number.	Eq. (7.6.6)

P

p	Pressure of moist air (Pa).	Eq. (4.1.6)
p	Operating pressure for a zone (Pa).	Eq. (9.4.8)
P	Perimeter of a duct section (m).	Eq. (9.3.2)
P	Datum pressure for a subtree (Pa).	Sec. (9.4.4)
P	Absolute pressure (kPa).	Sec. (11.1.1.1)
p_a	Partial pressure of air (Pa).	Eq. (4.1.6)
P_c	AHU casing pressure (Pa).	Eq. (10.3.5)
P_c	Critical pressure (kPa).	Sec. (11.1.1.2)
P_e	Electrical power required by fan motor (kW).	Eq. (10.1.2)
P_f	Shaft power absorbed by a fan (kW).	Eq. (10.1.1)
p_{in}	Water pressure at entry to a coil (kPa).	Eq. (8.4.3)
p_{out}	Water pressure at exit from a coil (kPa).	Eq. (8.4.3)
P_{sat}	Saturation pressure (kPa).	Sec. (11.1.1.1)
p_w	Partial pressure of water vapour (Pa).	Eq. (4.1.6)
p_{ws}	Saturation vapour pressure for water (Pa).	Eq. (4.1.4)
Pr	Prandtl number.	Eq. (4.2.9)
P'	Pressure at a duct terminal, referenced to the datum (Pa).	Sec. (9.4.4)
P_1	Fan inlet pressure (Pa).	Eq. (10.3.7)
P_2	Fan outlet pressure (Pa).	Eq. (10.3.2)

Q

$Q_{a,\,act}$	Actual volume flow rate of air (L/s).	Eq. (6.1.2)
$Q_{a,\,std}$	Volume flow rate of air under ASHRAE standard conditions (L/s).	Eq. (6.1.2)
q_{actual}	Actual rate of heat transfer for a heat exchanger (kW).	Eq. (8.4.5)
Q_f	Actual volume air flow through a fan (L/s).	Eq. (10.1.1)
Q_l	Rate of heat exchange between a duct section and its surroundings (W).	Eq. (9.3.10)
q_l	Sensible load (kW).	Eq. (6.1.1)
$q_{\max}$	Maximum possible rate of heat transfer for a heat exchanger (kW).	Eq. (8.4.5)
q_{rd}	Heat gain due to transmission through a return air duct wall (kW).	Sec. (5.1.1)
q_{rf}	Dissipation of return air fan energy (kW).	Sec. (5.1.1)
q_s	Latent load (kW).	Eq. (6.1.1)
Q_s	Air flow through a duct system (L/s).	Eq. (9.4.10)
q_{sd}	Heat gain due to transmission through a supply air duct wall (kW).	Sec. (5.1.1)
q_{sf}	Dissipation of supply air fan energy (kW).	Sec. (5.1.1)
q_t	Cooling capacity of a coil surface (kW).	Eq. (7.3.24)
q_{td}	Cooling capacity of the dry portion of a coil surface (kW).	Eq. (7.3.24)
q_{tw}	Cooling capacity of the wet portion of a coil surface (kW).	Eq. (7.3.24)
Q_w	Chilled water flow rate (L/s).	Fig. (6.14)

R

r	Radius of a condensate drop (m).	Eq. (7.2.2)
$\hat{r}$	Effective maximum radius of a condensate drop (m).	Eq. (7.2.2)
R	Universal gas constant (kJ/kg.K).	Eq. (4.1.19)
R	Overall resistance to heat transfer between air and coolant (m².°C/W).	Eq. (7.4.4)
R_a	Gas constant for dry air (kJ/kg.K).	Eq. (4.1.20)
R_{aD}	Dry air-side thermal resistance (m².°C/W).	Eq. (7.4.4)
R_{aW}	Wet air-side thermal resistance (m².°C/W).	Eq. (7.4.19)
R_c	Radius of curvature of a pipe bend (m).	Eq. (7.6.14)
R_f	Thermal resistance offered by coil fins (m².°C/W).	Eq. (7.4.6)
R_i	Combined thermal resistance for outside condensate layer, metal, and coolant-side film (m².°C/W).	Eq. (7.5.10)
R_m	Total metal thermal resistance (m².°C/W).	Eq. (7.4.4)
R_{mD}	Total metal thermal resistance for a dry surface (m².°C/W).	Eq. (7.4.32)
R_{mW}	Total metal thermal resistance for a wet surface (m².°C/W).	Eq. (7.4.22)
R_{ow}	Thermal resistance for the condensate layer (m².°C/W).	Eq. (7.5.11)
R_t	Thermal resistance offered by tube walls (m².°C/W).	Eq. (7.4.6)
R_w	Coolant-side thermal resistance (m².°C/W).	Eq. (7.4.4)
Re	Reynolds number (dimensionless).	Eq. (7.6.6)
Re_{crit}	Critical Reynolds number (dimensionless).	Eq. (7.6.16)
Re_d	Reynolds number based on hydraulic diameter (dimensionless).	Eq. (7.5.20)
Re_s	Reynolds number based on fin spacing (dimensionless).	Eq. (7.7.10)

Re^*	Reynolds number based on hydraulic diameter, and corrected using a geometry factor (dimensionless).	Eq. (7.5.23)

S

s	Entropy (kJ/kg.K).	Sec. (11.1.1.1)
S_f	Vertical spacing between adjacent tubes in a cooling coil (mm).	Eq. (7.1.6)
s_f	Entropy of a saturated liquid (kJ/kg.K).	Sec. (11.1.1.1)
s_{fg}	Entropy change during vapourization of a liquid (kJ/kg.K).	Sec. (11.1.1.1)
s_g	Entropy of a saturated vapour (kJ/kg.K).	Sec. (11.1.1.1)
s_1	Functional relationship between total pressure coefficient and volume coefficient.	Eq. (10.2.11)
S_1	Spacing between the leading edge of a fin in a cooling coil and the centreline of the first row of tubes (mm).	Eq. (7.1.4)
s_2	Functional relationship between power coefficient and volume coefficient.	Eq. (10.2.11)
S_2	Spacing between the trailing edge of a fin in a cooling coil and the centreline of the last row of tubes (mm).	Eq. (7.1.4)
S_3	Spacing between the lower edge of a fin in a cooling coil and the centreline of the lowest tubes in the coil (mm).	Eq. (7.1.3)
S_4	Spacing between the upper edge of a fin in a cooling coil and the centreline of the uppermost tubes in the coil (mm).	Eq. (7.1.3)
Sc	Schmidt number.	Eq. (4.2.10)
Sh	Sherwood number.	Eq. (7.2.1)
St	Stanton number.	Eq. (7.5.14)
St_m	Mass transfer Stanton number.	Eq. (7.5.16)

T

t	Temperature (°C).	Eq. (4.1.24)
T	Absolute temperature (K).	Eq. (4.1.4)
t_a	Temperature of air in free stream (°C).	Eq. (7.3.2)
t_a	Temperature of air external to a conduit (°C).	Eq. (9.3.10)
t_{ab}	Temperature of free-stream air at a dry-wet boundary (°C).	Eq. (7.3.11)
t_{al}	Temperature of free-stream air entering a cooling coil (°C).	Eq. (7.3.11)
t_{al}''	Dew-point temperature of free-stream air entering a cooling coil (°C).	Eq. (7.3.10)
t_{a2}	Temperature of free-stream air leaving cooling coil (°C).	Eq. (7.3.21)
T_{br}	Normal boiling point (K).	Sec. (11.1.1.2)
T_c	Critical temperature (K).	Sec. (11.1.1.2)
t_d	Dew-point temperature (°C).	Eq. (4.1.15)
t_e	Temperature of air entering a conduit (°C).	Eq. (9.3.10)
$t_{f,b}$	Temperature at base of fin (°C).	Eq. (7.4.8)
$t_{f,m}$	Mean fin temperature (°C).	Eq. (7.4.8)
T_{fr}	Freezing point (K).	Sec. (11.1.1.2)
t_i	Temperature of air entering zone (°C).	Eq. (6.1.7)
t_l	Temperature of air leaving a conduit (°C).	Eq. (9.3.10)

t_m	Mean temperature (°C).	Eq. (7.5.8)
t_o	Temperature of air leaving zone (°C).	Eq. (6.1.6)
T_r	Reduced temperature.	Eq. (4.2.6)
t_s	Supply air temperature (°C).	Fig. (6.9)
t_s	Temperature of saturated air at an air-water interface (°C).	Eq. (7.3.2)
T_{sat}	Saturation temperature (K).	Sec. (11.1.1.1)
t_{sb}	Temperature of saturated air at air-water interface at dry-wet boundary (°C).	Eq. (7.3.10)
t_{sm}	Mean temperature of saturated air at air-water interface (°C).	Eq. (7.4.30)
t_{s1}	Temperature of saturated air at air-water interface at entry to cooling coil (°C).	Sec. (6.3.3)
t_{s2}	Temperature of saturated air at air-water interface at exit from cooling coil (°C).	Sec. (6.3.3)
$t_{\bar{s}}$	Effective surface temperature for a cooling coil (°C).	Eq. (7.3.19)
t_{set}	Thermostat set point (°C).	Sec. (6.3.4.1.5)
t^*	Wet bulb temperature (°C).	Eq. (4.1.29)
t_w	Water temperature (°C).	Eq. (7.3.6)
t_{wb}	Temperature of water at a dry-wet boundary (°C).	Eq. (7.3.11)
t_{wm}	Mean water temperature (°C).	Eq. (7.4.5)
t_{w1}	Temperature of water entering a cooling coil (°C).	Eq. (7.3.9)
t_{w2}	Temperature of water leaving a cooling coil (°C).	Eq. (7.3.9)

U

u	Fan impeller peripheral velocity (m/s).	Eq. (10.2.3)
U	Overall heat transfer coefficient (W/m².°C).	Eq. (9.3.10)
$U_{o,d}$	Overall heat transfer for the dry portion of a coil surface (W/m².C).	Eq. (7.4.1)
U_w	Mean water velocity (m/s).	Eq. (7.4.5)

V

v	Specific volume of dry air (kg/m³).	Eq. (4.1.18)
v	Velocity of the air leaving a fan (m/s).	Eq. (10.2.1)
v	Specific volume (m³/kg).	Sec. (11.1.1.1)
V	Volume (m³).	Eq. (4.1.19)
V	Velocity of air in a conduit (m/s).	Eq. (9.3.1)
$V_{a,std}$	Standard air velocity(m/s).	Eq. (7.4.18)
V_c	Air velocity based on minimum free-flow area of coil (m/s).	Eq. (7.7.1)
V_c	Critical volume (m³/kg).	Sec. (11.1.1.2)
v_f	Specific volume of saturated liquid (m³/kg).	Sec. (11.1.1.1)
V_f	Air face velocity for coil (m/s).	Eq. (7.7.1)
v_{fg}	Specific volume change during vapourization of a liquid (m³/kg).	Sec. (11.1.1.1)
v_g	Specific volume of saturated vapour (m³/kg).	Sec. (11.1.1.1)
V_0	Fan inlet or outlet velocity (m/s).	Eq. (10.2.13)

W

w	Fin height (mm).	Eq. (7.4.13)
w	Width of a duct section (m).	Eq. (9.3.2)
W	Humidity ratio (kg water vapour per kg dry air).	Eq. (4.1.1)
$\tilde{W}$	Estimate for humidity ratio (kg water vapour per kg dry air).	Eq. (6.3.1)
W_a	Humidity ratio of moist air in free stream (kg water vapour per kg dry air).	Eq. (7.3.2)
W_{a1}	Humdity ratio of moist air in free stream at entry to coil (kg water vapour per kg dry air).	Eq. (7.3.2)
W_{a2}	Humdity ratio of moist air in free stream at exit from coil. (kg water vapour per kg dry air).	Eq. (7.3.2)
W_i	Humidity ratio of air entering zone (kg water vapour per kg dry air).	Eq. (6.1.6)
W_m	Mean humidity ratio (kg water vapour per kg dry air).	Eq. (7.5.8)
W_o	Humidity ratio of air leaving zone (kg water vapour per kg dry air).	Eq. (6.1.6)
W_s	Saturation humidity ratio (kg water vapour per kg dry air).	Eq. (4.1.9)
W_s	Supply air humidity ratio (kg water vapour per kg dry air).	Fig. (6.9)
W_s	Humidity ratio of air saturated at an air-water interface (kg water vapour per kg dry air).	Eq. (7.3.2)
W_{s1}	Humidity ratio of air saturated at an air-water interface at entry to cooling coil (kg water vapour per kg dry air).	Eq. (7.3.2)
W_{s2}	Humidity ratio of air saturated at an air-water interface at exit from cooling coil (kg water vapour per kg dry air).	Eq. (7.3.2)
W_s^*	Saturation humidity ratio at specified wet bulb temperature (kg water vapour per kg dry air).	Eq. (4.1.28)

X

x	Quality (dimensionless).	Sec. (11.1.1.1)
x_a	Mole fraction of dry air.	Eq. (4.1.2)
x_b	Fin root radius for cooling coil (mm).	Eq. (7.1.5)
x_e	Radius of annular fin (mm).	Eq. (7.4.15)
x_w	Mole fraction of water.	Eq. (4.1.2)

Y

y	Parameter defined by equation (7.3.9) (°C.kg/kJ).	Eq. (7.3.9)
Y_f	Fin thickness for cooling coil (mm).	Eq. (7.1.5)

Z

z_c	Compressibility factor at the critical point (dimensionless).	Sec. (11.1.1.2)

Greek Symbols

α	Thermal diffusivity (m²/s).	Sec. (4.2.4)
α	Fraction of a surface area covered with drops with radius greater than a specified value.	Eq. (7.2.2)
β	Contact angle between a condensate drop and a surface.	Sec. (7.2.2)
β	Isobaric compressibility (K^{-1}).	Eq. (11.1.7)
γ	Ratio of specific heats (C_p/C_v).	Eq. (11.1.11)
Δh_m	Log-mean enthalpy difference (kJ/kg).	Eq. (9.4.6)
Δp_w	Pressure drop through a coil circuit (kPa).	Eq. (7.6.11)
Δp	Pressure drop (kPa).	Eq. (7.7.1)
ΔP	Pressure drop for a duct section (Pa).	Eq. (9.3.1)
ΔP	Pressure drop for a path in a network (Pa).	Eq. (9.4.6)
ΔP_{max}	Maximum pressure drop in a network (Pa).	Eq. (9.4.6)
ΔP_p	Pressure drop through a pass of straight pipe in a coil (kPa).	Eq. (7.6.11)
Δp_f	Static pressure rise across a fan (Pa).	Eq. (10.2.1)
Δp_{sys}	Duct system pressure drop (Pa).	Eq. (9.4.10)
Δp_t	Total pressure rise across a fan (Pa).	Eq. (10.1.1)
Δp_{ts}	System total pressure rise (Pa).	Eq. (10.1.4)
Δp_w	Pressure drop through a coil circuit (kPa).	Eq. (7.6.11)
Δp_w	Pressure drop across a coil bank (kPa).	Eq. (8.4.3)
$\Delta P'$	Pressure required to balance a path in a network (Pa).	Eq. (9.4.6)
ΔP^+	Extra pressure drop required to balance a duct system by closing a damper (Pa).	Eq. (9.4.11)
Δt	Temperature differential (°C).	Eq. (5.1.1)
Δt_f	Fan temperature gain (°C).	Sec. (6.3.3)
$\Delta t_{f,b}$	Temperature differential between base of fin and air (°C).	Eq. (7.4.8)
$\Delta t_{f,m}$	Mean temperature differential between fin surface and air (°C).	Eq. (7.4.8)
Δt_m	Mean temperature difference for a heat exchanger (°C).	Eq. (7.4.1)
Δt_r	Return air duct temperature gain (°C).	Sec. (6.3.3)
Δt_s	Supply air duct temperature gain (°C).	Sec. (6.3.3)
Δt_w	Water temperature rise across coil bank (°C).	Sec. (8.4.4)
ΔW_m	Log-mean humidity ratio difference for a heat exchanger (kg water vapour per kg dry air).	Eq. (7.4.1)
ϵ	Heat exchanger effectiveness (dimensionless).	Eq. (8.4.5)
ϵ	Roughness (m).	Eq. (9.3.7)
η	Fin effectiveness (dimensionless).	Eq. (7.4.9)
η_f	Fan efficiency (total).	Eq. (10.1.1)
η_m	Fan motor efficiency.	Eq. (10.1.2)
θ	Damper blade angle (°).	Eq. (9.4.4)
θ_a	Mean radiant temperature of the surfaces in a space, weighted according toarea (°C).	Eq. (3.1.2)
θ_r	Temperature of the air in a space (°C).	Eq. (3.1.2)
θ_{rs}	Resultant dry-bulb temperature (°C).	Eq. (3.1.1)
Θ	Angle subtended by a pipe bend.	Eq. (7.6.14)

μ	Degree of saturation.	Eq. (4.1.12)
μ	Dynamic viscosity (pa.s).	Eq. (4.2.1)
μ_l	Dynamic viscosity for a liquid (pa.s).	Sec. (11.1.1.1)
μ^*	Dynamic viscosity of vapour at atmospheric pressure (pa.s).	Eq. (11.1.31)
v_f	Specific volume of liquid water (m³/kg).	Eq. (7.6.1)
ρ	Density of moist air (kg/m³).	Eq. (4.1.22)
ρ_c	Critical density (kg/m³).	Eq. (11.1.32)
ρ_{act}	Actual density of moist air (kg/m³).	Eq. (6.1.2)
ρ_r	Reduced density (ρ/ρ_c).	Eq. (11.1.32)
σ	Surface tension (N/m).	Sec. (11.1.1.1)
$\Sigma\Delta P$	Sum of pressure drops over a path connecting a network datum with a terminal (Pa).	Sec. (9.4.4)
ϕ	Relative humidity.	Eq. (4.1.13)
ϕ	Fin efficiency.	Eq. (7.4.8)
ϕ	Fan volume coefficient (dimensionless).	Eq. (10.2.2)
Φ_{ij}	Constants in equation for dynamic viscosity of a mixture.	Eq. (4.2.1)
ϕ^*	Geometry factor (dimensionless).	Eq. (7.5.23)
χ	Constant, takes a value of +1 for a diverging network, and -1 for a converging network.	Eq. (9.4.8)
χ	Constant, takes a value of 1 if a fan motor is located within the air stream, and η_m if the motor is located outside the air stream.	Eq. (10.1.3)
ψ	Fan total pressure coefficient (dimensionless).	Eq. (10.2.3)
$\Omega^{(1,1)*}$	Reduced collision integral for calculating the transport coefficients.	Eq. (4.2.6)

Other Symbols.

$\wp_i$	Pressure loss through a branch of the supply-air subsystem (duct + coil + filter losses) (Pa).	Eq. (10.3.1)
$\wp_O$	Pressure loss through the outdoor air subsystem (duct + coil + filter losses) (Pa).	Eq. (10.3.4)
$\wp_R$	Pressure loss through the return air subsystem (duct + coil + filter losses) (Pa).	Eq. (10.3.3)

Chapter 1. Introduction.

1.1.Background.

It is well known that the humidity within a conditioned space may readily be controlled to a desired comfort level by the use of overcooling to establish the desired humidity ratio followed by reheating to establish the desired temperature. Indeed, such practices comprised the stock in trade of most air conditioning designers in the years prior to the energy crisis of the early seventies. Systems designed according to this methodology were undoubtedly capable of maintaining indoor air conditions of high quality, but only by means of profligate expenditure of energy. While energy was "cheap", such practices were accepted without question. However, the escalating costs of energy during the seventies led to a realization of the wastefulness of such practices, and to a demand for the adoption of more energy efficient techniques.

Revised codes and standards were enacted to encourage, and in some cases through weight of legislation to enforce a more responsible attitude towards the energy consumed by air conditioning systems. In response, the building and mechanical services industries enthusiastically adopted new or hitherto neglected techniques to reduce energy consumption. Building construction practices were tightened to eliminate heat gains or losses due to infiltration of outside air, and materials and constructional techniques offering improved thermal insulation were adopted. The introduction of energy efficient lighting saw power dissipation in fittings fall from levels in the vicinity of 40 W/m^2, to 15 W/m^2, and more recently to 11 W/m^2. Revised ventilation standards severely reduced minimum ventilation levels. The use of overcooling and reheat was discouraged, and in some legislatures was banned. Improved techniques were developed for the estimation of building thermal loads and energy consumption (Clarke, 1985), for optimal design of duct systems (Tsal et al., 1988a, b, 1990), and for the control of air conditioning systems (Fisk, 1981).

In terms of air conditioning equipment installed, probably the most notable response to the energy crisis was the rapid acceptance of the Variable Air Volume (VAV) system as a viable alternative to the simpler and hitherto almost universal Constant Air Volume (CAV) system. Such systems certainly offer the potential for significant energy savings by comparison with CAV systems. Fan power consumption is reduced at part load and, for multizone systems, over the whole operating range, while the use of overcooling and reheat, a necessity for load balancing in multizone CAV systems, is avoided. However, the operational experience with such systems has not always been entirely satisfactory. Air quality in particular has often left much to be desired (Gupta et al., 1987; Tamblyn, 1983). In retrospect it is fair to say that air quality was a casualty of the drive to reduce energy consumption. Revised ventilation standards, together with the virtual elimination of infiltration by tighter building construction and the current preference for sealed windows, resulted in ventilation levels which are now known to be unacceptably low (Fanger, 1988a). The reduction in zone sensible load through the adoption of more energy efficient lighting, without any accompanying reduction in latent load, has made much more difficult

the attainment of specified humidity levels[1]. In large part though, the failure of such systems to perform may be attributed to a lack of appreciation on the part of the designer of how important it is to design for part-load conditions, and of how to approach such design without reheat. Where the unacceptability of high part-load humidities has been recognised, strategies which have been devised to ameliorate the situation have almost universally reduced outdoor air levels to the minimum (inadequate) statutory requirement, or even lower through the subterfuge of maintaining a constant ratio of outdoor to return air in variable air volume systems. Alternatively, reheat has been reintroduced under the guise of "trim" or "terminal" reheat. The former measure merely compounds the air quality problem; the latter wastes energy.

The deterioration in indoor air quality has been exacerbated by the use of minimum cost plastic finishes, floor coverings and other furnishings, and by the installation of air conditioning systems selected on a lowest capital cost basis by entrepreneurial developers intent on recouping investments by on-selling buildings (and their still unidentified problems) on or before practical completion. The prevalence of sick buildings has stimulated the development and promotion of more rigid guidelines on minimum ventilation levels. These are exemplified by ASHRAE *Standard 62-1989* (ASHRAE, 1989b), which not only recommends minimum ventilation levels which exceed those previously advocated (although falling short of those recommended by Fanger (1988a[2]), but also advocate, among other measures, that system designers be required to document the basis for their designs and that means by which air flows can be measured in installed plant be provided. In addition, the standard stipulates that the ventilation air in the occupied space must not conflict with the comfort in the room, both from temperature and humidity standpoints, the relevant comfort standard being ASHRAE Standard 55-1989[3] (ASHRAE, 1989a). Such trends, together with a growing environmental awareness on the part of building owners, will undoubtedly encourage designers to seek solutions which simultaneously meet the seemingly contradictory requirements of energy efficiency and adherence to comfort and ventilation standards.

The requirements for healthy indoor air conditions may be summarised as follows: it is necessary to satisfy extant comfort standards (or more stringent specifications) in terms of both temperature and humidity, throughout the cooling period, while at the same time

---

1 To some extent, this reduction has been offset in recent years by the proliferation of personal computers. However, the growing popularity of laptop computers, and the transfer of their low power consumption technology to desktop machines, leads one to conclude that sensible heat ratios in occupied spaces will continue their downward trend.

2 ASHRAE *Standard 62-1989* recommends a minimum ventilation level of 10 lps/person for an office space (with or without smoking). This replaces ASHRAE *Standard 62-1981*, which permitted 2·5 L/s per person for the same office space, without smoking. In contrast, Fanger (1988a) recommends a minimum of 14 L/s per person for the office space without smoking, and 50 L/s per person with smoking.

3 The provisions of this standard are again more stringent than those of its predecessor (ASHRAE *Standard 55-1981*). In particular, an upper bound of 60% relative humidity is placed on the summer comfort region, as opposed to 12 g moisture/kg dry air for the previous standard.

providing sufficient ventilation to restrict the concentration of contaminants to an acceptable level. Furthermore, the energy expenditure to achieve this must be close to the minimum thermodynamic requirement. Stated in these terms, it seems almost obvious that the key determinant of success will be the efficiency of the dehumidification process. It is therefore surprising that the selection of the dehumidifier coil is usually relegated to the status of a secondary consideration in conventional design practice. Indeed, in many cases the designer's involvement with the coil selection process extends no further than the writing of a functional specification based on peak load conditions, the detailed task of (and contractual responsibility for) selection being left to the supplier of the dehumidifier. Two major factors contribute to this attitude. First, there is considerable misunderstanding within the design community of the heat and mass transfer processes occurring within the dehumidifier coil. Associated with this is a lack of appreciation of the manner in which a cooling coil responds to varying loads, and hence of the importance of critical part-load conditions in determining the overall suitability of a design solution.

Over the past twenty years, the air conditioning research group at the University of Adelaide has undertaken a thorough reexamination of the principles upon which current air conditioning practice has been built. The efforts of the group have focussed in particular on the poorly understood processes occurring in the dehumidifier coil. In an on-going research programme, a number of fundamental misconceptions regarding these processes have been exposed. The major findings of the research programme to date are presented in Luxton and Shaw (1991). These will be described in detail as appropriate, but may be summarised as follows:

i. Air flow through the cooling coil is laminar at all face velocities of interest. This runs contrary to the commonly held assumption (see for instance Threlkeld, 1970) that the flow is turbulent.

ii. On the air side, low face velocity, shallow coils with wide fin spacing promote dehumidification. The conventional philosophy favours high face velocity, deep coils with high fin densities where deep dehumidification is required.

iii. Maintaining a high coolant velocity will maintain a high coolant-side heat transfer coefficient, and hence keep the cool surface temperature low, and will also avoid a large rise in the temperature of the coolant through the coil. Both of these factors will maintain a high driving potential for dehumidification. Conventional practice favours a moderately low coolant velocity at peak with associated high temperature rise; coolant is throttled at part load, resulting in further deterioration in dehumidification potential.

iv. In normal circumstances, condensation on the coil is of a droplike character. In much of the literature (Threlkeld, 1970) filmwise condensation is assumed.

v. At equilibrium, the dehumidifier coil must remove heat and moisture from the air at the same rate as they are being added in the conditioned space. With space temperature and ambient conditions held constant, space humidity will adjust until equilibrium is attained. This process is known as the 'moisture staircase'.

The implications of these results for air conditioning practice have been realized in several areas. The most immediate outcome has been the development of a new design and selection method for dehumidifier coils and their associated controls. This method advocates the use of low air face velocity, together with the maintenance of a high coolant velocity across the complete operating range so to maintain a high potential for dehumidification. This design method, known as the *Low Face Velocity (LFV)/High Coolant Velocity (HCV)* method is described and compared with conventional practice, on a point-by-point basis, by Shaw and Luxton (1988b). The need to operate at lower face velocities than those common in conventional practice led to a reappraisal of the current standards for rating of cooling coils (ARI, 1981), and the proposal of an alternative method for rating wet coils. The new method does not rely on the more dubious assumptions embodied in ARI Standard 410-81 (Sekhar et al., 1988; Sekhar, 1990; Sekhar et al., 1991b). Used in conjunction with an experimental database assembled from tests in a unique closed circuit heat and mass transfer rig, this provides the basis for a new method of predicting coil performance. The importance of part-load conditions in the life-cycle of an air conditioning system has prompted an examination of the basis for specifying critical part-load conditions (Luxton et al., 1989). The various findings and developments originated by the group have been brought together in a methodology for air conditioning (Sekhar et al., 1989; Shaw et al., 1992).

In early 1992 the group undertook a series of design studies for commercial high-rise buildings in a tropical climate. A complementary study was subsequently undertaken in which the performances of a number of air conditioning strategies were compared in terms of the power consumption required to maintain a specified zone condition in a commercial office building as the outdoor air condition changed and as ventilation requirements were varied (Marshallsay et al., 1993). Experience gained with these studies has led to the development of a further air conditioning strategy, the High Driving Potential (HDP) outdoor air treatment system (Shaw et al., 1993) within which the LFV/HCV methodology becomes a design tool.

In addition, a means of controlling the space conditions for optimal human comfort has been developed (Shaw and Luxton, 1990). Used in conjunction with the previously mentioned technologies, this technique has the potential to effect considerable energy savings, while enhancing conditions for optimal human comfort.

1.2. The Present Study.

Any air conditioning design problem may be satisfied by an almost infinite number of solutions. An objective comparison of candidate solutions requires information regarding many factors, the most important of which are the first cost of the system, the quality of thermal environmental conditions according to relevant standards, and the running costs associated with achieving those conditions. A system offering the lowest first cost may be less than optimal if it fails to achieve specified zone thermal conditions, or if it does so only at the expense of excessive expenditure of energy. Indeed, depending on the relative weighting attached to the various factors above, such a solution may be judged inadmissible. An objective evaluation of candidate solutions in terms of thermal comfort

and running costs requires the prediction of system performance over a range of operating conditions. These should include not only conditions involving peak sensible zone loads, but also a number of critical part-load conditions which may in practice present a more challenging design problem than do the peak load conditions.

Evaluation of system performance using the design tools currently available is difficult. One of the most widely-used methods for selecting air conditioning coils is the Carrier method (Australian Construction Services, 1988a). Although this method has enjoyed widespread popularity for many years, the physical bases upon which it has been founded are questionable (McKenzie, 1991). In addition, *design practices based on the Carrier method tend to regard the zone conditions as given, rather than being an unknown for which a solution is sought.* The utility of this method for predicting system performance is, at best, limited. Computer programmes for predicting the performance of cooling coils for a specified set of operating conditions are available from a number of equipment manufacturers. For the most part these are based on the rating methods specified by ARI *Standard 410-81*, and while this method does suffer from certain shortcomings (Sekhar et al., 1988), reasonably accurate predictions may be expected under most circumstances. Steady-state zone conditions may be found using such a programme in conjunction with an iterative graphical procedure. For a single zone system using a simple coil, this process is tedious; with the additional complications of composite coils and multiple zones, the task of producing a solution becomes excessively time-consuming and error-prone.

In view of the rudimentary nature of the design tools currently available, there is an understandable reluctance on the part of designers to explore more than a very minimal subset of the solutions possible for a given problem, or to evaluate the performance of candidate solutions at other than peak load conditions for which there is usually a contractual obligation. Occasionally one or two part-load conditions may be checked. Indeed, in a great many cases first cost would seem to be the major, and sometimes the only criterion used to select one system in preference to another. Given the widespread use of computers in design offices throughout the world for applications as diverse as word-processing, inventory control, duct design and load calculation, there is a surprising lack of software packages available for the steady-state prediction of air conditioning system performance. The author is in fact unaware of any previous efforts to develop steady-state system simulation methodologies based on a firm physical foundation. Certainly, energy estimation packages such as BUNYIP (Moller and Wooldridge, 1985) and ESP (Clarke, 1985) do make claim to do just this. However, these are primarily building load estimation programmes which have been augmented by the use of rudimentary plant models. Modelling of the cooling coil in these packages is particularly lacking in adequacy for the task of predicting thermal environmental conditions. At the other end of the spectrum, one may cite studies such as that by Hamilton and Miller (1990), in which the emphasis has now been placed on the modelling of the air conditioning plant (the refrigeration components in particular). Again, the ability to model thermal environmental conditions is lacking.

The present study has been conceived against the background of the perceived inadequacies in current design methodology described above, and within the context of the University

of Adelaide research programme outlined earlier. The aims of the study may be summarized as follows:

i. To develop a robust set of tools to model the performance of a range of composite coil configurations, under arbitrarily specified operating conditions.

ii. Using these tools as a basis, to develop an operational model to predict the steady state performance of single and multizone air conditioning systems.

iii. For a number of representative case studies, to use the computational model as an exploratory tool to examine the performance of a series of candidate design solutions, and hence to make recommendations regarding a methodology for the design of air conditioning systems in an operational setting.

In view of the broad nature of the field under consideration, it has been necessary to impose certain restrictions on the scope of the work undertaken within this study. The nature and extent of these restrictions will become clear following a detailed reading of the body of the thesis. It should be borne in mind that the present study comprises one facet of a much broader research programme, and that the development of the model described in this study is an evolutionary process. The following restrictions are indicative of those currently in force, and also provide some indication of proposed future areas of research:

i. The model offers a high degree of flexibility in terms of the system geometries which may be represented. Prime consideration has been given to developing a model which is capable of representing geometries which currently have widespread commercial application, or which the group considered to offer particular promise in terms of advancing the state of the art. Of necessity, the current model does not offer sufficient flexibility to model *all* conventional system types. The model is currently incapable of representing dual-duct systems, for instance. In fact, the model does not currently offer any capability to represent heating cycles. The structure of the model and of the software code in which it finds expression have, however been designed to be flexible and it is envisaged that incorporation of these capabilities will represent a straightforward extension of the current methodology.

ii. Current work has been restricted to the simulation of chilled water systems. Direct Expansion (DX) systems are seen to be an important area of future application of the modelling, and plans are currently in hand to extend the capability of the model to handle such systems. While no case studies involving DX systems are presented in this thesis, some consideration is given to the factors involved in incorporating such a capability into the model.

Clearly, a design methodology may be considered to have achieved a degree of completeness when it is capable of guiding the user to an optimal solution. While it may be comforting to believe that a unique optimum design solution exists for any given air conditioning design problem, the solution space and the number of variables involved render the task of finding such an optimum well nigh impossible, even if all parties

involved can agree on the criteria by which optimality will be measured. At best, one can hope to achieve a near-optimal design using a set of proven design heuristics. Certainly, classical methods of mathematical and combinatorial optimization (Stoecker, 1989) are invaluable tools in the search for an optimum design, as are alternative techniques such as process integration (Linnhoff, 1993). Such tools will only be applied effectively if used in conjunction with a set of design heuristics to reduce the size of the solution space systematically. Such techniques are probably best implemented within the framework of a knowledge-based system. This presupposes the availability of adequate system models, both as a means of defining what is to be optimized, and as a tool to be used to develop the heuristics required to manage the optimization process. Previous attempts to codify air conditioning design methodology using artificial intelligence techniques (Ungbhakorn, 1989) have met with limited success[4]. One suspects that the absence of an underlying system model may have been a significant factor contributing to the disappointing results obtained using such codes. While the immediate aim of the present study is to design a simulation model for use within a traditional design framework, the longer term perspective envisages that the model, together with the design heuristics developed through experience with the model, will form the basis for a knowledge-based system which will guide the user towards a near optimal solution.

1.3. Scope of Work.

The scope of the work undertaken in this study is described below through an outline of the contents of the remaining chapters of the thesis.

The air conditioning design methodology described herein forms the basis for a software package known as ZEBRA[5], of which the system simulation model forms the core. In its entirety, it is envisaged that the ZEBRA package will comprise a comprehensive set of air conditioning design and project management tools implemented as a system of interacting tasks and databases in a multitasking computing environment. Chapter 2 provides a brief description of the ZEBRA system architecture. The system simulation software has been designed using the object-oriented design methodology (Booch, 1991), and implemented using the C++ programming language (Stroustrup, 1991). The object-oriented paradigm represents a significant departure from more traditional software analysis and design techniques, and has greatly influenced the perspective and abstractions which have been developed in this study. This in turn is reflected in the structure of the thesis. It is therefore appropriate to provide a brief discussion of the key features of the object-oriented design technique, and their implications for system modelling.

---

4 In January 1993, Professor Ungbhakorn advised the author that his expert system package had received a lukewarm reception from the industry, who regarded it as "better than nothing". He added that he felt that air conditioning design methodology was still insufficiently well understood to permit it to be implemented using an expert system.

5 ZEBRA is not an acronym for anything in particular. It merely fits within the menagerie of names which have emerged to describe software packages relating to air conditioning, e.g. DONKEY, BUNYIP, CHEETAH, CAMEL, etc.

Chapter 3 provides a brief review of current and recent standards relating to ventilation, human comfort and refrigerants. Essentially, this builds on the discussion initiated at the beginning of this chapter, and considers the implications of the various standards for air conditioning system design. A number of other studies relating to environmental quality are also considered.

The vast majority of the thermodynamic processes occurring within an air conditioning system involve moist air, and thus properly belong within the domain of psychrometrics. Chapter 4 presents a set of relationships to compute the fundamental psychrometric properties, together with various transport properties of moist air, such as viscosity and thermal conductivity, which are also of importance to this study. A general computational framework for psychrometric properties is established using the object-oriented paradigm. This is used extensively in the system simulation model.

Air conditioning systems are the subject of chapters 5 and 6, which are thus central to the entire thesis. The purpose of chapter 5 is to provide a brief review of the types of all-air systems available to meet a specified set of zone conditions, and to describe the thermodynamic processes which comprise the associated air conditioning cycles. Air conditioning systems are introduced with a brief review of the major types used in conventional practice, together with an analysis of their comparative advantages and shortcomings. This leads to a discussion of recommended practice, based on experience gained with the LFV/HCV and HDP systems which have been developed by the Adelaide University group.

Whereas chapter 5 has emphasized the physical aspects of system design, the intention in chapter 6 is to introduce a set of basic abstractions and algorithms to simulate air conditioning hardware, and thus to develop a system model possessing sufficient generality to simulate a broad range of system configurations, of both conventional and advanced design. The object of any air conditioning system is to maintain a specified set of conditions within the conditioned space or zone. The zone is thus a fundamental abstraction in our simulation, and it is fitting that it should be introduced prior to beginning our discussion of systems. The description of the zone centres on its properties, the processes occurring within the zone, and the data structures required to measure and control zone conditions. Most of the remaining major data structures are introduced at this point. Detailed discussion of a number of these is deferred to the following chapters. Attention here is directed towards those structures which are of central importance in configuring and controlling the system; the Air Handling Unit (AHU), the controller, and the various duct connectors. The chapter closes with a detailed description of a set of algorithms to determine the steady-state operating point of a multizone system in either VAV or CAV mode, when subject to control of zone dry-bulb temperature, and possibly of maximum zone humidity level.

The next four chapters present a more detailed discussion of modelling techniques and of design considerations for a number of types of system components.

Cooling coils are considered in chapter 7. The chapter commences by defining the coil geometry and dimensional data required to describe the range of coils which will be

considered in the following discussion. While the techniques presented are capable of generalization to handle a much wider range of coil dimensions and geometries, direct support is currently restricted to a range of plate fin-and-tube coils for which experimental data are available. Previous work undertaken by the Adelaide University group has exposed a number of common misunderstandings concerning the nature of the processes occurring within the cooling coil, especially in respect of the flow regimes and condensation mechanisms to be expected. A concise summary of these findings is presented as these are fundamental to the modelling procedure. Coil performance models fall into two general categories; differential and integral models. The former class of models are exemplified by the work of Hill and Jeter (1991) and Van Aken (1993), and characteristically attempt to provide a more or less detailed simulation of the physical processes occurring within the coil. Generally, models of the type used by Hill and Jeter are too computationally intensive to be comfortably integrated into the present framework, and are thus the subject of a very brief review. Van Aken simplified the process by defining separate flow regimes, based on observations by Gilbert (1987), and used the computational strategy of solving the time-dependent heat transfer equations to find the equilibrium operating state of the coil. His work is thoroughly documented in his thesis as referenced, and here is also briefly summarized in Chapter 7. Discussion of these computational models is followed by a detailed development of a dual-potential model for heat and mass transfer across a wetted surface, and its application to coils with varying proportions of wet and dry surface areas. A prototype algorithm for coil performance prediction based on the dual-potential model is presented, together with a discussion of a number of numerical considerations relevant to its solution. The discussion of this section makes no assumptions regarding the form of the air and coolant side heat and mass transfer coefficients; the prototype algorithm serves as a framework within which these coefficients will be calculated as necessary by calling appropriate, and as yet undefined 'black box' functions. The following two sections present alternative methods for calculating the air side heat and mass transfer coefficients for a given coil configuration, by recourse to suitable empirically-derived relationships. The first of these is the ARI method (ARI, 1981). It is shown that, in spite of the dubious nature of the physical foundations of this method, it does provide an adequate means of correlating dry coil data. Its performance in respect of wetted coil surfaces is not so satisfactory. A new coil rating method based on a more secure physical foundation, and developed specifically with wetted surface heat exchangers in mind, is presented as an alternative. This is the Adelaide University or AU method (Sekhar, 1990; Sekhar et al., 1991). Two further items must be specified to obtain a closed solution; a coolant side heat transfer coefficient, and a series of experimentally-derived correlations describing the air-side heat and mass transfer characteristics. Appropriate coolant-side heat transfer coefficients may be specified by reference to the literature. Discussion here is restricted to cooling coils using chilled water as a working fluid; some consideration of techniques for extending the methodology to handle DX coils is given in chapter 11. Algorithms are presented for estimating air and water pressure drops through a coil. Both the ARI and the AU method essentially provide a framework for the analysis and reduction of experimental coil performance data. Adelaide University have now assembled an extensive database of coil performance test data obtained using a unique closed-circuit heat and mass transfer rig. Algorithms and procedures which have been developed for the classification and analysis of experimental data, based on the ARI method for fully dry coils, and the AU method for fully wet coils, are described. This

chapter closes with a summary of the correlations which have been extracted from the existing database, together with some discussion of the range of experimental parameters required to complete the database.

Chapter 8 considers various issues relating to the simulation of composite coil banks. An algorithm is described which permits the experimental correlations, which have been exclusively derived using half-circuited coils, to be used for coils having arbitrary circuiting. This is followed by a description of a set of data structures and algorithms which will permit complex coil banks to be constructed using elementary coils as building blocks. A facility is provided to permit certain circuits to be deactivated, so increasing coolant velocity through the remaining tubes; this is a key strategy in the LFV/HCV design methodology (Sekhar et al., 1989).

In addition to the space loads, the cooling coil must offset loads resulting from heat transfer through the supply and return air duct walls, and from power dissipated by the supply air fan. Techniques for modelling these latter two loads are the subject of chapters 9 and 10 respectively. Power dissipation through the fan may be determined if the air volume flow rate and pressure drop through the duct system are known. In the class of problems considered here, volume flow rate will be known; the pressure drop may be calculated using the duct model outlined below.

In the perspective developed here, air conditioning ducts may be envisaged to be a specialized variant of a more general class of networks, which also includes pipe networks. Hierarchies of abstractions of this kind are easily handled using the inheritance mechanism supported by object-oriented programming languages such as C++. The intention here is to build up a set of abstractions and associated operations which will be suitable for modelling not only duct systems, but also pipe networks; characteristics and operations specific to either of the more specialized classes may be added by defining derived classes inheriting the properties of the base class. Fundamentally, a network may be considered to be composed of two types of entities; (pipe or duct) sections and fittings. Data structures are developed to support both of these types of entities. These are used to construct networks of arbitrary complexity. In most cases, duct systems may be adequately modelled as tree networks (Tsal et al., 1990). Algorithms are presented to compute the heat transfer into, and pressure losses through a tree network of known fluid velocity distribution. The techniques described in this and the following chapter are still under development, and while the concepts have been implemented in software, they have not yet been fully integrated into the system model.

The treatment in chapter 10 parallels that in the preceding chapter, in that both fans and pumps may be treated as specialized derivatives of a base class of rotodynamic machines. A data structure is presented which will permit a base fan characteristic to be represented using cubic splines. Using this as a basis, a set of algorithms are developed which will permit the fan operating point to be found for a given pressure and flow rate by application of the fan laws in conjunction with the base fan characteristic. The model developed readily generalizes to describe pump operation.

As mentioned previously, extension of the methodology to cover DX systems is as yet incomplete. However, chapter 11 covers a number of issues relating to the implementation of DX cycle simulation within the general framework established in earlier chapters. A detailed description is given of a class which provides facilities to calculate the thermodynamic and transport properties of a working fluid, either at a point, or following a thermodynamic process. This class is not self-contained, but is intended to form the basis of derived classes which will permit the application to specific fluids of the facilities provided in the base class. A hierarchy of derived classes which will model the properties of a broad range of refrigerants is presented. The next section describes the design of a set of classes to represent the hardware components of a refrigeration system. Again, full use is made of the C++ inheritance mechanism to provide a hierarchy of classes with those classes at the base of the hierarchy providing a more abstract representation, while the concrete details of the hardware components are filled out in the derived classes. The use of multiple inheritance to integrate the refrigeration cycle into the existing ZEBRA framework is also considered. The chapter closes with a brief consideration of a number of issues relating to the control of DX systems in rooftop applications.

The discussion so far has been restricted to issues relating to modelling the performance of a system for a particular set of loads. Chapter 12 seeks to set the ZEBRA methodology into a broader framework by considering the manner in which ZEBRA interfaces with the outside world, be it a user at a keyboard, or a cooperating process in a multitasking environment. At the beginning of a run ZEBRA reads the configuration of a system from a file, and constructs the objects required to construct a software representation of the system. This is done using a nested hierarchy of constructors, which permit systems of arbitrary complexity to be modelled. Once the system is constructed, ZEBRA is accessed by issuing commands in an elementary interpreted language, ZPL (Zebra Programming Language). This provides facilities to run simulations, perform basic editing tasks, change loads, query system state and terminate a sequence. The use of these facilities in simulating system performance over a programmed sequence of loads is considered.

Chapter 13 describes the application of the ZEBRA design methodology to a number of cases based on recent experience. The examples have been chosen to illustrate on the one hand how the design tools have been applied to solve otherwise intractable technical design problems under extremely demanding load conditions, and on the other hand to show how the methodology may be used to achieve design economies in more standard applications.

The final chapter summarizes the work undertaken to date, and indicates the future directions anticipated for this work.

Chapter 2. The ZEBRA Software Package.

2.1. Background to the Present Development.

During the 1970's a considerable number of computer-based design tools became available to the air conditioning industry. For the most part these tools fall into three major categories. On the one hand, packages such as CAMEL (Australian Construction Services, 1991) and TEMPER (Australian Construction Services, 1987) provide a means of estimating the space loads within a multizone building as a function of external climatic conditions and target indoor conditions, occupancy, lighting and equipment schedules, and building materials and construction techniques[6]. At the same time, a large number of software packages modelling or purporting to model specific equipment items and subsystems have become available. Most major equipment suppliers will make copies of proprietary coil selection programmes available to selected clients. Similarly, manufacturers' catalogues of fan, pump and filter characteristics are commonly available in computer-compatible form. Duct system design codes such as DONKEY (Australian Construction Services, 1988b; see also the review by Tsal and Adler, 1987) also fall into this second category. The third category comprises the broad class of energy estimation packages exemplified by codes such as BUNYIP (Moller and Wooldridge, 1985), ESP (Clarke, 1985), ESP-II (Australian Construction Services, 1985) and DOE-2 (U.S. Department of Energy, 1981). In essence, these codes are based on load estimation programmes which have been augmented with the incorporation of rudimentary plant models.

While the first generation of computer-based air conditioning design tools have certainly undergone an evolutionary process over the years, it is probably fair to say that the vast majority of such tools commonly found in design offices are, even in their most recent forms, based on software design methodologies and computing hardware and operating system considerations and constraints which are more appropriate to the 1970's than to current best practice. This is not an unusual situation. The design and implementation of a major software package represents a significant investment on the part of the developers, and there is a natural reluctance to contemplate complete redesign while it is still possible to graft new features onto the original, and to "bandaid" glaring defects and shortcomings. It is instructive to consider the constraints imposed on the future development of many of the major extant packages by the initial design decisions mentioned above:

i. Many of the major codes described above were originally designed to run in batch-orientated mode on mainframe computers. Subsequently, most have been modified

---

6 It is important to differentiate here between codes based on a steady-state heat transfer model, which in essence ignore the effects of thermal storage in the building fabric, and those which do attempt to model the resulting transients, by using finite-difference techniques, for instance. Ignoring the thermal history of the building will typically result in overestimation of the space loads during both cooling and heating periods (von Thun and Witte, 1991), leading to the selection of oversize equipment. Corrections are available for models of the former type (Australian Construction Services, 1988a). However, it is more acccurate to calculate the transient effects explicitly in the model, rather than applying somewhat dubious corrections *a posteriori*.

for use on workstations and personal computers, which has given the developers the opportunity to provide the user with a minimal, but in most cases severely limited facility to interact with the programme. Nevertheless, the primary means of feeding system specifications to such packages remains the construction of "card images", which clearly reflects on the vintage of the original designs. Menu-driven front ends are now available for a number of these packages. These do not however go very far towards remedying the basic shortcomings imposed by the initial design decisions.

ii. In most, and probably all cases, FORTRAN will have been the language of choice of the developer. This is not surprising, since in most cases the developer will have been the programmer, and computer languages other than FORTRAN were almost unknown among the scientific and engineering communities at this time. Even where professional programmers may have been employed on such projects, FORTRAN would almost inevitably have been the language of choice, since compilers for alternative languages, such as C, Pascal, ALGOL-68 and Simula enjoyed a comparatively limited distribution during this period. FORTRAN provides the user with only the most rudimentary tools for memory management, and for representing any but the most basic of data structures. This in turn severely curtails the programmer's ability to model complex systems in a manner which is at once efficient in the use of memory and transparent to other parties who may attempt to modify the code at a later date. It should be noted that the software design tools of this period were also severely limited in their ability to support large design projects (Booch, 1991).

iii. In some cases, the physical models and computational procedures used do not represent best practice, even by the standards of the day, as a result of compromises resulting from the relatively limited computational resources provided by hardware of this period, and by the need to minimize memory and CPU usage to reduce costs when using a centralized computing facility. The high cost of memory and address-space limitations associated with personal computers until recently meant that the growing use of PC-based versions of the software, in itself, provided little impetus to reexamine the suitability of the algorithms used, even though the placement of computing power on the desktop, rather than keeping it locked up in computing centres, has given the user the ability to experiment without fear of incurring a direct cost for the use of computer resources. However, the falling cost of computing power and memory, together with advances in compiler and operating system technologies have largely removed the rationale for oversimplifying the physical models and solution algorithms used in order to keep computing costs and resource usage within externally imposed limits. This is not to suggest that considerations of efficiency are no longer relevant. Placement of an ever-expanding quantity of computing power on the desktop gives the user the ability to explore a far greater range of candidate solutions than has hitherto been

possible[7]. However, the developer is now in a position to assess more objectively the tradeoffs which are possible between the need to minimize computer run times, and the need to provide a realistic and robust representation of the systems modelled.

iv. The computer-based design tools currently available to the designer directly offer no new insights into the design process, beyond that which results from the ability to trial more candidate solutions, and assess performance against more load conditions than would be possible from a purely manual approach. A design office will typically employ a disparate collection of programmes to automate various repetitive and time-consuming aspects of an existing design methodology on a piecemeal basis. Typical candidates for automation in this manner are the calculation of design loads, and computation of coil performance for a given set of coil-on conditions. Calculation of the resulting indoor thermal conditions is invariably performed manually, usually using graphical methods and a psychrometric chart. As was noted in the preceding chapter, this process is tedious and error-prone, particularly in the presence of multiple zones and composite coils. It is far more rewarding to take a holistic approach, in which the system is treated as a whole. That this has not hitherto been done should come as no surprise, given the background against which the currently available design tools were conceived. The limitations of the computing environments available in the 1970's, which have been described above, and the batch-orientated nature of the operating systems to which most engineers had access during this period, were conducive to a piecemeal approach to the automation of the design process, and indeed offered little scope for a more sophisticated approach. The growing availability of multitasking operating systems, such as UNIX and VAX/VMS, from the mid-1970's, potentially offered the ability to configure the various aspects of the design process as a set of cooperating tasks, each implementing one or more aspects of the design process. However, system programming for multitasking operating systems has tended to remain a specialist activity, the full potential of which has probably been little realised by engineers, with the exception of those engaged in developing real-time applications. Again, we note that the growing popularity of personal computers, which remained until recently essentially single-tasking, provided little incentive to develop a more holistic approach to automating the design process. The recent development, and expanding popularity of multitasking PC-based operating systems, such as Windows, offers new opportunities to develop a set of software tools conceived from the outset as a set of cooperating tasks, which may be configured in a number of ways, not simply to automate various aspects of the design process, but also to provide an integrated means of guiding the user towards a near-optimal approach to designing for a specific application.

---

[7] A search under the guidance of an optimization routine or an expert system, in particular, may involve a very large number of trial solutions.

2.2. An Overview of the ZEBRA Software Package.

It would be nice to be able to claim that the ZEBRA package was designed from the outset as a means of addressing the shortcomings listed above. This however was not the case. In its original form the development of the ZEBRA package and its immediate predecessor (Sekhar, 1990) was motivated by a desire to produce a user-friendly package implementing a coil prediction methodology developed at the University of Adelaide. As such, its objectives were probably little broader than those motivating most proprietary coil selection packages, although there was from the outset a notion that the software should be capable of a lot more, and that it should eventually guide the user towards an optimal coil selection, based on factors specific to the application. A significant step in this direction involved the development of an algorithm to solve the moisture staircase problem, thus solving for the thermal state of a single zone air conditioning system at equilibrium. This algorithm was subsequently extended to simulate a wide range of system types, incorporating single and multiple zones, and using simple or complex coil geometries. Use of the package in conjunction with other design tools on a number of design studies and consultancies lead to the recognition of the above-listed shortcomings in the art as currently practised, and thence to the conception of the ZEBRA package in its present, and projected future forms.

In its entirety ZEBRA is an ambitious project which will rely for its development on input from a number of sources. The projected package is shown schematically in figure 2.1.

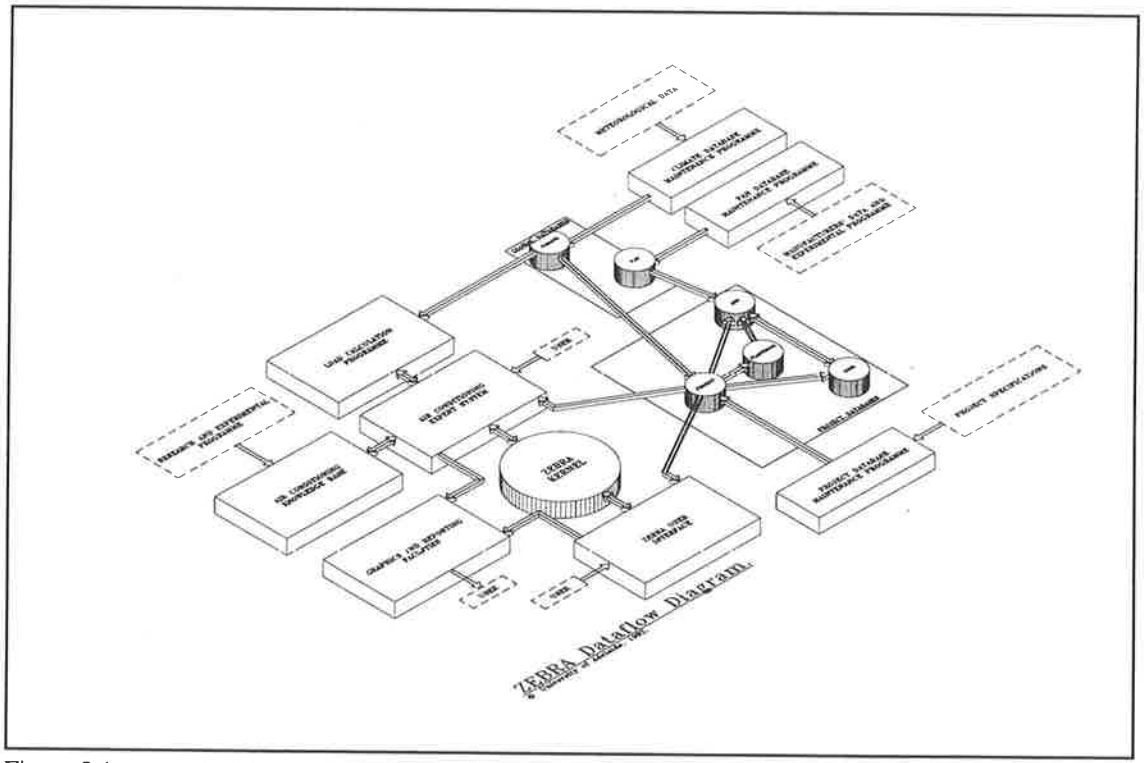


Figure 2.1. Architecture of the ZEBRA software package.

At the heart of the system is the Zebra Kernel (the **zk** process), which contains the system simulation software. The data structures and algorithms which currently form the basis for the services provided by the Zebra Kernel, together with a number intended for future

implementation, are described in detail in the later chapters of the thesis. The Zebra Kernel is self-contained in that it provides a full set of facilities to permit it to be run in standalone mode, including:

i. Assembly of a model of a system for simulation from a set of specifications supplied in a text file or compatible medium. The structure of the text file, and a description of a set of facilities to extract information from the file are described in section 2.4.

ii. Direct user interaction with the Zebra Kernel via a command-line interpreter (**CLI**). Commands and command sequences are entered in a basic interpreted language known as Zebra Programming Language (**ZPL**).

iii. Generation upon demand of reports describing the current state of the system, or a designated subset thereof.

Further description of these facilities in their present state will be found in chapter 12. It is not envisaged however that the Zebra Kernel will normally be run in standalone mode; this mode is provided primarily for use during the developmental phase. It is projected that the Zebra Kernel will run within a multitasking environment[8], and will provide a set of components which will offer interactive functions which parallel those described above, but which communicate with one or more cooperating processes, rather than communicating directly with an operator seated at a keyboard. These facilities will essentially comprise an enhancement of the existing facilities described above, with which they will share a great deal of common source code. Thus:

i. The system specifications may be communicated to the Zebra Kernel via some medium other than a text file. In particular, they may be communicated via a common block of memory. The intention is that:

   a. System specifications will be archived on disc in a format compatible with the input requirements of the Zebra Kernel.
   b. Specification sets may be read from archive by the Zebra User Interface (**ZUI**), which also provides facilities for editing existing specification sets, and creating new ones.
   c. A new **zk** process may be created by the Zebra User Interface, and a reference passed to the source of the current system specifications (a common memory area). These specifications are used by the **zk** process to build a system model.
   d. With the **zk** process loaded, it may be accessed via the command-line interpreter (see (ii) below) by the Zebra User Interface, or another cooperating process.

---

8 A 32-bit version of the Microsoft Windows® operating system is currently projected as the target host environment.

e. When a new set of system specifications is read from disc, or the Zebra User Interface is shut down, the user will be given the option of archiving any changes which have been made to the current set of system specifications. A currently loaded **zk** process will be terminated at this time, if it has not already been terminated.

ii. The command-line interpreter will accept and process commands entered at the keyboard. Alternatively, command sequences may be supplied from another source, such as a common memory area, or a disc file. The **CLI** accepts command-line sequences to:

a. Run a system simulation using the currently-specified system configuration and loads.
b. Perform a limited range of editing tasks on a loaded system.
c. Query the current state of the system or a designated portion thereof.
d. Identify a source (a file or common block of memory) from which a set of editing instructions and command-line sequences may be read and processed sequentially.
e. Terminate the **zk** process.

iii. The state of the system, or a portion thereof, may be queried and returned to another process for display in graphical or tabular form, or for further processing.

The Zebra User Interface will be the primary means of accessing the Zebra Kernel. However, it will not be the only task which interacts with the Zebra Kernel. Some of the other processes which will interact with a loaded **zk** process, and which are shown in figure 2.1, are described briefly below:

i. Graphics and Report Generation Facilities. The **zk** process will receive a request from the **ZUI** to transmit a block of data specifying the current state of the system, or a designated portion thereof. This block will be transmitted either directly to a cooperating process, or indirectly via the **ZUI**. The target destination may be a report generation facility, which will format the data into tabular or graphical form, or a spreadsheet programme. In the latter case, it may be appropriate for the **ZUI** to perform some intermediate translation on the data before transmitting it to the spreadsheet.

ii. Expert System. Having loaded a **zk** process, the user may request the **ZUI** to pass control to an expert system shell, which will maintain a dialogue with the **zk** process to solve a designated problem. The eventual aim is to use the expert system to automate the greater part of the system design process, guiding the user from the initial choice of a system configuration through to the detailed selection of components. The design process will be guided by interaction not only with the **zk** process, but also with other components of the Zebra system, including the load calculation programme, and the databases. In its final form, the expert system will thus implement all the functionality which resides in the **ZUI**, while providing the user with expert guidance in selecting and assessing system configurations and

components. The immediate aims are far more modest. The initial task selected for implementation using the expert system is that of identifying a cooling coil or set of coils which are in some sense optimal for a specific application, given that the overall system has been selected beforehand. This is currently a time-consuming procedure which relies heavily upon the experience and intuition of the designer to select a set of candidate coil configurations which are worthy of consideration.

iii. Load Calculation Programme. Most of the existing load calculation codes trace their ancestry back to the 1970's, and consequently suffer from the shortcomings described in section 2.1. That it is now timely to assess the suitability of existing codes, and the requirements for a new generation of load calculation codes, has been recognized by a number of bodies, including the CSIRO Division of Building, Construction and Engineering. From the viewpoint of the present endeavour, the major deficiency of the existing codes is that they offer a static picture of the zone loads, and no interactive capability. Thus, a set of loads must be calculated beforehand for a predetermined period (typically 24 hours), and manually transcribed into a form suitable for input to the Zebra Kernel. It is assumed that the dry-bulb temperature which will be maintained within the various zones is known *a priori*, and equal to the relevant thermostat set point. Consider the following situations:

a. An undersize coil is being trialled, and the target zone dry-bulb temperature cannot be satisfied by the cooling capacity available. The user is interested in finding out the thermal conditions which the coil can maintain within the zone. The problem is amenable to an iterative solution, provided the zone loads corresponding to a particular trial value for the dry-bulb temperature can be calculated.

b. A zone is subject to comfort integration control (Shaw and Luxton, 1990), and the target zone dry-bulb temperature can only conveniently be determined at run-time.

In both of the above cases there is a need for the **zk** process to interact dynamically with the load calculation programme, to specify a revised set of zone target temperatures to the load calculation programme, and to receive a revised estimate of the zone loads in response.

iv. Databases. The Zebra system interacts with a number of databases. These will probably be implemented using a commercial database package, and its associated support and programming tools. The databases are essentially of two types:

a. The *project* database will store an index to the system specification files and report files associated with a specific project. It may also serve as a repository for appropriate ancillary data, such as project management information.

b. *Global* databases contain information which is available to all projects based on need. The climate database contains climatic data specific to various geographic locations. The fan database stores fan characteristics for a number of commercial families of fans. Similar databases for other equipment items may also be implemented.

2.3. Programming Considerations.

The Zebra Kernel has been designed using object-oriented design methodology, and implemented using the C++ programming language. C++ is a language possessing features which support the object model, and thus make the language particularly suited to implementing applications designed according to the object-oriented paradigm. The purpose of the present section is to introduce a number of key features of the object model, and their implementation within the C++ programming language. This is desirable since it will assist the reader in understanding the rationale used to select and design the major abstractions involved in the air conditioning system model used by Zebra. The following discussion is necessarily brief, and can do scant justice to the subject material. For a detailed and readable account of the object-oriented design methodology the reader is referred to Booch (1991). The C++ programming language has been described in full by Ellis and Stroustrup (1990) and by Stroustrup (1991). It should be further added that the reader may find it advantageous to peruse the following subject matter in a cursory manner in the first instance, referring back to it for an explanation of points of interest as necessary.

The following sections serve two further purposes. A number of fundamental data structures are used to construct the simulation models described in later chapters. These will be familiar to computer scientists, but are likely to lie outside the domain of experience of the average engineer from a non-computing background. Finally, the opportunity is taken to describe some of the notation used to represent and describe data structures and algorithms in the remaining chapters of this thesis.

2.3.1. The Object Model.

Programming models are distinguished from one another by the fundamental abstractions they employ to represent entities within the problem space. In the object-oriented model the fundamental abstractions are *objects* and *classes*. These are complementary concepts. According to Booch (1991), the three distinguishing attributes of an object are:

- State
- Behaviour
- Identity

Every object is an instance of an underlying class, which defines the structure of an object, and the operations which may be performed upon it. These concepts are perhaps best illustrated by reference to a specific example. Suppose we have defined a class *Component*, which provides a representation of an equipment item within a refrigeration cycle. The representation chosen is sufficiently general to describe the fundamental thermodynamic characteristics of any component within a refrigeration cycle, be it a compressor, a condenser, an evaporator, a TX valve, or even a length of pipe. There are

reasons for providing this degree of generality in this class, which will be made clear later. Further details of the physical basis for this example will be found in chapter 11. The emphasis here is on using the example to illustrate various aspects of the object model. The class may be defined as follows[9]:

```
class Component
{
protected :
  FluidState Entering;   // Condition of fluid at entry to the component
  FluidState Leaving;    // Condition of fluid leaving the component
  double superheat;      // Superheat (K) of fluid at entry or exit
                         // (as appropriate)
  double subcooling;     // Subcooling (K) of fluid at entry or exit
                         // (as appropriate)
  double W;              // Work (kW) : +ve if done by the component
                         //           : -ve if done on the component
  double Q;              // Heat (kW) : +ve if transferred to component
                         //           : -ve if transferred from component
  double m;              // Mass flow rate (kg/s) of fluid through the
                         // component
  fluid* f;              // Pointer to working fluid
public :
  //-------------------------------------------------------------------
  // Constructors.
  //-------------------------------------------------------------------
  Component (fluid* f) { Component::f = f; }
  //-------------------------------------------------------------------
  // Destructor.
  //-------------------------------------------------------------------
  virtual ~Component () {}
  //-------------------------------------------------------------------
  // Selector functions.
  //-------------------------------------------------------------------
  fluid::state EnteringState () const { return Entering.state; }
  double EnteringTemperature () const { return Entering.T; }
  double EnteringPressure () const { return Entering.P; }
  double EnteringEnthalpy () const { return Entering.h; }
  double EnteringEntropy () const { return Entering.s; }
  double EnteringVolume () const { return Entering.v; }
  double EnteringQuality () const { return Entering.x; }
  fluid::state LeavingState () const { return Leaving.state; }
  double LeavingTemperature () const { return Leaving.T; }
  double LeavingPressure () const { return Leaving.P; }
  double LeavingEnthalpy () const { return Leaving.h; }
  double LeavingEntropy () const { return Leaving.s; }
  double LeavingVolume () const { return Leaving.v; }
  double LeavingQuality () const { return Leaving.x; }
  double Superheat () const { return superheat; }
  double Subcooling () const { return subcooling; }
  double Work () const { return W; }
  double Heat () const { return Q; }
  double MassFlow () const { return m; }
  //-------------------------------------------------------------------
  // Set mass flow of working fluid through the component.
  //-------------------------------------------------------------------
  void MassFlow (const double m) { Component::m = m; }
  //-------------------------------------------------------------------
  // Set entering state to current state of working fluid.
  //-------------------------------------------------------------------
  void UpdateEnteringState ();
```

---

[9] Throughout, segments of source code, and C++ identifiers are presented in `Courier` font. In C++, all text lying between the symbol `//` and the end of the line is treated as a comment.

```
  //-------------------------------------------------------------------
  // Set leaving state to current state of working fluid.
  //-------------------------------------------------------------------
  void UpdateLeavingState ();
  //-------------------------------------------------------------------
  // Virtual function describing process whereby working fluid is taken
  // from entering to leaving condition.
  //-------------------------------------------------------------------
  virtual void Process () = 0;
};
```

The ordering of entities within a class is arbitrary. The structure of the class gains clarity however, if the various entities are grouped in the manner shown. Thus, the first part of the class defines a number of *member variables*. These variables, together with the values currently assigned to them, determine the *state* of an object belonging to the class. The latter part of the class defines a set of member functions, which operate on the member variables of an object, or upon other objects, and thus determine the *behaviour* of the object. An object assumes a unique *identity* when it is *declared.*

Upon examining the member variables of this class, it will be seen that three of these belong to classes which are defined elsewhere. Member variables `Entering` and `Leaving` belong to class ***FluidState***. This is a fairly simple class designed to store a set of variables describing the thermodynamic state of a working fluid at a point, and is defined thus:

```
struct FluidState
{
  fluid::state state;      // State of fluid
  double T;                // Temperature, K
  double P;                // Pressure, kPa
  double h;                // Enthalpy, kJ/kg
  double s;                // Entropy, kJ/kg.K
  double v;                // Specific volume, m^3/kg
  double x;                // Quality
  FluidState () { state = fluid::UNDEFINED; }
  ~FluidState () {}
};
```

In the above, variable `state` is of an enumerated type `fluid::state`, defined as

```
enum state { UNDEFINED, SUBCOOLED_LIQUID, SATURATED_LIQUID, WET_VAPOUR,
             SATURATED_VAPOUR, SUPERHEATED_VAPOUR };
```

As the notation indicates, this type is defined in class ***fluid***. Variable `f` in class ***Component*** is a *pointer* to a variable of class ***fluid***, which is a base class for a hierarchy of classes defining member variables describing the current thermodynamic state and transport properties of a working fluid, and member functions to manipulate those variables as the fluid is subjected to a thermodynamic process. A full description of this hierarchy of classes will be found in chapter 11. It is now clear how we may use this class to simulate a refrigeration cycle. Suppose that we have an object of class ***fluid*** (or more correctly, a class derived from class ***fluid***,) representing the working fluid in the refrigeration cycle. Suppose further that we have a set of objects of class ***Component*** which our code can visit in sequence[10], and that each of these objects has access through

[10] They may for instance be stored in an array, or one of the data structures described in section 2.3.2.

its member variable `f` to the unique object representing the working fluid. Then, as each object is visited, the following operations are performed:

```
:
Component* c;
:
c->UpdateEnteringState ();
c->Process ();
c->UpdateLeavingState ();
:
```

In this sequence of operations, the component referenced by pointer `c` first has the thermodynamic state of the entering fluid updated to reflect the current state of the working fluid. The state of the working fluid is then modified in accordance with the thermodynamic process to which it will be subjected within the component. This is achieved by invoking member function `Process`. Finally, member variable `Leaving` is updated to reflect the now current state of the working fluid.

The *interface* for class ***Component*** defines member functions `UpdateEnteringState` and `UpdateLeavingState`, but does not implement them. Thus the *implementation* must be provided elsewhere. Member function `UpdateEnteringState` may be implemented using the following code:

```
void Component::UpdateEnteringState ()
{
  Leaving.state = f->CurrentState ();
  Leaving.T = f->Temperature ();
  Leaving.P = f->Pressure ();
  Leaving.h = f->Enthalpy ();
  Leaving.s = f->Entropy ();
  Leaving.v = f->SpecificVolume ();
  if (Leaving.state == SATURATED_LIQUID ||
      Leaving.state == WET_VAPOUR ||
      Leaving.state == SATURATED_VAPOUR)
    Leaving.x = f->Quality ();
}
```

and similarly for member function `UpdateLeavingState`. Since the implementation of this function is external to the *scope* within which it is defined, the defining scope must be explicitly indicated. This is done by prefixing the member name with the class name followed by `::`. This means of identifying the defining scope for member variables and functions will be encountered elsewhere.

No implementation is provided for member function `Process`, which is a *pure virtual* function, as indicated by the fact that its definition is prefixed with the keyword **virtual**, and equated to zero. The presence of a pure virtual function makes class ***Component*** an *abstract* class, and objects of this class cannot be created; an attempt to do so will give rise to a compilation error. Member function `Process` has been declared to be a pure virtual function since no meaningful default thermodynamic process can be ascribed to a generic component. The purpose of class ***Component*** is to provide an abstraction describing the state and behaviour common to all components in a thermodynamic process, and to serve as a *base class* or *superclass* from which more specialized classes can be derived. Such classes are known as *derived classes* or *subclasses*. Suppose that we had a need to develop

a class to describe the characteristics of a (generic) compressor. This could most efficiently be done by defining it as a subclass of class *Component*:

```
class Compressor : public Component
{
  :
public :
  :
  virtual void Process ();
  :
};
```

Member function `Process` is no longer a pure virtual function (it is implemented externally from the scope of the outline interface above), and objects of this derived class can be created. Subject to rules which will be defined below, class *Compressor* has access to the attributes of its superclass, and these need not be redefined in the derived class. A derived class is said to *inherit* the attributes of its base class. The fact that member function `Process` has been declared to be a **virtual** member function means that it can be redefined by a function having the same definition in a base class. Class *Compressor* may in turn serve as a base class from which further classes may be derived, and these will inherit not only the attributes of class *Compressor*, but also the attributes of its base class(es). Thus we may develop a class *ScrewCompressor* which represents a further specialization of the concepts developed so far, which is derived from class *Compressor*, and which redefines **virtual** member function `Process`. Thus, member function `Process` as defined within class *Compressor* might perform an isentropic compression on the working fluid. Within the derived class we would probably like to modify the process to model more accurately the behaviour of the fluid in a real compressor, and this might conceivably be achieved by an algorithm which performs an isentropic compression upon the working fluid to achieve a first estimate of the final state, and then seeks to refine the estimate according to some iterative procedure. Since class *Compressor* is visible to class *ScrewCompressor*, the latter class is able to access the implementation of member function `Process` defined in the former using the scope definition rule described above:

```
void ScrewCompressor::Process ()
{
   // Make initial estimate of final fluid state.
  Compressor::Process ();
  // Refine estimate.
  :
}
```

Even though it is not possible to create an object belonging to an abstract class, objects belonging to classes derived from an abstract base class may be referenced by a pointer of the base type, as has been done in some of the code fragments above. The virtual calling mechanism ensures that the correct version of a virtual function will be invoked at run time for objects referenced in this manner.

In the examples above, classes have been derived from a single base class. It is also possible for a class to be derived from more than one base class. This mechanism is known as *multiple inheritance*, and is occasionally useful. Assume that we have a class *Evaporator* (derived from *Component*), and a class *Coil*. This latter class simulates the state and behaviour of a cooling coil, but without reference to the state and behaviour of the coolant; this must be specified in a derived class. Clearly, if we were to develop a class *AirCooledEvaporator* to be used within the context of our refrigeration cycle model, it would be desirable for it to inherit the attributes not only of class *Evaporator*, but also of class *Coil*. This can be achieved by defining the interface thus:

```
class AirCooledEvaporator : public Evaporator, public Coil
{
  :
};
```

Objects declared within a C++ programme are by default *automatic* variables. They come into existence, and are optionally initialized, when programme execution moves into the scope of their declaration, and they are destroyed when they go out of scope. It would be inappropriate to define precisely in this discussion what constitutes the scope of a variable. Suffice to say that the scope may be a function, or a block (parenthized thus `{ ... }`) within a function; a more precise definition will be found in either of the references cited earlier (Ellis and Stroustrup, 1990; Stroustrup; 1991). When programme execution visits the scope of a particular automatic variable, no assumptions can be made about the state of the variable, apart from those arising from the explicit initialization of the variable. In effect, a new instance of the variable is created (and initialized) each time the scope is visited. Variables may be given a life which transcends the periods during which they are in scope by declaring them to be **static**. Static variables are created and initialized when programme execution is initiated, and are destroyed when execution terminates. They preserve state while out of scope, but are only accessible while in scope. A third mechanism exists for creating objects and other variables; they may be explicitly created using the **new** keyword, thus

```
Compressor* c = new Compressor ();
```

and destroyed using the **delete** keyword

```
delete c;
```

Objects created in this manner remain in existence until they are explicitly deleted, or until the programme terminates. They are also accessible for as long as a pointer to them is in scope.

Regardless of what mechanism is used to create an object, a special member function known as a *constructor* is invoked when the object is created, and a *destructor* is invoked when it is destroyed. A constructor is a function having the same name as the class to which it belongs. A constructor may take arguments, but may not return a value. If a constructor is not explicitly defined in the interface of the class, a default constructor is supplied by the compiler. This will create an uninitialized object of the desired class. An explicitly defined constructor will usually at least initialize the major member variables to

appropriate default values. In the case of a complex class the constructor may be called upon to perform a far more significant set of operations, which may for example include the dynamic allocation of memory for data structures, opening and reading from disc files, and initializing displays or other computer peripherals. Any class may define several alternative constructors differing in their argument lists.

A destructor takes the name of the class to which it belongs, prefixed with the symbol ~. A destructor for a class is unique, takes no arguments, and returns no value. A destructor may however be declared **virtual**. As in the case of a constructor, a default destructor will be supplied by the compiler if none is explicitly defined. The default destructor simply deallocates the memory occupied by the object. A destructor should in most cases aim to return the system to the state it was in before the object was created, and may for instance be called upon to invoke destructors for data structures dynamically created by the object, and close disc files opened by the object. Simple constructors and destructors are defined within the interfaces for class *Component* and class *FluidState*.

Booch (1991) defines the following three categories, to which most of the remaining member functions of a class will belong:

"•	Modifier	An operation that alters the state of an object; a writer or accessor operation
•	Selector	An operation that accesses the state of an object, but does not alter the state; a reader operation
•	Iterator	An operation that permits all parts of an object to be accessed in some well-defined order"

The programmer may control access to the member variables and functions of a class by invoking the keywords **public**, **protected** or **private**. The type of access which applies to a particular member is determined by the last of the above keywords which proceed it in the interface. Thus, in the definition for class *Component*, the member variables are defined to be **protected**, while the member functions are **public**. By default, members of a class defined using the keyword **class** are **private**, while members of a class defined using the keyword **struct** are **public**. The two types of class are identical in all other respects. The types of access implied by the three keywords listed above are as follows:

public : Any user-written code may access the member.

protected : The member may only be accessed by member functions of the class, or member functions of classes derived from the class.

private : Access is restricted to member functions of the class.

In the case of class *Component* and its derived classes, user-written code may read, but not alter the state of an object, using a supplied set of selector functions. The state of the object may only be altered in a prescribed manner by invoking the member functions

`UpdateEnteringState`, `UpdateLeavingState`, `Process` or `MassFlow`[11]. Class ***FluidState*** is intended solely for internal use by class ***Component***, and there is no good reason to restrict access to its members.

Finally, it should be borne in mind that, while C++ provides a comprehensive set of tools to support the object-oriented programming paradigm, it is derived from, and may itself be used as a procedural programming language. In this respect it differs from some other languages, such as Smalltalk, in which every declaration and action must be associated with some object. C++ in fact tends to encourage a programming style which combines elements of the object-oriented model, and elements of the procedural model. It is thus possible for functions to exist outside the class structure of the model. Such functions are referred to as *free subprogrammes*.

The above necessarily comprises a very brief introduction to some of the key concepts of object-oriented programming as implemented using the programming language C++. For a far more comprehensive treatment, the reader is urged to consult the cited references.

2.3.2. Fundamental Data Structures.

A *container type* is a data structure designed to store and provide access to a set of objects of the same, or related types. Container types are used extensively within the Zebra Kernel, both as the fundamental building blocks for more sophisticated data structures, and as a means of obtaining sequential or indexed access to the various component objects of a simulation model. In an object-oriented programming language it is convenient to implement container types as classes. A particular application may require the use of the same fundamental data structure to hold objects of many different types. In a traditional programming language this would mean replicating what is essentially the same source code many times over. The situation can be alleviated using inheritance. C++ offers a feature known as the *template* which reduces the task to one of writing the required class once only (Stroustrup, 1990). A template class is a class, the interface for which takes one or more types as arguments. During compilation classes are generated to satisfy each of the argument lists used. The mechanisms by which the compiler achieves this are of little or no concern to the programmer. Judicious use of inheritance in combination with templates will reduce the binary code generated to a minimum.

The following discussion briefly describes the distinguishing characteristics of three types of container class which will be encountered in subsequent chapters. The interfaces for these classes are described as a means of defining the operations which may be performed upon them, but the reader is referred elsewhere for details of the implementation. The opportunity is also taken to elucidate certain additional features of C++.

---

[11] Note that there are two member functions bearing the name `MassFlow`. Since they accept argument list of different types, they are different functions. The mechanism whereby C++ permits the same name to be shared by different functions is known as *overloading*.

i. Singly-linked Lists. A singly-linked list contains a sequence of nodes connected in one direction (the forward direction) by a pointer to the next node on the list (figure 2.2). Each node contains an object or variable of the designated type. Two

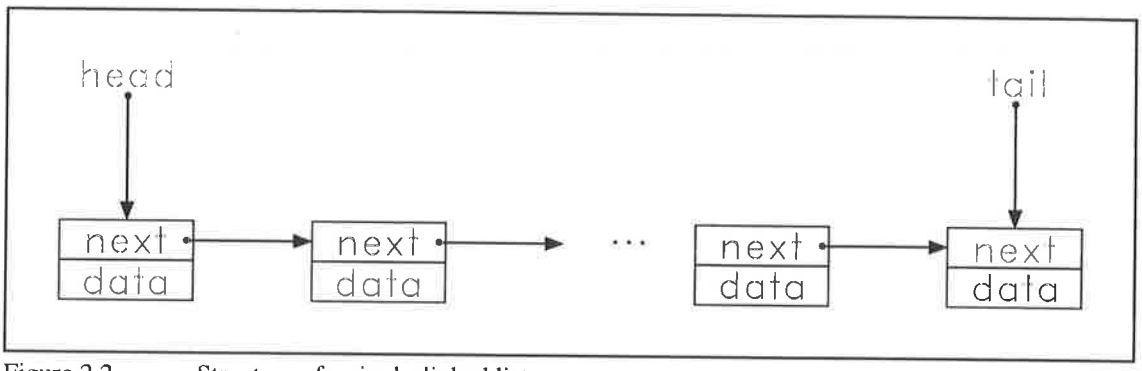


Figure 2.2. Structure of a singly-linked list.

special nodes are designated, the *head* and the *tail*. The public interface for a template class to represent a singly-linked list may be written thus:

```
template <class T> class Slist : private slist_base
{
  friend class Slist_iter<T>;
public :
  void insert (const T& a);
  void append (const T& a);
  T get ();
  T& head ();
  T& current ();
  T& first ();
  T& next ();
  void clear ();
  int NumberOfElements ();
  int Isempty ();
};
```

The implementation follows that given by Stroustrup (1990). Two points about the implementation need to be explained here:

a. The template class is derived from a base class *slist_base*, which implements the operations common to all derived classes.

b. The base class maintains a current pointer which references the currently addressed node.

The template class takes one argument (`T`), which denotes the type of object or variable the class will contain. A class to contain a singly-linked list of integers may thus be declared as

```
Slist<int> l;
```

The member functions defined in the interface perform the following operations:

`insert` inserts a node containing an object of the designated type at the head of the list.

`append`	appends a node containing an object of the designated type at the tail of the list.
`get`	returns the object contained in the node at the head of the list, and deletes the node.
`head`	returns a reference to the object contained in the node at the head of the list. The current pointer is left where it is.
`current`	returns a reference to the object contained in the node addressed by the current pointer.
`first`	moves the current pointer to the head of the list, and returns a reference to the object contained in that node.
`next`	moves the current pointer to the next node on the list, and returns a reference to the object contained in that node.
`clear`	removes all nodes from the list.

For an object of this class an attempt to move the current pointer beyond the end of the list, or to `get` an element from an empty list, will result in an error.

An *iterator* class is provided to traverse a list, and is defined by the template

```
template <class T> class Slist_iter : private slist_base_iter
{
public :
  Slist_iter (Slist<T>& s) : slist_base_iter (s) {}
  T operator () ();
};
```

Referring back to the interface for class ***Slist***, it will be seen that this latter class is declared to be a **friend** of ***Slist***. A friend class or function has access to the private parts of the class which declares it to be a friend. This class defines two member functions. The first is a constructor which takes as an argument a reference to the list upon which the iterator will operate. The second defines the operator `()` so that it will return the value of the object contained in the next node on the list (when first invoked, it accesses the head of the list). This feature of C++ is known as *operator overloading*, and permits the operation of a range of standard operators, including the mathematical operators, logical operators, the subscripting operator `[]`, the function call operator `()`, and a number of others, to be redefined in respect of their action upon objects of a particular class. The manner in which a list and its iterator cooperate to traverse a list is illustrated by the following code fragment:

```
Slist<int> l;
// Build the list.
...
// Traverse the list.
Slist_iter<int> t (l);
for (int i = 0; i < l.NumberOfElements (); i++)
{
  int j = t ();
  // Manipulate element j.
  ...
}
```

For small types such as integers, it is straightforward to store the elements themselves on the list. For complex objects it is more convenient to store pointers to the objects on the list. Class *Splist* serves this purpose. Its interface, and that of its iterator, are defined as follows:

```
template <class T> class Splist : private slist_base
{
  friend class Splist_iter<T>;
public :
  void insert (T* p);
  void append (T* p);
  void clear ();
  T* get ();
  T* head ();
  T* current ();
  T* first ();
  T* next ();
  T* NumberOfElements ();
  T* Isempty ();
};

template <class T> class Splist_iter : private slist_base_iter
{
public :
  Splist_iter (Splist<T>& s) : slist_base_iter (s) {}
  T* operator () ();
};
```

The actions performed by the member functions of the above classes essentially parallel those performed by the corresponding member functions of the classes previously defined. Note that in this latter case, an attempt to move the current pointer beyond the end of the list, or to `get` from an empty list, will return a null pointer.

ii. Doubly-linked Lists. The doubly-linked list (figure 2.3) differs from its singly-linked counterpart in that each node maintains a link not only forward to the next node on the list, but also backwards to the preceding node, thus providing the ability to traverse the list in either direction. The implementation used by the Zebra Kernel follows that of Hansen (1987), but has been modified considerably to take advantage of templates. In most respects, the implementation parallels that described above for a singly-linked list, and the discussion here will be kept brief, and will be restricted to a consideration of the public interface for a class to represent a doubly-linked list of pointers:

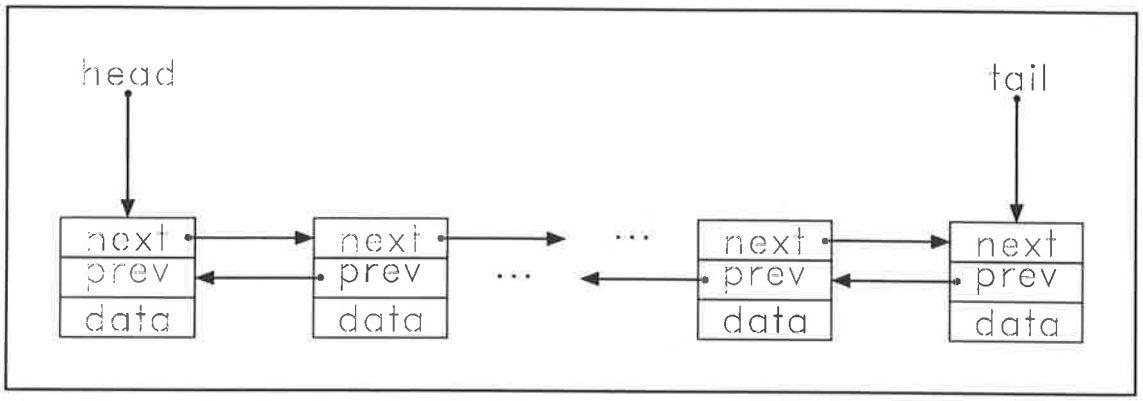


Figure 2.3. Structure of a doubly-linked list.

```
template <class T> class Dplist : private dlist_base
{
public :
  void clear ();
  int isempty ();
  void reset ();
  int NumberOfElements ();
  int next (T*& p);
  int prev (T*& p);
  int get (T*& p);
  int getnext (T*& p);
  int getprev (T*& p);
  void append (T* p);
  void insert (T* p);
  void appendhere (T* p);
  void inserthere (T* p);
};
```

Insofar as they have no counterpart in the definition for a singly-linked list, or their purpose differs from that of the corresponding function in the interface, the functions defined in the above interface perform the following functions:

`reset` resets the current pointer to undefined (null).

`next` moves the current pointer forward to the next node, if the current pointer is defined; otherwise moves it to the head of the list. If the operation would move the current pointer beyond the end of the list, it becomes undefined, and a null pointer is returned. Otherwise, the pointer contained in the node referenced by the current pointer is returned.

`prev` moves the current pointer backward to the preceding node, if the current pointer is defined; otherwise moves it to the tail of the list. If the operation would move the current pointer beyond the front of the list, it becomes undefined, and a null pointer is returned. Otherwise, the pointer contained in the node referenced by the current pointer is returned.

`get` operates on the node addressed by the current pointer, if the pointer is defined. Otherwise operates on the node at the head of the list.

The pointer contained in the selected node is returned, and the node removed from the list and deleted. If the selected node is at the current pointer, the current pointer is moved to the succeeding node.

`getnext` operates on the node succeeding the one addressed by the current pointer. If the current pointer is already at the head of the list, returns a null value. If the pointer is not defined, selects the head of the list. The pointer contained in the selected node is returned, and the node removed from the list and deleted.

`getprev` operates on the node preceding the one addressed by the current pointer. If the current pointer is already at the tail of the list, returns a null value. If the pointer is not defined, selects the tail of the list. The pointer contained in the selected node is returned, and the node removed from the list and deleted.

`appendhere` appends a node after the current pointer, if it is defined. If it is not, inserts the node at the tail of the list.

`inserthere` inserts a node before the current pointer, if it is defined. If it is not, inserts the node at the head of the list.

iii. Associative Arrays. An associative array, also known as a *dictionary* or a *map*, generalizes the concept of an array, as implemented as a standard tool in most programming languages. An element of an array is defined by two entities, a *key* or *index*, and a *value*. To declare an array it is necessary to reserve a contiguous block of memory spanning the range of values that the index may take. In the case of an associative array, elements are allocated only as needed. Furthermore, the key need not be an integer. It can be of any type upon which an ordering can be imposed, and with the ability of C++ to overload the logical operators, most types can be used as a key. The value furthermore, can be of an arbitrary type. The associative array therefore becomes an ideal candidate for implementation as a template class. The elements of the array must be stored in some data structure which permits them to be sorted according to the key. The implementation described by Stroustrup (1990) is based on a doubly-linked list. In the implementation used by the Zebra Kernel, a binary search tree is used as the underlying data structure. From the point of view of the user, the syntax of the interface is almost identical. Assume the existence of a class *GenericBinaryTree*, which implements a binary tree, the elements of which are of arbitrary type, a class *GenericNode*, which implements a node of the generic tree, and class *GenericTreeIterator*, which is an iterator for the generic tree. Then we may define a node containing one element of an associative array:

```
template <class K, class V> class MNode : private GenericNode
{
  friend class Map<K, V>;
  friend class MapIter<K, V>;
  const K key;
  V value;
public :
  MNode (const K& k, const V& v) : key (k), value (v) {}
  virtual ~MNode ();
};
```

The above template takes two arguments. K is the type of the key, and V is the type of the value. In much simplified form, the interface for the template class implementing the associative array may be written:

```
template <class K, class V> class Map : private GenericBinaryTree
{
  friend class MapIter<K, V>;
  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  // Private member variables.
  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  V def_val;              // Default value
  K def_key;              // Default key
  MNode<K, V>* current;   // Pointer to the current node
public :
  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  // Private member functions.
  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  //--------------------------------------------------------------
  // Constructor for class Map<class K, class V>.
  //--------------------------------------------------------------
  Map (const K& k, const V& v) : def_key (k), def_val (v) {}
  //--------------------------------------------------------------
  // Destructor for class Map<class K, class V>.
  //--------------------------------------------------------------
  virtual ~Map ();
  //--------------------------------------------------------------
  // Indexing operator.  Returns a reference to the value of an
  // element with the specified key.
  //--------------------------------------------------------------
  V& operator[] (const K& k);
  //--------------------------------------------------------------
  // Clear the array.
  //--------------------------------------------------------------
  void clear ();
  //--------------------------------------------------------------
  // Remove a specified element from the array.
  //--------------------------------------------------------------
  void remove (const K& k);
  //--------------------------------------------------------------
  // Return the number of elements in the array.
  //--------------------------------------------------------------
  int size ();
  //--------------------------------------------------------------
  // Array membership.  Returns 1 if specified key in array,
  // 0 otherwise.
  //--------------------------------------------------------------
  int in (const K& k);
  //--------------------------------------------------------------
  // Iteration functions.
  //--------------------------------------------------------------
  MapIter<K, V> first ();
  MapIter<K, V> last ();
};
```

For the most part, the member functions in the above interface are self-explanatory. The function which is of real interest is the indexing operator, which is derived by overloading the standard C subscripting operator. This function returns a reference to the value of an element with the key specified by the argument, if such an element is found in the array. If no such element is found, an element having this key and the default value is dynamically created, and a reference to the value is returned. In this way, an array element may be specified on the left-hand side of an assignment operator. The syntax of usage of an associative array is thus identical with that of a standard array. Suppose that the zones in a building are referenced according to a name stored in a text string, and that we wish to store the sensible load values in an array accessed by the appropriate set of keys. This may be achieved by a set of statements of the following form:

```
:
Map<String, double> SensibleHeat ("", 0.0);
:
SensibleHeat[Floor1NorthEast] = 24.0;
SensibleHeat[Floor1SouthWest] = 22.0;
:
```

The situation will frequently arise where we wish to traverse part, or all of an asssociative array in sequence according to the key. An iterator is provided for this purpose:

```
template <class K, class V>
class MapIter : public GenericTreeIterator
{
  friend class Map<K, V>;
public :
  //-------------------------------------------------------------
  // Constructor for class MapIter<class K, class V>.
  //-------------------------------------------------------------
  MapIter (Map<K, V>& m) : GenericTreeIterator (&m) {}
  //-------------------------------------------------------------
  // Destructor for class MapIter<class K, class V>.
  //-------------------------------------------------------------
  ~MapIter () {}
  //-------------------------------------------------------------
  // Access key and data.
  //-------------------------------------------------------------
  const K& key ();
  V& value ();
  //-------------------------------------------------------------
  // Increment and decrement operators.
  //-------------------------------------------------------------
  MapIter& operator-- ();      // prefix
  void operator-- (int);       // postfix
  MapIter& operator++ ();      // prefix
  void operator++ (int);       // postfix
};
```

The constructor for this class takes as an argument a reference to the array upon which the iterator will operate. Iteration is performed using the increment and decrement operators. The prefix operators return a reference to the iterator, or a null reference if the traversal attempts to proceed beyond the end of the array. They may thus be used to detect the end of a traversal sequence. Continuing our

previous example, the sensible space loads for a set of zones may be printed to the screen using following code:

```
:
for (MapIter<String, double> p (SensibleHeat); p; ++p)
  cout << p.key () << '\t' << p.value () << '\n';
:
```

which will produce the following output:

```
Floor1NorthEast    24.0
Floor1SouthWest    22.0
```

Finally, note that the interface for class *Map* defines two iterators, one initailized to point to the first element in the array, and one to point to the last element.

2.3.3. Notation.

The present thesis is concerned to a large extent with describing the manner in which software has been configured to represent and model a class of physical systems. The suitability of a notational scheme to describe a software system is determined largely by by the perspective chosen by the developer in designing the system. The various programming methodologies developed over the years have spawned a plethora of notational schemes (Yourdon, 1989).

Booch (1991) presents a comprehensive set of notational tools to support the object-oriented programming paradigm. The tools recommended by Booch provide the user with a means of describing a software system from a number of complementary, but mutually orthogonal perspectives. In the current work we are concerned mainly with describing the class structures of software systems. Booch's class diagrams serve this purpose admirably.

At the micro level we are concerned with describing the algorithms implemented within the various class member functions and free subprogrammes. A *programme description language* (PDL) or *pseudocode* scheme has been developed for this purpose. Some use has also been made of the much-maligned flow chart[12].

The characteristics of the notational schemes used in this work are elucidated in the following.

2.3.3.1. Class Diagrams.

In the notation devised by Booch, a class is represented by the icon shown in figure 2.4. Booch classifies the relationships between classes into four types; inheritance relationships, using relationships, instantiation relationships, and metaclass relationships. In the present work, the three former types will be of interest to us. A subset of the notation devised by

---

[12] For a reassessment of the flow chart, see Yourdon (1989).

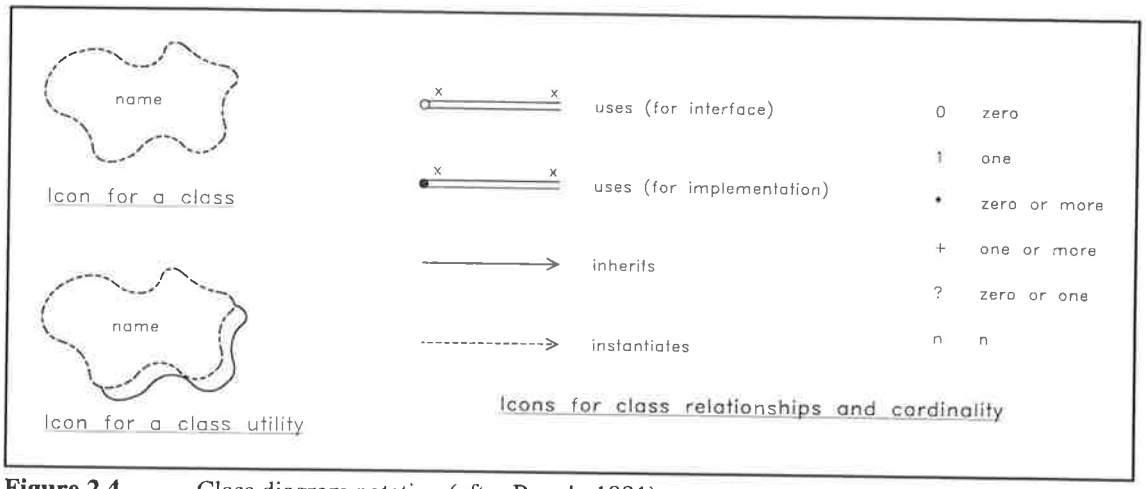


Figure 2.4. Class diagram notation (after Booch, 1991).

Booch to represent these types of relationship is also shown in figure 2.4, to which the following notes refer.

i. Inheritance relationships. A derived class inherits the attributes of its base class(es), as described in section 2.2. Such a relationship is denoted by a solid line originating at the derived class, and terminating in an arrowhead at the base class. This notation serves to describe both single and multiple inheritance relationships.

ii. Using relationships. A using relationship implies that one class (the *client*) uses the services provided by another. Booch distinguishes between two types of using relationship. In the one, the interface of a class uses another class, and the relationship is explicitly indicated to all client classes of the using class. Alternatively, the implementation of a class may enter into a using relationship with another class. Both types of relationship are indicated by a double solid line. Where the interface of a class uses the services of another, an open circle is placed at the end of the double line adjacent to the using class; a closed circle is used to indicate that it is the implementation of the using class that is involved in the relationship. A set of symbols to indicate the cardinality of a using relationship are also shown in figure 2.4. To understand how this notation works, consider the case where be have a class *A* which uses class *B*, and that we place the icon representing this relationship connecting the icons for the two classes, with the symbol `1` adjacent to class *A*, and the symbol + adjacent to class *B*. This implies that for each instance of class *A*, there can be one or more instances of class *B*, and that for each instance of class *B*, there can be one and only one instance of class *A*.

iii. Instantiation Relationships. Instantiation occurs when we derive a class of a specific type from a class of a generic type. In C++, this type of relationship is supported by templates. Thus, referring to section 2.2, if we derived a class to represent a singly-linked list of integers (by declaring an object of the appropriate type), we would say that this provides an instantiation of the appropriate template class. Relationships of this type are indicated by a broken line connecting the icons

representing the two classes, and terminating with an arrowhead adjacent to the template class.

We may also encounter the situation where a set of free subprogrammes use the services provided by one or more classes. Booch refers to such a set of subprogrammes as a set of *class utilities*, which are also represented by an icon shown in figure 2.4.

2.3.3.2. Programme Description Language.

A programme description language is essentially a means of imposing a degree of rigour on a textual or mathematical description of an algorithm by setting the description within a syntactical structure loosely based on the syntax of a structured programming language. The PDL may therefore be seen as an intermediate stage between the plain textual description, and a machine-readable programming language. The intention is that the resulting pseudocode will be easily understood by a reader with limited programming experience, while at the same time presenting the programmer with an algorithm description which may readily be translated into machine-readable code, while preserving its structure. Booch in fact suggests the possibility of using a language such as C++ directly as a programme description language, and one is reminded that one of the earliest structured programming languages, ALGOL (*Algo*rithmic *L*anguage; Dijkstra, 1960), was designed with the intention that it should be used not only for implementing software, but also as a means of describing and documenting algorithms. For present purposes however, a slightly less formal approach was felt to be in order. The ability to present an algorithm description using a mixture of textual and mathematical notations, as most appropriate, permits a considerable degree of conciseness to be achieved. It also makes the algorithm descriptions more readily accessible to the reader not familiar with the nuances of a specific programming language.

In the PDL used herein, the syntax has been largely borrowed from the C language (Kernighan and Ritchie, 1988). The semantics are based on a number of structured programming languages with which the author has worked over the years, but probably owe more to Pascal (Jensen and Wirth, 1978) than to any other.

An algorithm description is composed of a sequence of statements, each of which is terminated by a semicolon. Statements may be grouped into compound statements or blocks, delimited by parentheses, thus {}. A block may be prefixed by an iterative or conditional expression, statements within the block being commonly subject to the actions implied by that statement. Blocks may be nested. In general, a compound statement may be used wherever a simple statement would be permitted, and *vice versa*. Statements may be of two kinds, mathematical statements and imperative statements. The expressions within statements are italicized.

A mathematical statement is composed of a mathematical expression comprising a set of mathematical operators, and the entities on which they act. Each entity may be represented in a standard symbolic form, or by a textual description of the entity. The value to which an expression evaluates may optionally be assigned to an expression on the left-hand side of the statement. The following operators are used in mathematical statements:

Assignment : ⇐

Arithmetic Operators[13] : + - / ×

Precedence Operators : () []

Operators of the latter type may be nested.

An imperative statement is a textual statement which concisely defines an action or set of actions to be performed. Usually, this will correspond to a function or subroutine call.

A logical expression comprises one or more simple logical expressions, separated by the logical operators:

and or

A logical expression, and each of its component expressions must evaluate to *true* or *false*. At its simplest, a simple logical expression may simply comprise a boolean variable. It may also comprise a comparison between two entities using the comparison operators:

= ≠ < ≤ ≥ >

The PDL also supports the negation operator

not

The precedence operators may also be used within a logical expression.

The execution of statements within an algorithm is controlled by iterative and conditional expressions. The PDL supports the following iterative expressions:

for *sequence description* **do**
{
⋮
}

while *logical expression* **do**
{
⋮
}

In the first case, the compound statement is evaluated for each item in the specified sequence. In the second case the compound statement is evaluated repeatedly for so long

---

[13] In accordance with customary usage, the multiplication operator may be omitted where its presence is clearly implied by the syntax.

as the logical expression evaluates to true. A conditional expression, in its most general form, may be written thus:

if *logical expression*$_1$ **then** {
⋮
}
else if *logical expression*$_2$ **then** {
⋮
}
⋮
else {
⋮
}

A compound conditional expression, such as that shown above, will cause the compound statement following the *first* logical expression which evaluates to true to be executed. If none of the logical expressions evaluates to true, the compound statement following the optional final **else** statement will be evaluated. Once a true logical expression has been found, none of the succeeding logical expressions will be evaluated.

Comments are denoted by text in Helvetica font enclosed in brackets.

2.4. Reading System Specifications.

The Zebra Kernel extracts the specifications for a computational model for an air conditioning system from a text file, or compatible medium. This last remark implies that the medium can be accessed as if it were a text file using the C++ input/output streams mechanism (Stroustrup, 1990), which defines a hierarchy of classes providing progressively more specialized services for handling an input/output stream. Class *istream* is a base class providing services for extracting data from, and otherwise manipulating input streams. Classes derived from *istream* including *ifstream*, which adds facilities for input operations on file-based streams, including member functions to open and close files, and *istrstream*, which provides facilities for extracting data from null-terminated text strings in memory, using standard C++ input operators[14]. By writing the data extraction facilities in terms of the base class *istream*, the nature of the input source becomes transparent to the client functions. Ultimately of course, the input source must be specified somewhere, but by proceeding in this manner, such considerations can be isolated to the outermost levels of a programme which uses the services.

The system specifications are extracted from the file as a sequence of *lines* or *records*, each of which is prefixed by a five-digit code, which determines what information is to be extracted from the body of the record. In addition, the file may contain blank lines, and

---

[14] Class *ifstream* also inherits from class *fstream*, which provides basic services for file streams, regardless of the direction of data transfer. Similarly, *istrstream* inherits from class *strstream*. Ultimately, all stream classes are derived from a common base class *ios*.

comment lines. In the case of the latter, the first non-whitespace character on the line must be #. Both blank lines and comment lines are skipped on input.

Class *TextSource* is provided to facilitate the extraction of data from the text file (or other source). The constructor for this class takes the form

```
TextSource::TextSource (istream& isb,
                        char* fn = "");
```

The first argument in the above specifies the source from which the data are to be read. The second specifies a null-terminated character string which may be used to identify the data source (in error messages for instance).

The most important member function of class *TextSource* is the following:

```
istream* TextSource::InputLine ();
```

which reads the next line of data from the text source, skipping any blank lines or comment lines found in the process. If a valid line of data is found, it is stored, null-terminated, in a dynamically-created internal buffer, which will be released by a subsequent call to `InputLine`, or when the destructor is invoked. If a buffer is created, an object of class *istrstream* is attached to it, enabling client code to extract the data from the record using standard C++ input operators. A pointer to the *istrstream* object referencing the internal buffer is returned. If its value is null, it should be interpreted to mean that the input source is exhausted.

Additional member functions perform the following services:

i. `istream* TextSource::str ();`

   Returns a pointer to the *istrstream* object referencing the internal buffer. See also the remarks above.

ii. `int TextSource::code ();`

   Attempts to extract a five-digit numeric code from the beginning of the record in the internal buffer. If successful, the value of the code is returned; otherwise a value of -1 is returned.

iii. `int TextSource::operator! ();`

   Overloading the operator ! returns a value of 1 if the input source is exhausted, 0 otherwise.

iv. `int TextSource::line ();`

   Returns the number of the current line within the text source. Mainly used by the error reporting routines.

v.
```
void TextSource::Ierror (char* s);
void TextSource::Ierror (error_code c);
void TextSource::Ierror (error_code c,
                         long n);
void TextSource::Ierror (error_code c,
                         int n);
void TextSource::Ierror (error_code c,
                         unsigned n);
void TextSource::Ierror (char* s,
                         unsigned n);
void TextSource::Ierror (error_code c,
                         double r);
void TextSource::Ierror (error_code c,
                         char* s);
void TextSource::Ierror (error_code c,
                         char cd);
```

Overloaded error-handling routine, which enables error messages to be output in a range of formats. Invoking any of the above functions causes programme execution to terminate unconditionally.

Many of the classes used by the Zebra Kernel define constructors which take a reference to an object of class *TextSource* as an argument, or otherwise provide the constructor with a mechanism to read the specifications for the object being created from such a source. The various prefix codes control the flow of execution of the programme while the system model is being assembled. In particular, they indicate what constructors are to be invoked. Consider the following sequence of input records:

```
⋮
# Commence input for Zone 1.
16000 1
# Name of zone.
16001 NW Zone
# Comment.
16002 Perimeter zone
# 1 = Thermostat enabled; 24.0 = Thermostat setting (deg. C).
16003 1 24.0
# 13.189 = Sensible load (kW); 0.941 = RSHR.
16008 13.189 0.941
# Terminate input for zone.
16999
⋮
```

When a record prefixed with code `16000` is encountered, it indicates that a new object of type *Zone* is to be created, which will be done provided the code is valid at that point in the execution of the programme. The remainder of the record is read, and a new object class *Zone* created thus:

```
Zone* z = new Zone (uid, t);
```

where the arguments to the constructor specify a unique numeric index for the zone (`uid`; read from the body of the record), and a reference to the object of class *TextSource* from which the specifications are being read. At this point, control is passed to the constructor for class *Zone*, which processes the remainder of the records in the above sequence. The

main loop of this function implements the logic indicated by the following (incomplete) code fragment[15]:

```
for (boolean finish = false; !finish && t.InputLine ();)
{
  long code = t.code ();
  switch (code)
  {
    case -1    : ...  // Error - no code read from record
    case 16001 : ...
    case 16002 : ...
    case 16003 : ...
      :
    case 16008 : ...
      :
    case 16999 : finish = true; break;
    default    : ...  // Error - code read is invalid
  }
}
```

The loop is repeated until an error condition is encountered, the boolean flag `finish` is assigned a value of *true*, or the end of the input source is reached. At each repetition a record is read from the input source, and control passed to the appropriate labelled statement according to the value specified by the code. The statement labelled `default` serves to catch a code which has not been specifically enumerated among the preceding `case` statements; such a code will be invalid at this point in the programme execution. When code `16999` is encountered, data specification for the zone is complete. Flag `finish` is assigned a value of *true* causing programme execution to break out of the loop, ultimately returning execution to the routine from which the constructor was invoked.

The purpose of the constructor may not be satisfied if the records above are specified in random order. Some records will be mandatory; in the case of many others it makes no sense for the prefix code to be duplicated within the specifications for a particular object. Again, instances may arise where codes must be read in a specific order. Violations of any of the above conditions can be detected provided it is known which records have been read. Before the above loop is entered, an associative array is declared taking an integer variable as the key, and a boolean variable as the value, and an element is created for each valid code, and initialized to *false*:

---

[15] In the actual implementation the prefix codes are referred to by a descriptive set of pseudonyms, rather than their numeric values. Thus:

```
const int ZONE_BEGIN              = 16000;
const int ZONE_NAME               = 16001;
const int ZONE_COMMENT            = 16002;
const int ZONE_THERMOSTAT_SETTING = 16003;
```

and so forth. In later chapters the appropriate pseudonym will generally be used where it is desired to refer to a specific code. In the present discussion, it is however convenient to use the numeric values.

```
Map<long, boolean> rd (0, false);
rd[16001] = false;
rd[16002] = false;
  ⋮
rd[16999] = false;
```

Thus, as each code is encountered, it is possible to check whether any preconditions have been violated, before the element of the array corresponding to the code is assigned a value of *true*. Finally, after execution has broken out of the loop, the array may be checked to see that all mandatory records have been read.

It is of course implicit in the above that certain prefix codes will cause new objects of various types to be created, control being passed to the appropriate constructor to read the specifications for the new object, before being passed back to the constructor from which it was invoked. We will even have occasion to invoke constructors recursively. Discussion of such cases will be deferred until appropriate in the following.

Finally, it should be pointed out that while the input files required to support the above scheme may become quite complex, the structuring of the files lends itself to generation by a menu-driven input utility. This consideration was instrumental in the design of the scheme.

Chapter 3. Ventilation and Air Quality.

Comfort air conditioning is concerned with the process of maintaining the state of the air within an enclosed space in a condition which is at once conducive to the comfort and health of the occupants. The acceptability or otherwise of an indoor environment will be determined by a number of factors. The provision of air conditioning to an occupied space provides an opportunity to control two of the major components which determine the acceptability of an indoor environment, namely:

i. The thermal climate, as determined by the dry bulb temperature, the moisture content of the air in the space and the degree of air movement.

ii. The presence and concentration of atmospheric contaminants, which may include (ASHRAE, 1993):
   - Bioeffluents, produced by biological processes;
   - Volatile organic compounds;
   - Particulate matter;
   - Inorganic gaseous combustion products;
   - Radon progeny and other airborne radioactive substances;
   - Formaldehyde, pesticides and aerosols;
   - Particles of organic and inorganic solids and liquids;
   - Charged particles, such as hydrated ions, and
   - Condensation nuclei for viruses and bacteria,

   among others, including in particular carbon dioxide.

From the point of view of the overall thermal design of an air conditioning system, the engineer may seek to control the above components of the indoor environment by modulating the following three variables:

i. Dry bulb temperature;
ii. Relative humidity;
iii. Distribution of ventilation air.

Th functional relationships which determine the acceptability of the indoor environment have been subject to investigation by workers from a broad range of disciplines for many years now. The causal factors which have been investigated have included the above three, together with a wide range of others over which the engineer has no direct control. The purpose of the following discussion is not to review the field in its entirety[16], but rather to use a small sample of the literature to establish a basis upon which a set of target design values for the above variables can be selected. Once a set of target design criteria have been established for an application, the relative energy efficiency of various candidate design solutions can meaningfully be compared subject to the provision that they are constrained to satisfy the design criteria across the range of operation of the system.

---

[16] An extensive review of recent work on the bases of the health and comfort aspects of indoor air climates is provided in the ASHRAE Fundamentals Volume (1993; chapter 8).

3.1. Thermal Comfort.

Conditions of thermal comfort may be said to be satisfied if the majority of the occupants of the space desire an environment which is neither hotter nor colder than its current level. The physiological and environmental factors which influence the mechanisms whereby the body establishes equilibrium with the thermal environment have been extensively investigated (Fanger, 1972; Fisk, 1980; ASHRAE, 1993). On the basis of an extensive series of field trials and climate chamber tests, Fanger (1972) came to the conclusion that the major factors influencing the condition at which thermal comfort is reached are:

- Activity level (heat production in the body);
- Thermal resistance of the clothing (clo value);
- Air temperature;
- Mean radiant temperature;
- Relative air velocity;
- Water vapour pressure in ambient air.

Using his measurements as a basis, Fanger derived the comfort equation, which describes the locus of combinations of the above factors which will produce optimal thermal comfort. The comfort equation may be written (Fisk, 1980) in simplified form as

$$\theta_{rs} = 33{\cdot}5 - 3I_{cl} - (0{\cdot}08 + 0{\cdot}05I_{cl})M_s \quad , \tag{3.1.1}$$

in which

θ_{rs} is the resultant dry-bulb temperature,
I_{cl} is the insulation provided by the clothing of the occupants, and
M_s is the activity level of the occupants.

The resultant dry-bulb temperature,

$$\theta_{rs} = 0{\cdot}5\theta_a + 0{\cdot}5\theta_r \tag{3.1.2}$$

where θ_a is the temperature of the air in the space, and θ_r is the mean radiant temperature of the surfaces in the space, weighted according to area[17]. The simplified form of the equation is suitable for use where the air velocity is not sufficiently high to cause discomfort. It is also assumed in the simplified form of the equation that the influence of humidity on thermal comfort is of secondary importance in the vicinity of the preferred temperature. The important observation to be made from the above equation is that the optimal dry-bulb temperature for a conditioned space is not a unique value, but that it depends on the activity level of the occupants of the space (M_s), and upon the degree of

---

[17] In the discussion in the following chapters, it is implicitly assumed that $\theta_a = \theta_{rs}$. The algorithms developed in chapter 6 may readily be adapted to account for deviations from this assumption, provided a detailed thermal model of the zone is available.

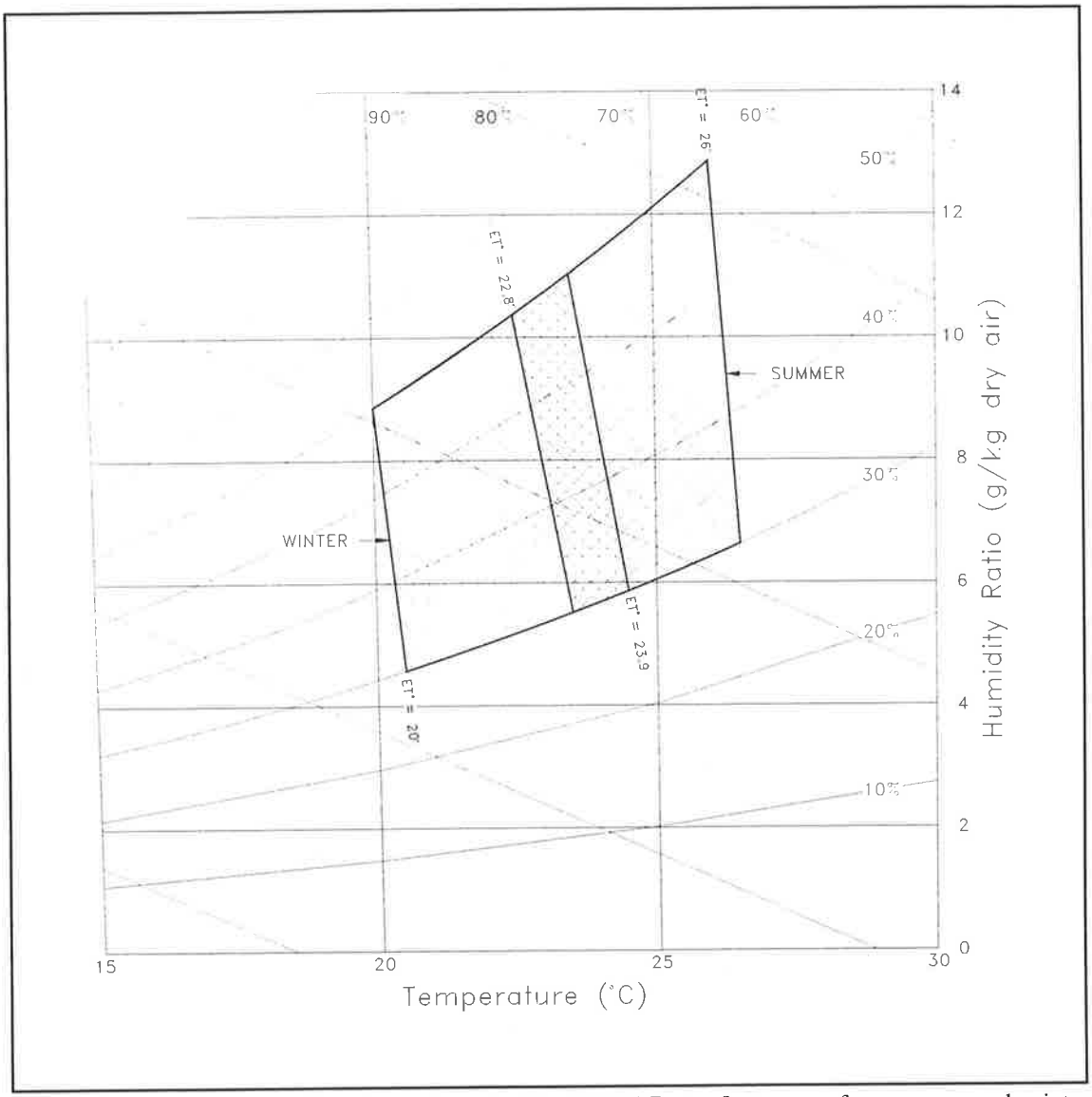


Figure 3.1. Psychrometric chart showing the ASHRAE comfort zones for summer and winter conditions.

insulation provided by their clothing[18] (I_{cl}). While the mean activity level in a functionally designated space may usually be assumed to remain constant throughout the year, the clothing worn by the occupants will vary, except perhaps in tropical climates. People tend to dress according to their *perception* of what they *should* wear. So the CEO who lives in an air conditioned house with a heated garage, who drives to work in an air conditioned car which will be parked in his/her designated space beside the lift, spends the day in an air conditioned office and reverses the process in the evening will probably wear an undershirt and thick socks and work shirt in winter, but no undershirt, thin socks and a lightweight shirt in the summer. Thus the optimal resultant dry-bulb temperature will vary according

---

18 The measure of activity level (M_s) is the metabolic rate for unit surface area for the subject, measured in units of W/m² in equation (3.1.1). The insulation value of clothing (I_{cl}) is expressed in units of clo. ASHRAE (1993) tabulates the rate of metabolic heat generation for various activities, and the insulation values for a number of typical clothing ensembles.

to equation (3.1.1), and to an extent will track the ambient dry-bulb temperature. This observation forms the basis for the comfort integration method of thermal control (Shaw and Luxton, 1990), application of which enables significant savings in energy consumption to be made.

The effective temperature ET^* (Gagge et al., 1971; Fobelets and Gagge, 1988) is an environmental index which seeks to provide a single measure of the human response to a given set of psychrometric conditions. The effective temperature describing a given parcel of air is defined as the dry-bulb temperature of a "standard" parcel of air at the same dry bulb temperature, but at 50% relative humidity which would give rise to the same rate of heat transfer to or from a human body. In general, this is a function not only of the psychrometric condition of the test environment, but also of the activity level and the permeability index (i_m) of the clothing worn by the subject[19]

The comfort zone as defined by ASHRAE Standard 55-1989 (ASHRAE, 1989a) is delineated on a psychrometric chart by lines of constant effective temperature, and by lines of constant relative humidity (figure 3.1). In accordance with the principles enunciated earlier, separate but overlapping comfort zones are defined for summer and winter conditions. The winter comfort zone lies between 20 and 23·9°C ET^*, the summer zone between 22·8 and 26°C ET^*. In each case these are to be interpreted as standard effective temperatures, evaluated under a set of standard conditions representative of typical indoor applications. The defining conditions are I_{cl} = 0·6 clo; i_m = 0·4; M_s = 1·0 met[20]; $\theta_r = \theta_a$ and air velocity below 0·1 m/s. It will be observed from figure 3.1 that, within the range of temperatures covered by the comfort zone, humidity is indeed a secondary factor in determining human comfort, thus justifying the use of the simplified form of the comfort equation (3.1.1) for preliminary computations at least. For a broad range centred upon the comfort zone, the effective temperature in itself is an index of comfort rather than of health. Within this range, a departure from the preferred temperature has no direct effect on the health of a subject. As the temperature increases significantly beyond a comfortable level, the humidity of the air becomes increasingly important in determining the rate of heat loss from a human body, and the effective temperature becomes increasingly important as a determinant of human health. Similarly, with falling temperature, a point is reached at which health and body functions may be adversely affected. The effective temperature can be correlated with the response of the human body to the thermal environment, in which case it is known as the comfort-health index (CHI; Gagge et al., 1971; ASHRAE, 1993). The extremes of the range are however of little interest in comfort air conditioning applications.

As we have seen, relative humidity is not a major factor in determining the comfort of the occupants of a space, provided the dry-bulb temperature is close to the preferred level. The relative humidity which can be tolerated has been stated by one authority (Fisk, 1980) to span the range from 25 to 75%. While this range may be argued to be acceptable from a

---

19 ET^* is related to the defining variables by a coupled set of equations. A computer programme to solve these is listed in Gagge et al. (1971).

20 1 met = 58·2 W/m².

comfort point of view, health considerations dictate that narrower limits be imposed upon the acceptable relative humidity range. High levels of relative humidity are associated with the growth of bacterial, viral and fungal populations (Brown and Mathieson, 1991; ASHRAE, 1993), and with the promotion of certain allergens, including in particular the faeces of the house-dust mite (*Dermatophagoides pteronyssinus*; Spieksma, 1991, 1993). Relative humidity levels of greater than 60% will support populations of the house-dust mite, the range of 70-80% being optimal for the growth of populations (Spieksma, 1970, 1990; Spieksma et al., 1971). Excessively low levels of relative humidity may lead to dehydration and cause injury to the skin, eyes, nose, throat and mucous membranes (ASHRAE, 1993). The resulting dry tissues are less resistant to infection, and low relative humidity levels have been linked to infections of the upper respiratory tract in particular (Gelperin, 1973; Green, 1979, 1982). Relative humidity levels also have a significant effect on dust emission from carpets and furnishings, and on static electricity (Brown and Mathieson, 1991). ASHRAE Standard 55-1989 specifies a comfort zone delimited by relative humidity levels of 30 and 60% (figure 3.1). Two air conditioning practitioners (Brown and Mathieson, 1991) recommend a much narrower range of 45-55% as being the optimal range to satisfy health and comfort requirements. This latter range would seem to be an appropriate design target for quality installations. ASHRAE Standard 55-1989 further stipulates that the relative humidity of air in low velocity ducts should not exceed 70% to avoid fungal growth.

3.2. Ventilation.

The purpose of ventilation is to prevent the concentration of airborne contaminants building up to a point at which they either cause discomfort or constitute a health hazard to the occupants of the space. ASHRAE Standard 62-1989 is titled "Ventilation of Acceptable Indoor Air Quality", and seeks to specify the minimum ventilation rates for a conditioned space which will prove both acceptable to the occupants and avoid adverse health effects. The ASHRAE standard specifies a set of minimum permissible ventilation rates, expressed as volume flow of fresh air per unit time per occupant, and tabulated according to the use to which the space is put. The standard is avowedly "comfort-based" (Cox and Miró, 1990), the minimum ventilation rates being selected on the basis that they will be adequate to reduce the level of odours such that at least 80% of the occupants will be satisfied, while reducing the concentration of contaminants to a level which is not harmful. The standard is prefaced with the disclaimer that "With respect to tobacco smoke and other contaminants, the standard does not, and cannot, ensure the avoidance of all possible adverse health effects, but it reflects recognized consensus criteria and guidance".

For office areas with "a moderate level of smoking", a minimum ventilation rate of 10 L/s per person is stipulated. The 1981 edition of the standard stipulated a minimum ventilation rate of 2·5 L/s per person for a smokefree office environment, and 10 L/s per person if smoking is permitted. Recent British recommendations (quoted by Langley and Whitbread, 1990; see also Procell, 1990) suggest that 8-8·5 L/s per person is appropriate for a non-smoking environment, increasing to 16 L/s per person with light smoking, and 32 L/s per person with heavy smoking. The latter figure is based on work which indicates that if 80% of visitors to a space are to be satisfied with the air quality, 4 to 5 times as

much fresh air must be supplied to the space as would be required if it was a smokefree environment. Current Australian practice, as reflected in the 1991 revision of AS 1668.2, and in the rates specified by the Commonwealth Government and private developers (Brown and Mathieson, 1991) suggests that a figure of 10 L/s per person is representative. This figure may be taken to apply to buildings which will be almost, if not entirely smokefree. The Government of Singapore on the other hand imposes an *upper* limit of 2·5 L/s per person[21] on the ventilation rates for new buildings, in the interests of energy conservation.

Recently, Fanger has challenged the basis upon which the minimum ventilation rates specified by most standards are calculated, it being implicitly assumed that the principle source of indoor air pollution is the occupants themselves. In order to quantify field measurements of air pollution sources and air pollution, Fanger (1988b) introduced two new units, the olf and the decipol, defined as:

"One olf is the emission rate of air pollutants (bioeffluents) from a standard person".

"One decipol is the pollution caused by one standard person (one olf) ventilated by 10 L/s of unpolluted air".

Fanger (1988a; Fanger et al., 1988) cites the results of a study in which a panel of judges assessed the air quality in a randomly selected sample of 15 offices and 5 assembly halls in Copenhagen. The average office in the study occupied a floor area of 230 m², and had 17 occupants. However, while the occupants contributed 17 olfs to the perceived air pollution in the space, some 28 olfs were contributed by materials in the space, 58 olfs by the ventilation system, and 35 olfs by smoking. The average ventilation rate within the spaces tested was 25 L/s per person, which is far in excess of the minimum value recommended by standards cited earlier. However, when the extra pollution load is taken into account, this translates to only 4 L/s per olf (or 4 L/s per person if no other pollutants are present), with correspondingly lower values pertaining in spaces ventilated at the minimum rate permitted by the standards cited above.

Fanger's recommendations, which form the basis for a draft European Standard, can be summarized as follows:

i. Furnishings, building materials, paper, office machines and other potential pollution sources within the conditioned space should be selected on the basis of minimal olf value.

ii. Potential pollution sources within ventilation systems should be identified, and ventilation systems designed both to minimize their contribution to the total pollution in the spaces served when new, and to facilitate maintenance so that they can be kept clean throughout the lifecycle of the system.

---

[21] This can be increased to 3·5 L/s per person if a special case can be made to the controlling authority (the Ministry of National Development).

iii. Greater attention should be given to cleaning buildings for the purpose of reducing olf values, rather than cleaning primarily for aesthetic reasons.

iv. Ventilation rates should be specified at a level sufficient to handle the total pollution load, and future standards should specify minimum ventilation rates in terms of olfs per unit floor area.

The concentrations of carbon dioxide, carbon monoxide, particulates and total volatile organic compounds within the spaces were measured, but were found to correlate poorly with the levels of dissatisfaction experienced with the air quality in the various spaces. Fanger (1988b) makes the point that the decipol load is a measure of perceived odour levels, rather than of health risk. However, he does speculate (1988a) that the hidden pollution sources uncovered in this study are a likely cause of the physical symptoms associated with sick building syndrome.

In terms of design economics, there is a need to balance the cost of designing buildings and building services and of selecting furnishings to minimize pollution potential, against the additional cost necessarily incurred in providing sufficient ventilation to offset a higher pollution load. Candidate air conditioning design solutions need to be compared in terms of their energy efficiency over a range of ventilation loads. A preliminary study of this aspect of air conditioning system performance has been conducted (Marshallsay et al., 1993). This study in revised form is considered in Chapter 13.

Chapter 4. Properties of Moist Air.

The processes occurring within the air side of an air conditioning cycle are complicated by the fact that we are dealing with a two-phase fluid. Proper control of the indoor thermal environment requires not only that we maintain the dry-bulb temperature at a comfortable level, but also that we constrain the room humidity to fall within certain specified limits. The science of psychrometrics, which describes the thermodynamic properties of moist air, is thus fundamental to an understanding of the air conditioning process. The purpose of this chapter is to establish a computational framework for the calculation of moist air properties at various points in the air conditioning cycle. As a preliminary step, a set of relationships connecting the various thermodynamic properties of moist air are developed. In certain stages of the air conditioning cycle, including in particular flow through the cooling coil, transport properties such as viscosity and thermal conductivity are also of importance; expressions to permit the calculation of these properties for a mixture are presented. These form the basis for a C++ class *AirState*, which is used to evaluate the properties of moist air either at a point, or following a psychrometric process.

4.1. Psychrometric Properties.

The thermodynamic properties of moist air may be obtained by reference to tables of properties, or to the psychrometric chart. The current ASHRAE tables (ASHRAE, 1985), and the corresponding psychrometric charts (Stewart et al., 1983) are based on a rigorous set of formulations for the thermodynamic properties of dry and saturated moist air, and of saturated water vapour, developed by Hyland and Wexler (1983a,b). Within the current computational scheme, it is convenient and sufficiently accurate to use a set of relationships based on the assumption that the components of moist air are perfect gases. The relationships in the following are, unless otherwise stated, taken from ASHRAE (1985) and Stewart et al. (1983), to which the reader is referred for a more detailed discussion of the derivations and the underlying assumptions involved.

4.1.1. Fundamental Humidity Parameters.

The moisture content of air is most commonly expressed in terms of the humidity ratio, which is defined as the ratio of the mass of water vapour to the mass of dry air:

$$W = \frac{m_w}{m_a} \qquad \textbf{(4.1.1)}$$

The mole fraction is a quantity which finds widespread use in computations involving the properties of mixtures of gases. By definition, the mole fraction for a particular component is the ratio of the number of moles of that component to the total number of moles in a sample of the mixture. Let the mole fraction for dry air be x_a, and that for water vapour be x_w. It follows that for a mixture of air and water vapour only

$$x_a + x_w = 1 \quad . \tag{4.1.2}$$

To obtain the mole fraction of either component, given the humidity ratio, equation (4.1.2) may be used in conjunction with the following expression:

$$W = \frac{M_w}{M_a}\frac{x_w}{x_a} \tag{4.1.3}$$

in which M_a and M_w are the molecular weight of air and water vapour respectively. From Hyland and Wexler (1983b), the molecular weight of dry air is 28·9645 kg/kmol, while that of water vapour is 18·01528 kg/kmol. This large difference is of great significance when studying climatic conditions.

4.1.2. Humidity Parameters Referenced to the Saturation Line.

The partial pressure of water vapour is a function of dry-bulb temperature, and may be calculated using expressions developed by Hyland and Wexler (1983). The saturation pressure *over ice* within the temperature range of -100°C to 0°C is given by

$$\ln p_{ws} = \sum_{i=0}^{5} m_i T^{i-1} + m_6 \ln T \qquad (Pa) \tag{4.1.4}$$

where p_{ws} is the saturation vapour pressure, T is the absolute temperature (K), and

$m_0 = -0{\cdot}56745359 \times 10^4$,
$m_1 = 0{\cdot}63925247 \times 10$,
$m_2 = -0{\cdot}96778430 \times 10^{-2}$,
$m_3 = 0{\cdot}62215701 \times 10^{-6}$,
$m_4 = 0{\cdot}20747825 \times 10^{-8}$,
$m_5 = -0{\cdot}94840240 \times 10^{-12}$,
$m_6 = 0{\cdot}41635019 \times 10$.

For water vapour *over liquid water* within the temperature range from 0°C to 200°C, the saturation pressure is given by

$$\ln p_{ws} = \sum_{i=-1}^{3} g_i T^i + g_4 \ln T \qquad (Pa) \tag{4.1.5}$$

where,

$g_{-1} = -0{\cdot}58002206 \times 10^4$,
$g_0 = 0{\cdot}13914993 \times 10$,
$g_1 = -0{\cdot}48640239 \times 10^{-1}$,
$g_2 = 0{\cdot}41764768 \times 10^{-4}$,
$g_3 = -0{\cdot}14452093 \times 10^{-7}$,
$g_4 = 0{\cdot}65459673 \times 10$.

If p_a is the partial pressure of air, and p_w is the partial pressure of water vapour, then the total pressure, $p = p_a + p_w$. From the ideal gas relationship, the following expressions for the mole fraction of air and water vapour respectively may be derived:

$$x_a = \frac{p_a}{p_a + p_w} = \frac{p_a}{p} \quad , \tag{4.1.6}$$

$$x_w = \frac{p_w}{p_a + p_w} = \frac{p_w}{p} \quad . \tag{4.1.7}$$

From equation (4.1.3), it follows that the humidity ratio is related to the partial pressure of water vapour by the expression

$$W = 0{\cdot}62198 \frac{p_w}{p - p_w} \quad . \tag{4.1.8}$$

Hence, the saturation humidity ratio

$$W_s = 0{\cdot}62198 \frac{p_{ws}}{p - p_{ws}} \quad . \tag{4.1.9}$$

Use of equation (4.1.9) to evaluate the saturation humidity ratio is subject to a number of inaccuracies arising from

i. The effect of dissolved gases on the properties of the condensed phase;
ii. The effect of pressure on the properties of the condensed phase;
iii. The effect of intermolecular forces on the properties of the moisture itself.

These effects can be compensated for by applying an enhancement factor, f_s:

$$W_s = 0{\cdot}62198 \frac{f_s p_{ws}}{p - f_s p_{ws}} \quad . \tag{4.1.10}$$

The enhancement factor is tabulated as a function of dry-bulb temperature and pressure in ASHRAE (1985). For computational purposes, it is more convenient to use the following set of equations derived from the tabulated data (Van Aken, 1993):

$$f_s = c_0 + c_1 t + c_2 t^2 \tag{4.1.11}$$

where

$$c_0 = 1{\cdot}0004 + 3{\cdot}995 \times 10^{-5} P$$

$$c_1 = \left(\frac{653{\cdot}71}{P} - 6{\cdot}1322 \right) \times 10^{-6} \qquad (for\ P < 72 kPa)$$

$$c_1 = (26{\cdot}537 - 0{\cdot}32692 P) \times 10^{-6} \qquad (for\ P \geq 72 kPa)$$

$$c_2 = \frac{10^{-7}}{0{\cdot}402017 - 2{\cdot}583545 \times 10^{-3} P + 4{\cdot}128112 \times 10^{-6} P^2} \quad .$$

The degree of saturation is the ratio of the humidity ratio of the air sample to the humidity ratio of saturated air at the same temperature and pressure

$$\mu = \left.\frac{W}{W_s}\right|_{t,p} \quad . \tag{4.1.12}$$

The relative humidity is defined as the ratio of the mole fraction of water vapour in the air sample to the mole fraction of water vapour in a saturated sample at the same dry-bulb temperature and pressure:

$$\phi = \left.\frac{x_w}{x_{ws}}\right|_{t,p} = \left.\frac{p_w}{p_{ws}}\right|_{t,p} \quad . \tag{4.1.13}$$

This is related to the degree of saturation by the expression:

$$\phi = \frac{\mu}{1 - (1 - \mu)\left(\frac{p_{ws}}{p}\right)} \quad . \tag{4.1.14}$$

The dew point temperature, t_d is defined as the temperature at which a sample of moist air, if cooled at constant pressure and humidity ratio, will become saturated. It is found by solving the equation

$$W_s(p, t_d) - W = 0 \quad . \tag{4.1.15}$$

Equation (4.1.15) must be solved iteratively. Alternatively, dew-point temperature may be found directly as a function of dry-bulb temperature and water vapour pressure using relationships due to Peppers (unpublished paper cited in ASHRAE, 1993):

For temperatures in the range of -50°C to 0°C:

$$t_d = 6{\cdot}09 + 12{\cdot}608\,\alpha + 0{\cdot}4959\,\alpha^2 \tag{4.1.16}$$

and in the range of 0°C to 93°C:

$$t_d = \sum_{i=0}^{3} b_i \alpha^i + b_4 p_w^{0{\cdot}1984} \tag{4.1.17}$$

where,

$\alpha \equiv \ln(p_w)$,
$b_0 = 6{\cdot}54$,
$b_1 = 14{\cdot}526$,
$b_2 = 0{\cdot}7389$,
$b_3 = 0{\cdot}09486$,
$b_4 = 0{\cdot}4569$.

Reasonable agreement exists between results calculated using the Peppers equation and those based on the more rigorous equation (4.1.15). In the present work Peppers equation is used to provide a first approximation to dew-point temperature, which is then refined

using equation (4.1.15). In this manner exact compatibility is established between the dew-point temperature, and the saturation humidity ratio, as calculated using equation (4.1.10), and the iterative solution converges rapidly.

4.1.3. Specific Volume.

In keeping with the definition of the humidity ratio, the specific volume of dry air, v is defined as the volume occupied by unit mass of *dry* air. Thus, for a sample of moist air,

$$v = \frac{V}{m_a} = \frac{V}{M_a n_a} \qquad \textbf{(4.1.18)}$$

where n_a is the number of moles of dry air in the sample. Now, invoking the equation of state for an ideal gas,

$$p_a V = n_a R T \qquad \textbf{(4.1.19)}$$

and bearing in mind that $p = p_a + p_w$, gives

$$v = \frac{R_a T}{p - p_w} \qquad \textbf{(4.1.20)}$$

where $R_a = R/M_a = 287{\cdot}055$ kJ/kg.K is the gas constant for dry air. Substituting for p_w from equation (4.1.8) and rearranging gives the following expression for v:

$$v = \frac{R_a T}{p}(1 + 1{\cdot}6078\, W) \quad . \qquad \textbf{(4.1.21)}$$

The density of moist air, conventionally defined as the mass of unit volume of *moist* air, is related to the specific volume by the expression

$$\rho = \frac{1 + W}{v} \quad . \qquad \textbf{(4.1.22)}$$

4.1.4. Enthalpy and Specific Heat.

The enthalpy for a mixture of ideal gases is equal to the sum of the partial enthalpies of the individual components. Thus for moist air, the specific enthalpy

$$h = h_a + W h_g \qquad \textbf{(4.1.23)}$$

where h_a is the specific enthalpy of dry air, and h_g is the specific enthalpy of saturated water at the temperature of the mixture. Following Van Aken (1993), approximate expressions may be derived for h_a and h_g:

$$\begin{aligned} h_a &\approx 1{\cdot}006\, t \\ h_g &\approx 2501{\cdot}8 + 1{\cdot}8144\, t \end{aligned} \qquad (kJ/kg) \; . \qquad \textbf{(4.1.24)}$$

These expressions are sufficiently accurate for use over the range dry-bulb temperatures commonly encountered in air conditioning practice[22]. Substituting them into equation (4.1.23) gives the following relationship between dry-bulb temperature, humidity ratio and enthalpy:

$$h = 1{\cdot}006t + W(2501{\cdot}8 + 1{\cdot}8144t) \qquad (kJ/kg) \ . \tag{4.1.25}$$

Specific heat, like enthalpy, may be regarded as a function of temperature alone at low pressure. Consequently, the specific heat for moist air may be found using a relationship of the form

$$C_p = C_{pa} + WC_{pw} \tag{4.1.26}$$

where C_{pa} and C_{pw} are the specific heat of dry air and water vapour respectively, both evaluated at the temperature of the mixture. These may be evaluated using the following general form (ASHRAE, 1973):

$$C_p = A + BT + CT^2 + DT^3 \qquad (kJ/kg.K) \ . \tag{4.1.27}$$

Coefficients to calculate the specific heat of dry air and water vapour are given in table 4.1.

Gas	Range, K	A	B	C	D
Water	290-380	-0·23980	$2{\cdot}29880\times10^{-2}$	$-8{\cdot}56702\times10^{-5}$	$1{\cdot}08197\times10^{-7}$
	373-535	8·13705	$-3{\cdot}73435\times10^{-2}$	$7{\cdot}48227\times10^{-5}$	$-4{\cdot}95562\times10^{-8}$
	535-1500	1·85444	$-1{\cdot}19408\times10^{-4}$	$8{\cdot}30428\times10^{-7}$	$-2{\cdot}77702\times10^{-10}$
Air	260-610	1·04466	$-3{\cdot}15967\times10^{-4}$	$7{\cdot}07909\times10^{-7}$	$-2{\cdot}70340\times10^{-10}$
	610-900	1·00205	$-1{\cdot}62983\times10^{-4}$	$5{\cdot}69525\times10^{-7}$	$-2{\cdot}68081\times10^{-10}$
	600-1500	$8{\cdot}73749\times10^{-1}$	$3{\cdot}22598\times10^{-4}$	$-3{\cdot}58454\times10^{-8}$	$-1{\cdot}99063\times10^{-11}$

Table 4.1. Constants for the Specific Heat Equation (eq. 4.1.27).

4.1.5. Thermodynamic Wet-bulb Temperature.

For any given state of moist air, there exists a temperature t^*, at which water evaporates into the air to bring it to saturation adiabatically at the same temperature and pressure. This temperature is known as the adiabatic saturation temperature, or thermodynamic wet-bulb temperature. If h is the enthalpy of moist air at the beginning of an adiabatic saturation

---

22 The coefficients in these expressions differ from the corresponding expressions published by ASHRAE (1985), but provide a more accurate representation.

process, and W is its humidity ratio, and assuming the process to be strictly adiabatic and steady-state, the energy equation for the process[23] may be written

$$h + (W_s^* - W)h_w^* = h_s^* \qquad \textbf{(4.1.28)}$$

in which W_s^* and h_s^* are the humidity ratio and enthalpy respectively of moist air saturated at temperature t^*, and h_w^* is the enthalpy of liquid water at the same temperature. Approximately,

$$h_w^* = 4{\cdot}186t^* \qquad (kJ/kg) \qquad \textbf{(4.1.29)}$$

Substituting equation (4.1.29) for h_w^*, and equation (4.1.25) for h and h_s^* into equation (4.1.28), and rearranging gives

$$W = \frac{(2501{\cdot}8 - 2{\cdot}3724t^*)W_s^* - 1{\cdot}006(t - t^*)}{2501{\cdot}8 + 1{\cdot}8144t - 4{\cdot}1868t^*} \qquad \textbf{(4.1.30)}$$

where t and t^* are expressed in units of °C. This expression can be solved numerically for wet-bulb temperature, provided an initial estimate for the wet-bulb temperature is available. Van Aken (1993) presents the following approximate expression[24] which permits wet-bulb temperature to be evaluated approximately for known dry-bulb temperature and relative humidity:

$$t^* \approx 0{\cdot}6186t + 0{\cdot}004\phi t + 0{\cdot}05243\phi - 5{\cdot}43707 \qquad (°C) \qquad \textbf{(4.1.31)}$$

4.1.6. Latent Heat of Vapourization of Water.

The need to calculate the latent heat of vapourization of water arises in connection with the estimation of latent loads. Van Aken (1993) has fitted a set of polynomial correlations to latent heat figures published in steam tables. These correlations take the form

$$h_{fg} = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 \qquad \textbf{(4.1.32)}$$

where $x = t/100$. A suitable set of coefficients are presented in table 4.2.

4.2. Transport Properties.

4.2.1. Dynamic Viscosity.

Bird et al. (1960) have shown that the dynamic viscosity for a mixture of n gas species may be calculated to a reasonable degree of accuracy using the following semiempirical formula:

---

[23] For a detailed discussion of the physics underlying the adiabatic saturation process, and practical measurement of the wet-bulb temperature, the reader is referred to Threlkeld (1970).

[24] Adapted from Worbs (1984).

Range, °C	a_0	a_1	a_2	a_3	a_4
0·01-50	2501·47	-234·66985	-4·6764297	0·0	0·0
50-150	2502·32	-239·74293	9·3564359	-14·970007	0·0
150-236	2333·9	0·0	-98·755	0·0	0·0
236-350	-9776·98	17890·709	-10037·168	2466·9129	-231·61670
350-370	-25953943	29126978	-12257818	2292889·7	-160859·36
370-374·1	370044720	-299095820	80584514	-7237315	0·0

Table 4.2. Constants for the latent heat of vapourization of water equation (eq. 4.1.32).

$$\mu_{mix} = \sum_{i=1}^{n} \frac{x_i \mu_i}{\sum_{j=1}^{n} x_j \Phi_{ij}} \quad , \tag{4.2.1}$$

where

$$\Phi_{ij} = \frac{1}{\sqrt{8}} \left(1 + \frac{M_i}{M_j} \right)^{-\frac{1}{2}} \left[1 + \left(\frac{\mu_i}{\mu_j} \right)^{\frac{1}{2}} \left(\frac{M_j}{M_i} \right)^{\frac{1}{4}} \right]^2 \quad . \tag{4.2.2}$$

In the above, x_i and x_j are the mole fractions of pure species i and j, μ_i and μ_j are dynamic viscosities of the species at the appropriate temperature and pressure, and M_i and M_j are the respective molecular weights. In applying the above formula to moist air, little accuracy is lost if it is assumed that air is a pseudo-pure substance. The problem thus reduces to one involving two gas phases only, air and water vapour.

Gas	Range, K	A	B	C	D
Water	**280-500**	0·0500699	365·423	16018·0	0·0
	500-750	0·368683	490·099	-13608·0	0·0
	750-1000	0·309818	575·159	-44383·0	0·0
Air	**85-1000**	0·671692	85·22974	-2111·475	106417·0

Table 4.3. Constants for the Coefficient of Viscosity Equation (eq. 4.2.1).

At atmospheric pressure, and within the range of temperatures encountered in air conditioning practice, viscosity is essentially independent of pressure (Bird et al., 1960). Empirical formulae for the coefficient of viscosity as a function of temperature are given by ASHRAE (1973):

$$\mu = \frac{\sqrt{T}}{\left(A + \frac{B}{T} + \frac{C}{T^2} + \frac{D}{T^3}\right)} \qquad (pa.s \times 10^6) \tag{4.2.3}$$

in which T is expressed in degrees K, and A, B, C and D are empirical constants, appropriate values for which are listed in table 4.3.

4.2.2. Thermal Conductivity.

By analogy with equation (4.2.1) above, the thermal conductivity of a mixture of gas species may be calculated using the expression:

$$k_{mix} = \sum_{i=1}^{n} \frac{x_i k_i}{\sum_{j=1}^{n} x_j \Phi_{ij}} \tag{4.2.4}$$

in which k_i are the thermal conductivities of the pure species, the remaining symbols being as defined above.

The thermal conductivity of dry air and water vapour may be estimated using an expression similar to that given above for viscosity:

$$k = \frac{\sqrt{T}}{\left(A + \frac{B}{T} + \frac{C}{T^2} + \frac{D}{T^3}\right)} \qquad (W/m.K) \tag{4.2.5}$$

in which k is expressed in units of W/m.K, and appropriate values for the empirical constants A, B, C and D are as given in table 4.4.

Gas	Range, K	A	B	C	D
Water	**373-600** **600-800** **800-1000**	-138·818 -3·40306 98·3080	$4{\cdot}80327\times10^{5}$ $3{\cdot}30796\times10^{5}$ $1{\cdot}72643\times10^{5}$	$-4{\cdot}88631\times10^{7}$ $-7{\cdot}28429\times10^{6}$ $5{\cdot}43128\times10^{7}$	0·0 0·0 0·0
Air	**80-300** **300-600** **600-1000**	385·859 328·052 539·544	$9{\cdot}11440\times10^{4}$ $1{\cdot}67320\times10^{5}$ $-3{\cdot}32903\times10^{5}$	$-2{\cdot}68667\times10^{6}$ $-3{\cdot}02953\times10^{7}$ $3{\cdot}59756\times10^{8}$	$5{\cdot}52604\times10^{7}$ $3{\cdot}05682\times10^{9}$ $-9{\cdot}67202\times10^{10}$

Table 4.4. Coefficients for the thermal conductivity equation (eq. 4.2.5).

4.2.3. Mass Diffusivity.

Van Aken (1990) has considered the problem of mass diffusion involving a binary mixture of gases, and hence derived a procedure for calculating the mass diffusivity of water in air, based on the expression

$$D_{AB} = 4{\cdot}020493\times10^{-9}\frac{T^{3/2}}{p\Omega^{(1,1)*}}f_{\eta}^{(2)} \qquad (m^2/s) \tag{4.2.6}$$

where $\Omega^{(1,1)*}$, the reduced collision integral and the function $f_{\eta}^{(2)}$ are to be evaluated at the reduced temperature

$$T^* = T/49 \quad . \tag{4.2.7}$$

$f_{\eta}^{(2)}$ is tabulated as a function of T^* in table 4.5; intermediate values may be obtained by linear interpolation with sufficient accuracy for present purposes. Van Aken (1990) presents the following set of relationships for $\Omega^{(1,1)*}$:

For $T^* < 1{\cdot}0$:

$$\Omega^{(1,1)*} = \frac{1}{\sum_{i=0}^{2} a_i (T^*)^i} \tag{4.2.8a}$$

where

$a_0 = 0{\cdot}18802897;$
$a_1 = 0{\cdot}67627103;$
$a_2 = -0{\cdot}16955131.$

For $1{\cdot}0 \le T^* \le 10$:

$$\Omega^{(1,1)*} = \sum_{i=0}^{6} \frac{a_i}{(T^*)^i} \tag{4.2.8b}$$

where

$a_0 = 0{\cdot}59727007;$
$a_1 = 1{\cdot}8466139;$
$a_2 = -4{\cdot}9068264;$
$a_3 = 11{\cdot}610331;$
$a_4 = -15{\cdot}155818;$
$a_5 = 10{\cdot}260397;$
$a_6 = -2{\cdot}8131158.$

For $T^* > 10$:

$$\Omega^{(1,1)*} = \frac{1{\cdot}0602}{T_r^{0{\cdot}156}} \quad . \tag{4.2.8c}$$

$T^* = kT/\epsilon$	$f_\eta^{(2)}$
0·30	1·0001
0·50	1·0000
0·75	1·0000
1·00	1·0000
1·25	1·0002
1·5	1·0006
2·0	1·0016
2·5	1·0026
3·0	1·0037
4·0	1·0050
5·0	1·0059
10·0	1·0076
50·0	1·0080
100·0	1·0080
400·0	1·0080

Table 4.5. Functions for calculating the higher approximations to the coefficient of diffusion.

4.2.4. Dimensionless Groups.

The relationships connecting the various transport properties for a fluid may be expressed in terms of a number of dimensionless groups. Of particular importance are the Prandtl number,

$$Pr \equiv \frac{C_p \mu}{k} \tag{4.2.9}$$

and the Schmidt number,

$$Sc \equiv \frac{\mu}{\rho D_{AB}} \tag{4.2.10}$$

The Lewis number derives from the work of W.K. Lewis who showed (Webb, 1990) that $h_c = K_y C_p$ for the case of $Pr = Sc$, where h_c is a convection heat transfer coefficient. The analogy between heat and mass transfer is not however perfect. For gaseous mixtures in general, Threlkeld (1970)[25] has shown that

---

[25] Webb (1990) erroneously asserts that Threlkeld defines $h_c / K_y C_p$ as the Lewis number, where K_y is the mass transfer coefficient based on mass fraction y (of water in air in this instance) as the driving potential. In fact, Threlkeld's definition is based on K_W, which is the mass transfer coefficient based on specific humidity as the driving potential. As shown by Webb, conversion between the two coefficients may be effected by the expression

$$Le \equiv \frac{h_c}{K_W C_p} = \left[\frac{Sc}{Pr}\right]^{1-c} = \left[\frac{\alpha}{D_{AB}}\right]^{1-c} \qquad \textbf{(4.2.11)}$$

where $\alpha \equiv k/\rho C_p$ is the *thermal diffusivity*. Based on a review of available experimental data, Kusuda (1965) recommends that a value of $c = ⅓$ be used for forced convection, and $c = 0{\cdot}52$ be used for natural convection.

4.3. A Class for the Computation of Moist Air Properties.

The C++ class *AirState* provides a framework for the computation of moist air properties. An object of class *AirState* might typically refer to the properties of a sample of air fixed at a point in space, or of a sample of air undergoing a thermodynamic process. The fluid properties which are of interest, and which may be accessed for a properly specified air state are the following[26]:

1. Pressure.
2. Dry-bulb temperature.
3*. Humidity ratio.
4*. Mole fraction of air.
5*. Mole fraction of water.
6. Saturation vapour pressure.
7*. Vapour pressure of water.
8. Saturation humidity ratio.
9*. Degree of saturation.
10*. Relative humidity.
11*. Wet bulb temperature.
12*. Specific volume.
13*. Density.
14*. Enthalpy.

$$K_y = \frac{K_W}{(1-y)^2}$$

In air conditioning applications, differences of up to approximately 4% may be expected between the two coefficients. In the present work, K_W is used exclusively.

[26] Saturation vapour pressure and saturation humidity ratio refer to a sample of moist air saturated at the same dry bulb temperature as the reference sample, and are not strictly speaking properties of the sample.

The Prandtl number and the Schmidt number are entirely determined by the thermodynamic and transport properties of an air sample, and may thus themselves properly be regarded as properties of the air sample. Lewis number is also a function of the convection processes which the sample is undergoing. However, as will be shown, it is also conveniently computed within the current framework.

15*. Dew-point temperature.
16. Specific heat.
17. Dynamic viscosity.
18. Thermal conductivity.
19. Mass diffusivity.
20. Prandtl number.
21. Schmidt number.
22. Lewis number.

Since moist air may be treated as a mixture of *two* gases, the thermodynamic and transport properties of a sample will be uniquely determined if *three* of the psychrometric properties of the sample are known. Within the context of the present study, all air properties are evaluated at the prevailing atmospheric pressure, which is specified[27]. For present purposes it may also be assumed without loss of generality that dry bulb temperature is either specified, or will be calculated as an integral part of the computations involved in taking an air sample through a psychrometric process. If any one of the asterisked properties from the list above is also known, the state of the air sample will be uniquely determined.

Fluid properties are declared to be of class ***air_property***, which provides a mechanism for keeping track of which fluid properties are known at a point, in the sense that they have either been specified or calculated for that point. Each of the properties listed above is represented by type ***air_property***, with two exceptions. Pressure is specified as part of the problem domain, and may be assumed to be always known at a point, while the Lewis number is a function not only of more basic air properties, but also of the convection mechanism operating. Objects of class ***air_property*** contain two fields, one containing the value of the fluid property, and the other indicating whether the value of the property is known.

If the value of the dry-bulb temperature and one[28] other asterisked ***air_property*** are known at a given point, the value of every other ***air_property*** (and the Lewis number) can be calculated at that point. An object of class ***AirState*** in this condition is flagged as *calculable*; an attempt to calculate the properties of an object for which this flag is *false* will result in an error. Two constructors are provided for class ***AirState***. The first takes no arguments apart from the pressure, and constructs an object for which every ***air_property*** is flagged as not known, and the object itself is flagged as not *calculable*. The alternative constructor enables dry and wet bulb temperatures, as well as pressure, to be specified, and creates an object which is *calculable*. An object of class ***AirState*** can be made *calculable* by one of two means; by specifying two fluid properties (usually dry bulb

---

27 The constructors for class ***AirState*** take pressure as an argument. If omitted, this defaults to standard atmospheric pressure (101325 Pa).

28 Saturation properties (vapour pressure of water, and humidity ratio) can be calculated on the basis of dry-bulb temperature alone, as can the mass diffusion coefficient. The mole fractions of air and water, together with the dew-point temperature are functions of humidity ratio alone. However, within the present computational scheme, dry-bulb temperature must also be known before these latter properties can be calculated.

temperature and one other asterisked property, in that order), or by a process proceeding from another *calculable* state. An object can be rendered not *calculable* by changing one fluid property only.

4.3.1. Accessor Functions.

Member functions are provided to access each of the listed fluid properties for a *calculable* object of class *AirState*. These have declarations of the form:

```
double AirState::WetBulbTemperature ();
```

In designing the class it was recognized that only a small subset of the properties listed are of any interest at any given point in the air conditioning cycle. For reasons of computational efficiency, fluid properties are thus only calculated as requested by a call to the appropriate accessor function, or as an intermediary in the calculation of other properties. When an accessor function is called, the value of the requested property is returned if it is known; otherwise it is calculated by a computationally direct method on the basis of the properties which are already known. The task of developing an efficient strategy for the computation of any fluid property given the dry bulb temperature, and an arbitrarily specified second property, was simplified by the recognition that humidity ratio is readily calculated using any of the other asterisked properties, and forms a useful intermediary in the computation of the remaining properties.

A strategy for the computation of the listed properties is described in the following notes. In the software implementation, properties required as intermediaries in the computation of some other property are accessed by calling the appropriate accessor function, rather than using the relevant variable directly, and are thus calculated as needed. This point is vital to understanding the following strategy.

1. ***Pressure*** and ***Dry Bulb Temperature*** cannot be calculated. Pressure is always known. An attempt to access an unknown dry bulb temperature is an error.

2. ***Humidity Ratio*** is calculated by searching through a list of other properties until a known property is found, and then calculated as follows:

    a. *Mole fraction of air* or *mole fraction of water* are known; humidity ratio is calculated using equations (4.1.2) and (4.1.3).

    b. *Vapour pressure of water* is known; humidity ratio is calculated using equation (4.1.8).

    c. *Degree of saturation* is known; humidity ratio is calculated using equation (4.1.12).

    d. *Relative humidity* is known; equation (4.1.13) is used to calculate the vapour pressure of water, whence humidity ratio can be calculated using equation (4.1.8).

e. *Wet bulb temperature* is known; humidity ratio is calculated using equation (4.1.30).

f. *Specific volume* is known; humidity ratio is calculated using equation (4.1.21).

g. *Density*is known; humidity ratio is calculated using equations (4.1.21) and (4.1.22).

h. *Enthalpy* is known; humidity ratio is calculated using equation (4.1.25).

i. *Dew point temperature* is known; humidity ratio is calculated using equation (4.1.15).

3. ***Mole Fraction of Air*** can be determined using equation (4.1.2), provided ***Mole Fraction of Water*** is known, and *vice versa*; otherwise equations (4.1.2) and (4.1.3) are used to calculate these properties.

4. ***Saturation Vapour Pressure*** is calculated using equation (4.1.4) or (4.1.5), as appropriate.

5. ***Vapour Pressure of Water*** is calculated using equation (4.1.8).

6. ***Saturation Humidity Ratio*** is calculated using equation (4.1.10), with the enhancement factor calculated using equation (4.1.11).

7. ***Degree of Saturation*** is calculated using equation (4.1.14), provided relative humidity is already known. Otherwise the definition (equation 4.1.12) is used.

8. ***Relative Humidity*** is calculated using equation (4.1.14), provided degree of saturation is already known. Otherwise it is calculated as the ratio of vapour pressure of water to saturation vapour pressure at the same temperature and pressure (equation 4.1.13).

9. ***Wet Bulb Temperature*** is found by inverting equation (4.1.30). Equation (4.1.31) is used to provide an initial estimate, whence a bracketing interval can be found using a procedure proposed by Swift and Lindfield (1978). Equation (4.1.30) can then be solved using a zero-finding algorithm (Brent, 1971).

10. ***Specific Volume*** is calculated using equation (4.1.21).

11. ***Density*** is calculated using equation (4.1.22).

12. ***Enthalpy*** is calculated using equation (4.1.25).

13. ***Dew Point Temperature*** is calculated directly using the Peppers equations (4.1.16 and 4.1.17) and then refined using equation (4.1.15).

14. ***Isobaric Specific Heat*** is calculated by finding the specific heats of water vapour and air separately using equation (4.1.27), and combining them in the correct proportions using equation (4.1.26).

15. ***Dynamic Viscosity*** is calculated by finding the viscosities of water vapour and air separately using equation (4.2.3), and combining them according to equation (4.2.1). Note that if the thermal conductivity is already known, the component viscosities, together with the coefficients (Φ_{ij}), will already have been calculated. These are stored as private variables of class *AirState* to ease the computational burden.

16. ***Thermal Conductivity*** is similarly calculated by separately finding the properties for water vapour and air using equation (4.2.5), and combining them according to equation (4.2.4). The remarks recorded above for dynamic viscosity concerning the coefficients (Φ_{ij}) are applicable here also.

17. The ***Mass Diffusivity*** coefficient is calculated using equations (4.2.6-4.2.8).

18. ***Prandtl Number*** and ***Schmidt Number*** are calculated by calling the accessor functions for the component properties (equations 4.2.9 and 4.2.10).

19. ***Lewis Number*** is calculated as the ratio of the Schmidt number to the Prandtl number raised to the appropriate power (equation 4.2.11). The accessor function takes the coefficient c as a sole argument. If omitted, this defaults to a value of ⅓, which is appropriate for the forced convection situation.

4.3.2. Functions to Set Fluid Properties.

A member function is provided to set each of the asterisked fluid properties. These functions have declarations of the form:

```
void AirState::WetBulbTemperature (double wbt);
```

Setting the dry bulb temperature will result in all other fluid properties being flagged as not known, the object of class *AirState* correspondingly being flagged as not *calculable*. Setting any other asterisked fluid property by this method will leave the dry bulb temperature, and any properties which are solely dependent on dry bulb temperature in their prior state, and will flag all other properties as not known. Thus, provided the dry bulb temperature is known, a *calculable* object results. When it is desired to set or change the value of the dry bulb temperature and one other property using this method, dry bulb temperature should be set first.

Calculations involving the *dbt-h-W* relationship (equation 4.1.25) occur extensively in computational operations involving the air conditioning cycle. For convenience, a set of functions are provided which enable two of the three properties to be specified, while returning the corresponding value of the third. These three functions are declared as follows:

```
double AirState::Enthalpy (const double W, const double dbt);
```

4.3.3. Process Functions.

Many psychrometric processes of interest occur under conditions in which the dry bulb temperature or the humidity ratio remain constant. Other processes, particularly those involving the load ratio line, may be conveniently treated as a combination of two processes, one occurring at constant dry bulb temperature, and another occurring at constant humidity ratio. Functions are provided, based on the *dbt-h-W* relationship, to enable one *calculable* air state to be transformed into another by a process which holds one property constant, while incrementing another. Thus, a member function to increment dry bulb temperature, while keeping humidity ratio constant, is declared as follows:

```
void AirState::CHR_DbtIncrement (const double dt);
```

The functions provided enable the user to specify an increment in enthalpy or dry bulb temperature, while keeping humidity ratio constant, or an increment in enthalpy or humidity ratio, while keeping dry bulb temperature constant.

The present complement of such functions is adequate for the computations described in the following. The class might be usefully extended to broaden its range of applicability by adding further process functions. Functions relating to the adiabatic saturation process would be particularly useful.

4.3.4. Mixing of Air Streams.

If two air streams (1 and 2), having mass flow rates $\dot{m}_1$ and $\dot{m}_2$, mix adiabatically, then the state of the resulting airstream (3) will be defined by the relationships

$$\dot{m}_3 = \dot{m}_1 + \dot{m}_2 \qquad \textbf{(4.3.1)}$$

$$h_3 = \frac{\dot{m}_1 h_1 + \dot{m}_2 h_2}{\dot{m}_3} \qquad \textbf{(4.3.2)}$$

$$W_3 = \frac{\dot{m}_1 W_1 + \dot{m}_2 W_2}{\dot{m}_3} \qquad \textbf{(4.3.3)}$$

The above relationships are implemented in a **static** member function, which takes as arguments pointers to the component air states, and the resultant air state, together with the mass flow rates for the mixing air streams, and calculates the air state for the resulting stream. The mass flow rate of the resulting air stream is returned:

```
static double AirState::Mix (AirState& a1,
                             const double m1,
                             AirState& a2,
                             const double m2,
                             AirState& a3);
```

Chapter 5. Air Conditioning Systems.

The air conditioning system may be taken to comprise the equipment items, together with the associated control strategies which the designer assembles to maintain a specified set of thermal environmental conditions within a zone, or a group of zones. The ability to predict the likelihood of a system meeting the design specifications depends critically on knowing how a system will respond to a changing diversity of loads in the various zones served by the system. Conventional design practice, with its emphasis on design for peak load, frequently overlooks or ignores the possible problems which may arise when the system is operated at part load.

The purpose of this chapter is to provide a brief review of the types of all-air systems available to meet a specified set of zone conditions. A range of systems is considered in terms of physical layout and general operating characteristics. The discussion at this point is of a qualitative nature; a detailed comparison of system types based on the results of a modelling study will be presented in chapter 13. Conventional systems are considered first. Attention here is focussed on the Constant Air Volume (CAV) and Variable Air Volume (VAV) systems, with and without reheat capability. Following this, alternative design strategies derived from work undertaken at the University of Adelaide are presented. These relate specifically to the Low Face Velocity/High Coolant Velocity (LFV/HCV) and the High Driving Potential (HDP) methodologies for air conditioning.

5.1. All-Air Systems.

The following discussion is concerned with all-air systems, which are characterized by the fact that cooling is achieved by circulating air as the sole working fluid to the conditioned zones. All-air systems may be divided into two broad classes, single-duct and dual-duct systems. ASHRAE (1987, chapter 2; see also chapter 51) describes a broad range of systems in both categories, and discusses the relative merits of each in some detail[29]. The present discussion is restricted to single-duct systems, and within that category is further restricted so that the various forms of induction system (fan-powered and unassisted) are ignored. In the following we shall consider constant air volume (CAV) and variable air volume (VAV) systems in both single and multizone applications. We shall also be concerned with the use of overcooling and reheat to attain specified zone humidity levels for both CAV and VAV systems. Section 5.2 describes the high driving potential (HDP) system of air conditioning as a logical development of the concepts developed in this

---

[29] Dual-duct systems are infrequently specified nowadays (McQuiston and Parker, 1994) owing to the fact that they are disadvantaged in respect of both capital cost and operating cost by comparison with their single-duct counterparts. Induction systems have been omitted from the discussion on the grounds that they do not readily fit into the design methodology and computational framework to be developed later in this, and in subsequent chapters. Note however that use of induction systems on the grounds of energy efficiency is recommended by some practitioners (see for instance the paper by Wessel in Gupta et al., 1987). The properties of the induction system should be assessed in a future study.

section. These two sections together provide a framework for the last two sections of this chapter, which develop a general set of data structures and algorithms to simulate systems based on the principles of these two sections.

5.1.1. Constant Air Volume Systems.

In the CAV system the volume air flow remains constant for all loads, water flow being modulated to maintain the required dry bulb temperature in the zone. The temperature differential across the zone for a given sensible load may be found according to the relationship

$$q_s = \dot{m} C_{p,a} \Delta t \qquad \textbf{(5.1.1)}$$

where the appropriate value for $C_{p,a}$ is a mean value integrated across the range of temperature and humidity ratio encountered in the zone; operationally, Δt may more accurately be calculated using relationships to be developed in section 6.1.

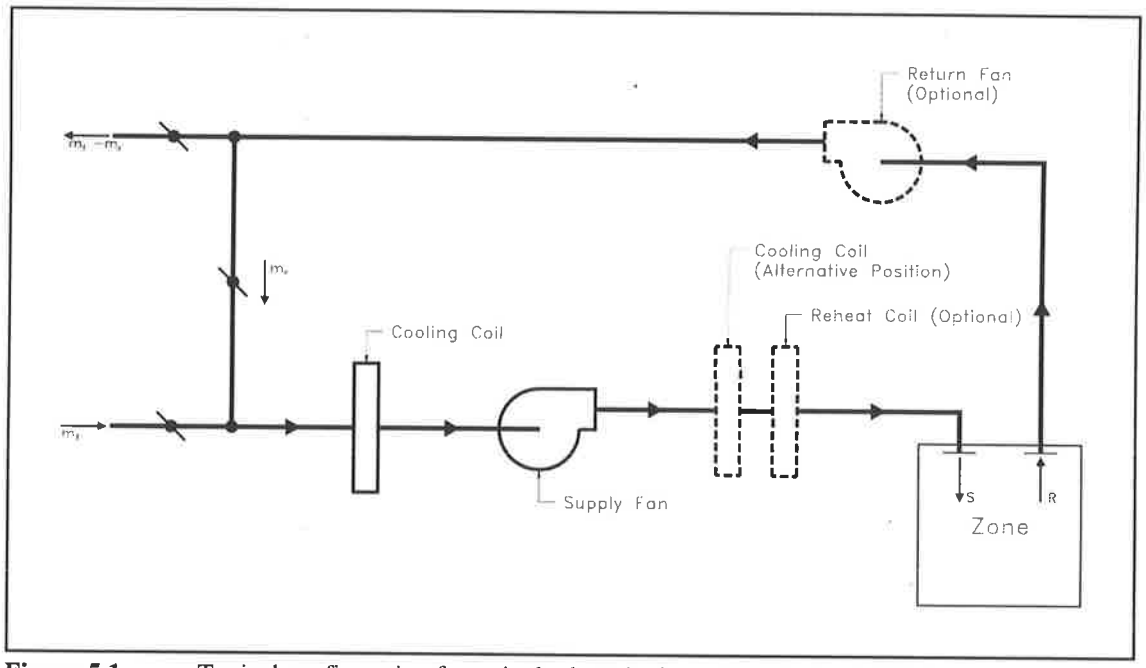


Figure 5.1. Typical configuration for a single-duct single-zone constant air volume system.

The configuration of a single-zone CAV system is shown schematically in figure 5.1. Two alternative positions for the supply air fan are shown in the diagram. With the fan placed downstream of the cooling coil (*draw-through* configuration), the power dissipated by the fan will be injected into the airstream in the form of reheat, thus marginally assisting dehumidification. Conversely, placing the fan upstream of the cooling coil (*blow-through* configuration) will result in preheating of the airstream, which will hinder dehumidification. Generally speaking, a draw-through configuration will produce more even distribution of the air flow across the face of the coil, and is to be preferred on these grounds.

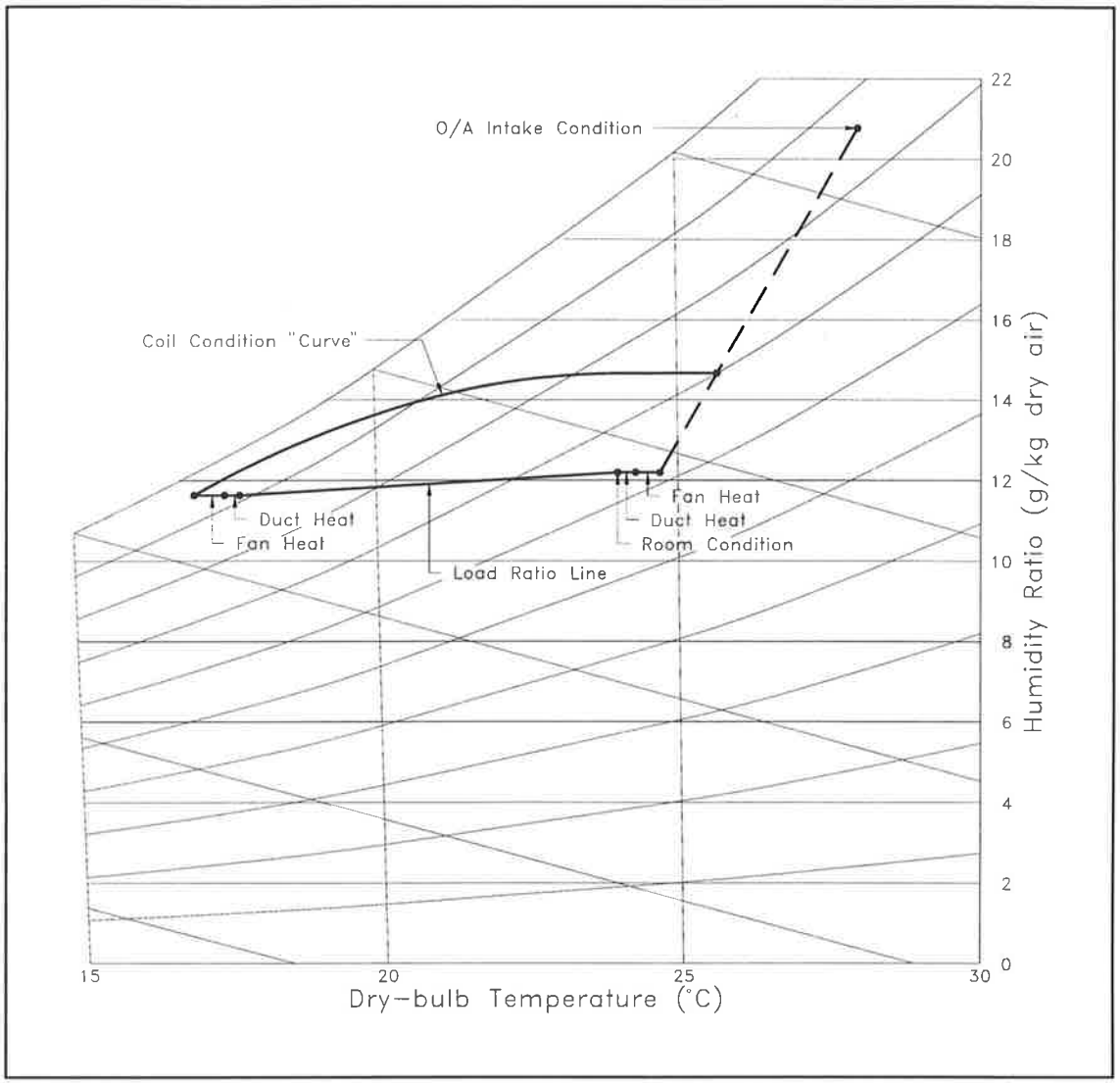


Figure 5.2. Psychrometric processes associated with the single-duct single-zone CAV system shown in figure 5.1.

The psychrometric processes occurring within a typical single-zone system with a draw-through unit are as shown in figure 5.2. Note in particular the additional heat loads imposed by:

i. Dissipation of supply air fan energy (q_{sf}).
ii. Heat gain due to transmission through the supply air duct wall (q_{sd}).
iii. Heat gain due to transmission through the return air duct wall (q_{rd}).
iv. Dissipation of return air fan energy (q_{rf}).

Further loads may be imposed by air leakage through the duct walls, and "leakage" involving equipment such as mixing boxes. These will not be considered further in this study.

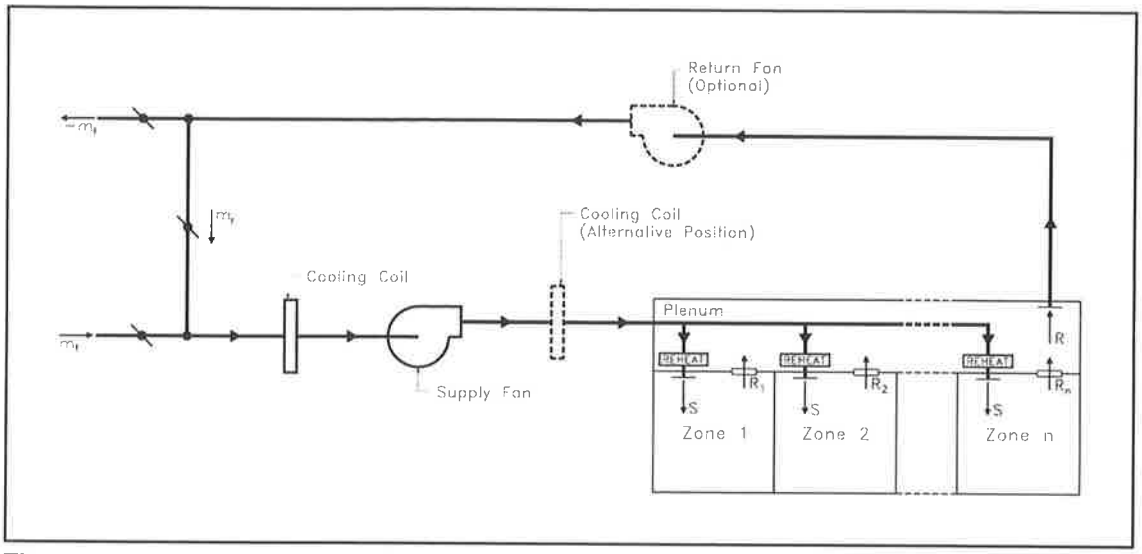


Figure 5.3. Typical configuration for a single-duct multizone constant air volume (reheat) system with a single cooling coil serving all zones.

It is an inherent characteristic of CAV systems that air quality, as determined by humidity levels, deteriorates markedly at part-load. A primary cause of this effect is the reduction in coil dehumidification potential at part-load caused by throttling the coolant flow, a feature which is common to all conventional systems. In CAV systems the effect is exacerbated by a rise in supply air temperature, which increases the equilibrium dew-point temperature for the zone. Control over zone humidity may be achieved by reheating the supply air using steam or hot water coils, or electrical power, causing the cooling coil to cool the supply air to a lower temperature than would be required to offset the actual zone load. By this device, usually referred to as overcooling and reheating, or simply as reheat, zone humidity may be modulated within a range limited asymptotically at the lower end by the surface dew point temperature of the cooling coil at the expense, of course, of increased energy consumption.

The CAV control methodology may be adapted to serve multiple zones. Two basic configurations are possible. The system of figure 5.3 employs a single cooling coil, with reheat coils in the duct servicing each zone. Such systems may be configured using either draw-through or blow-through units. The psychrometric processes occurring within such a system are as shown in figure 5.4. In this diagram, the condition of the air at entry to the various zones is denoted by the points labelled with an *r*; that of the air leaving the zones by the points labelled with a *z*. A common return plenum and duct system is shared by all zones. For all multizone CAV systems the fan unit must be selected on the basis of the air quantity required to offset the sum of the individual peak loads for the various zones, for a specified supply air temperature. In the most basic form of the system described above, the chilled water flow rate is modulated to maintain the supply air temperature at a constant level, reheat being employed to maintain specified temperatures in zones subject to off-design conditions. A load analyzer may be used to conserve energy by resetting the supply air temperature to satisfy the demands of the zone with the highest cooling load. The energy comparisons provided later in this study assume that such a strategy is always available. The use of reheat which is inherent in the control strategy employed by these systems has resulted in their commonly being referred to as *reheat* systems.

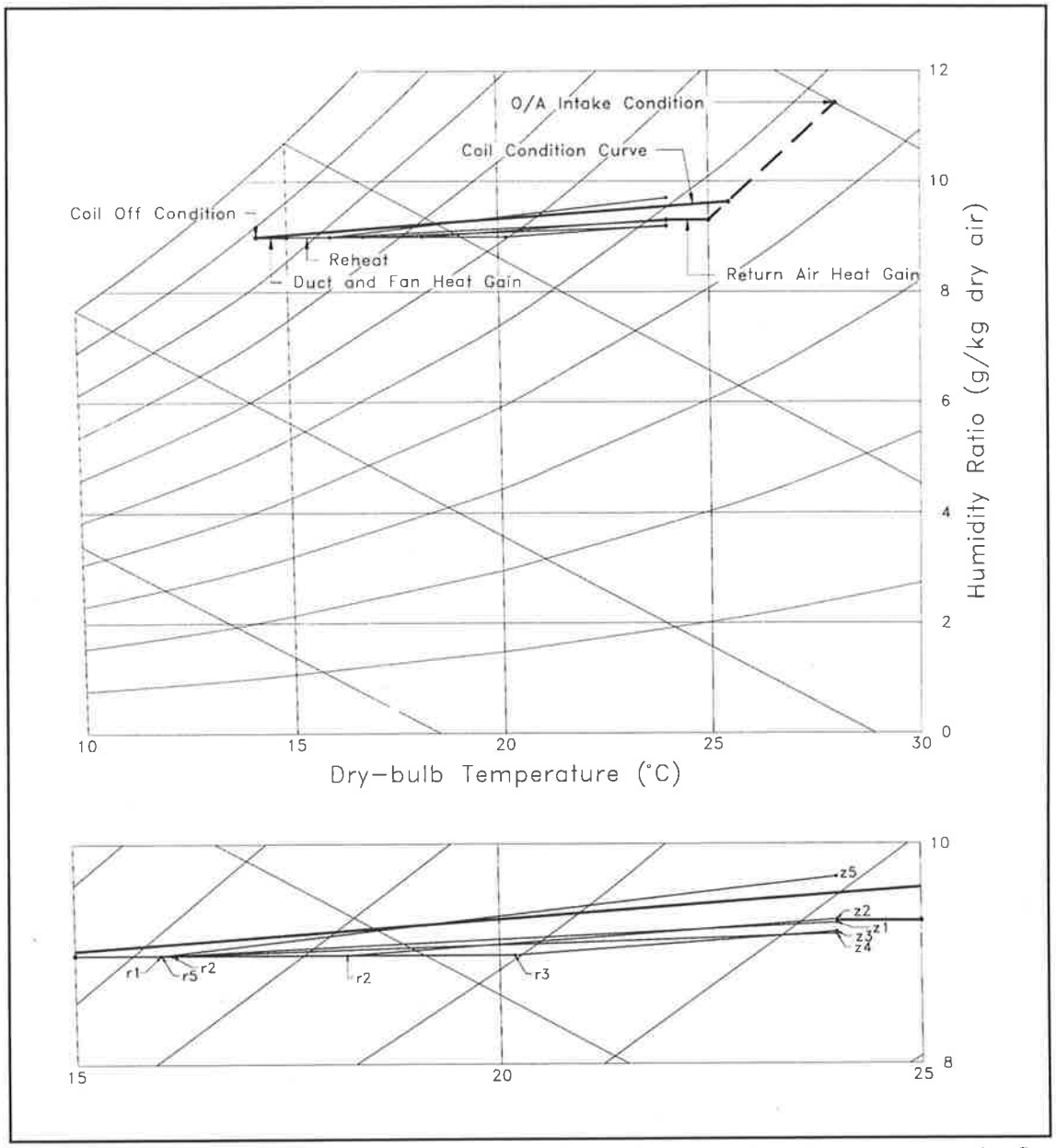


Figure 5.4. Psychrometric processes associated with a single-zone multizone CAV system serving five zones, using the configuration shown in figure 5.3.

By employing a separate cooling coil for each zone as shown in figure 5.5, the need to reheat the supply air to off-peak zones to attain the design room conditions is obviated. Such a system is necessarily configured using a blow-through unit. However, the decoupling between the various zones is not complete. Since all zones share a common return air plenum and duct, the return air from the various zones is mixed prior to being apportioned to the respective supply air coils. Thus, humid return air from off-peak zones will contribute to the latent load on the coils serving zones which are at or near peak. The processes associated with such a system are shown on the psychrometric chart of figure 5.6. In the lower part of this diagram, the coil-off points for the separate coils are marked with a *c*. It should be noted that the configurations of figures 5.3 and 5.5 are not

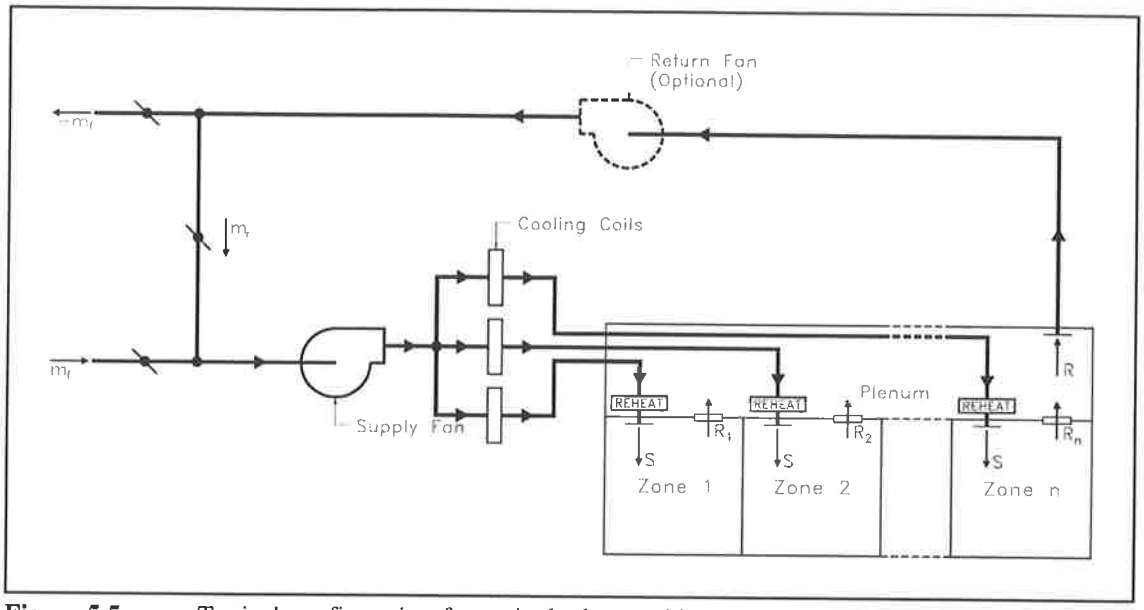


Figure 5.5. Typical configuration for a single-duct multizone constant air volume system with an individual cooling coil serving each zone.

necessarily mutually exclusive. In the most general case, a unit may contain several supply air coils, each serving a single zone or a group of zones.

5.2.2. Variable Air Volume Systems.

VAV systems maintain a constant supply air temperature under all load conditions, air flow rate being modulated to offset the zone loads in accordance with equation (5.1.1). The remarks made in connection with that equation concerning an appropriate value for $C_{p,a}$ are equally applicable here; the mass flow rate of air required to offset a given load for a given room condition may be more accurately determined within a computational setting using equation (6.1.9).

The layout of a typical multizone VAV system is shown schematically in figure 5.7, the corresponding psychrometric chart being shown in figure 5.8. Supply air temperature is controlled by a thermostat downstream of the supply air fan modulating the water flow to the cooling coil. Zone temperature is sensed by a thermostat within the space, and used to control the air flow to the space to that required to offset the loads. The load ratio lines for the individual zones are annotated separately in the lower part of figure 5.8, as has been the case with the systems described previously. The return air from the various zones is typically mixed in a common plenum, and returned via a common duct system.

Reheat is no longer required to control temperatures in off-peak zones, although it may still be required to achieve specified humidity levels at part load. In a conventional VAV system, the reduction in dehumidification potential at part-load caused by throttling the coolant flow will result in increasing room humidity, as is the case with the conventional CAV system. However, since the supply air temperature is maintained constant, air quality deteriorates far less rapidly under part-load conditions than is the case with CAV systems,

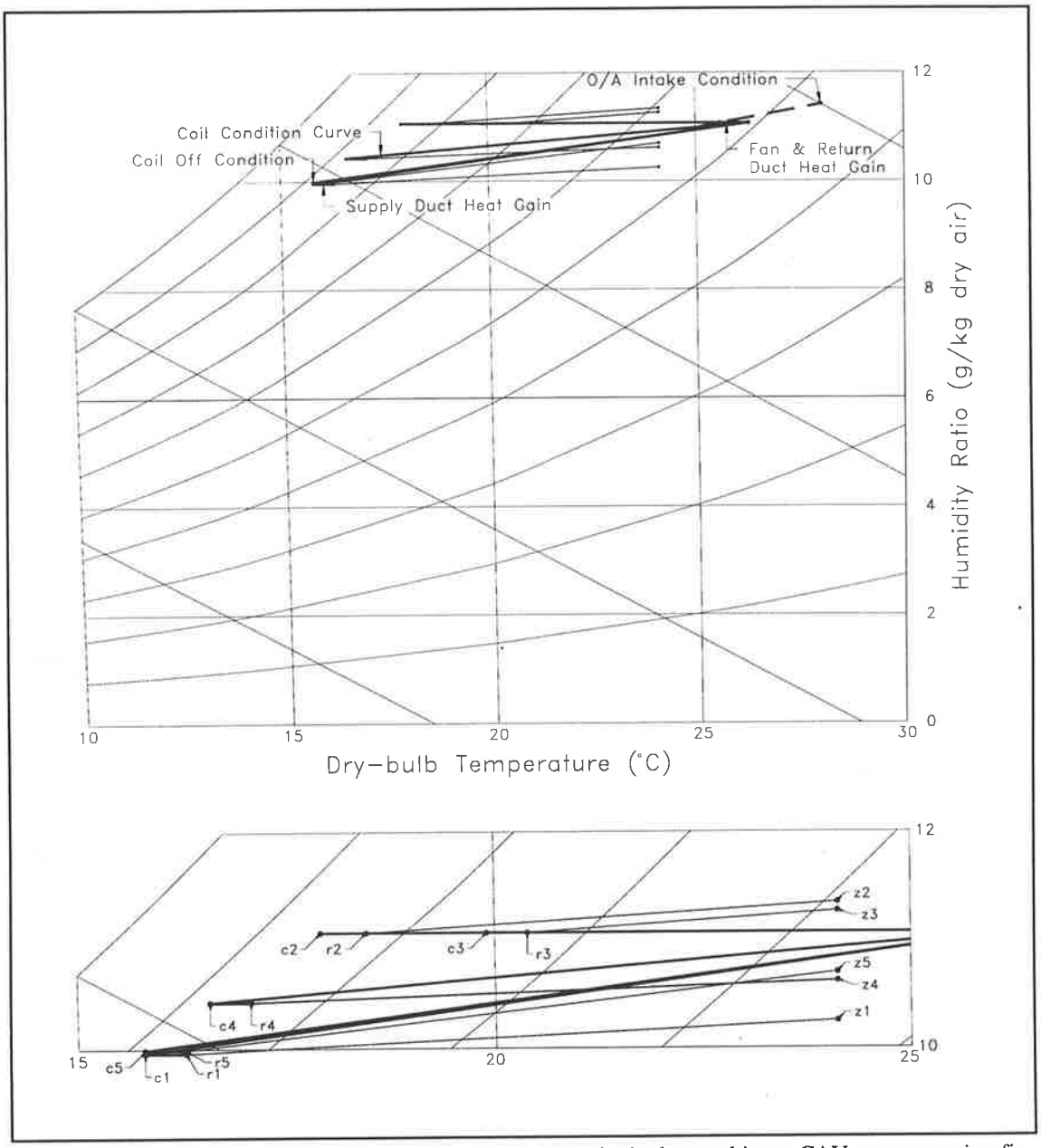


Figure 5.6. Psychrometric processes associated with a single-duct multizone CAV system serving five zones, using the configuration shown in figure 5.5.

and the reheat and associated overcooling required to achieve a specified humidity level is therefore less. Reheat is sometimes applied under conditions of reduced load to maintain air flow rate above a lower limit, and hence to minimize air distribution problems which may be associated with very low air flow rates (ASHRAE 1987, chapter 2). An alternative strategy to achieve the same end is that of resetting the supply air temperature upwards at reduced load, emulating to some extent the part-load behaviour of CAV systems. Supply air reset reduces part-load air quality, while reheat is wasteful of energy. In most cases, both can be avoided by correct design procedures (Gupta et al., 1987). Where reheat is used to control humidity two strategies are possible, as shown in figure 5.7. Reheat may

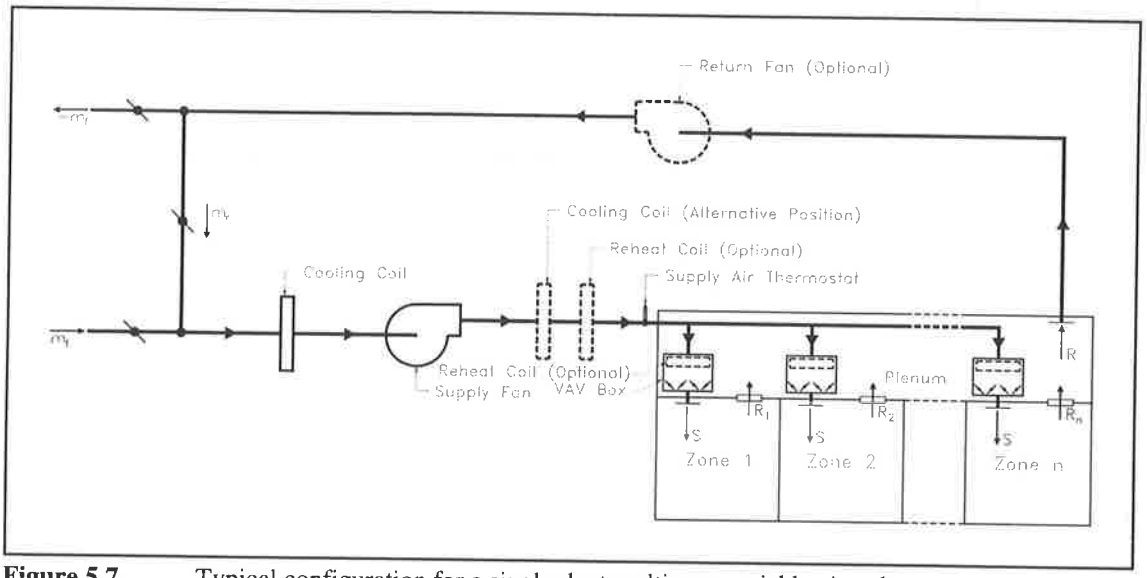


Figure 5.7. Typical configuration for a single-duct multizone variable air volume system.

be applied using a single heating coil located upstream of the supply-air thermostat[30]. Alternatively, the humidity may be controlled selectively on a zone-by-zone basis using individual heating coils for each of the controlled zones. In the modelling strategy developed in the next chapter, only the former case will be considered. Another method of maintaining air flow to a zone at low load is by use of a fan in the VAV box which draws return air from the ceiling plenum and mixes it with the supply air to raise the supply air temperature. This is a form of reheat. The system displaces some ventilation air in the zone and therefore can degrade air quality.

Since the various zones served by an individual unit will peak at different times of the day, air may be diverted from one zone to another as required. For a VAV system the fan is sized to meet the airflow required to offset the simultaneous peak load for the set of zones served by the fan, rather than the sum of the individual peak loads for the various zones. As a consequence, the component of the operating cost attributable to fan power in a VAV system is reduced by comparison with CAV systems, often substantially. Capital savings are also achieved by allowing the specification of a smaller fan. Unfortunately, these latter savings may be more than offset by the increased cost of controls and equipment to regulate fan speed and air distribution efficiently. With current tendering procedures favouring low first cost at the expense of environmental quality and operating cost, VAV systems have by no means superseded their CAV counterparts, in spite of inherently superior performance on both counts.

---

30 Variable air volume control may be applied to a unit using individual cooling coils for each zone, or group of zones, in a manner similar to that shown in figure 5.5. The reheat strategy described may be applied in this case by providing a reheat coil to match each cooling coil.

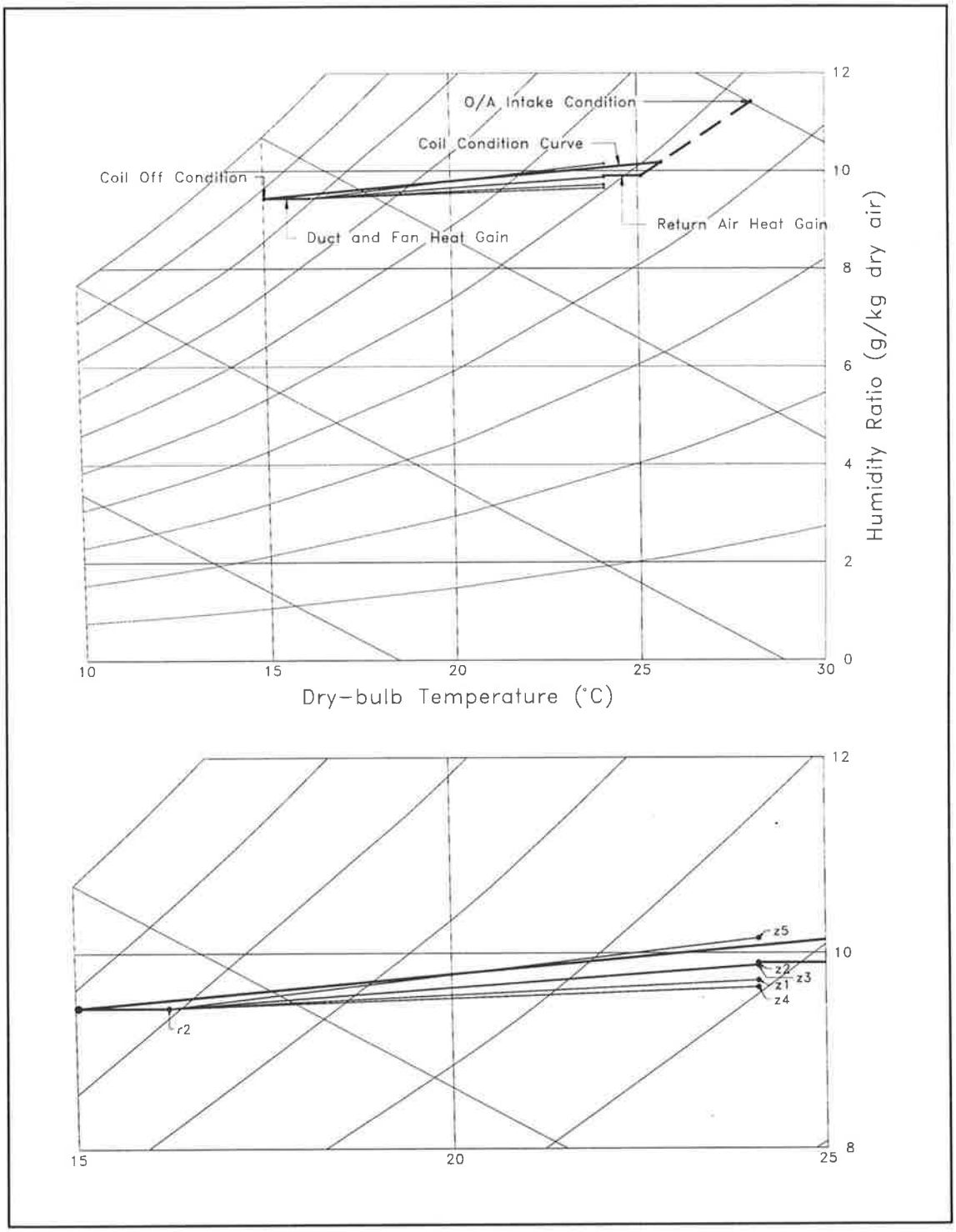


Figure 5.8. Psychrometric processes associated with a single-duct multizone system serving five zones, using the configuration shown in figure 5.7.

5.2. Improving Dehumidification in All-Air Systems.

As described in Chapter 3, the major rôle played by excessive room humidity in contributing to the sick-building syndrome is now well documented. In most situations the task of designing a system which will maintain humidity levels below a specified upper limit for zones subject to peak or near-peak loads is relatively straightforward. In conventional practice, the major emphasis is in fact on designing to provide satisfactory performance at peak load, part-load performance seldom being specified and hence receiving little if any attention. And yet it is axiomatic that part-load conditions will prevail throughout the greater part of the life-cycle of an air conditioning system. It is also at such part-load conditions that humidity problems are likely to occur. To accommodate a falling sensible load the supply of coolant to the cooling coil is throttled. But this reduces the slope of the coil condition curve; that is, it reduces the dehumidification potential of the coil. In CAV systems the situation is exacerbated by the fact that the coil-off condition will ride up the coil condition curve as the load falls and the supply air temperature is required to rise.

Paradoxically, the *demand* for dehumidification is usually greatest under part-load conditions. Consider the case of a typical office building. Occupancy, and hence *latent* load, may be expected to remain almost constant for the greater part of the time the system is operating. In perimeter zones in particular however, the *sensible* load will reduce as the transmission and solar loads decrease under part-load conditions, leading to a decrease in room sensible heat ratio. Thus, the ability of the system to dehumidify is reducing at the same time as the zone is requiring an increase in dehumidification per unit of sensible cooling to meet specified (or comfortable) humidity levels. Of course, the situation can always be corrected by overcooling the supply air to the design dew-point temperature, and then reheating, as was standard practice prior to the energy crises of the seventies. This practice is far less acceptable in the more energy conscious world of today, although it remains prevalent in the minds of designers whenever maximum humidity levels are specified at all times. Where humidity levels are not specified or are specified only at a given design condition, usually peak load, the onus is on the designer to achieve the humidity levels recommended in standard codes of practice by judicious design in the first instance, rather than by implementing energy-consuming fixes to fundamentally flawed systems.

At the heart of the dehumidification process is the cooling coil, and it is no exaggeration to say that the success of a design will be fundamentally affected by the designer's understanding of the processes occurring within this component. In the overall design process however, the cooling coil conventionally receives little attention. It is common practice for the designer to draw up a set of functional specifications based on peak-load conditions only, and leave the detailed task of selection to the supplier of the dehumidifier or air handling unit containing the dehumidifier.

The cooling coil has been the subject of an extensive experimental programme at the University of Adelaide, designed to describe and quantify the processes occurring within the coil, and to develop a rational scheme to parameterize the performance of a coil across its range of operation. In the course of the study a number of misconceptions regarding the

behaviour of cooling coils have been identified. Many of these can be traced to a misinterpretation of basic thermodynamic data and principles. These are reviewed in Shaw and Luxton (1988b), while Luxton and Shaw (1991) describe some of the more recent findings concerning the processes occurring within the dehumidifier. These will be introduced as appropriate within the following sections, and in Chapter 7.

Application of the principles enunciated during the above experimental programme has led to the development of two novel design methodologies, the Low Face Velocity/High Coolant Velocity (LFV/HCV) methodology and the High Driving Potential (HDP) methodology. It should be emphasized from the outset that these methodologies are not mutually exclusive. In addition, system design according to the principles embodied within these methodologies may be accommodated within a conventional framework. The LFV/HCV methodology is particularly suited to retrofits. The full benefits of the proposed methodologies in terms of increased rentable floor space and the ability to handle a diversity of zone loads will, however, be realised only if they are employed within the logical framework, or methodology, which has been developed.

5.2.1. LFV/HCV Systems.

The factors contributing to the poor part-load performance of conventional air conditioning systems have been described briefly above, and in greater detail by Shaw and Luxton (1988a). The LFV/HCV methodology represents a comprehensive means of rectifying the situation based on a reassessment of the fundamental principles which form the basis of conventional practice. The underlying principles of the new methodology may be readily understood by reference to figure 5.9, which schematically shows the temperature distribution in a heat exchange process between two fluids, a hot fluid (moist air), and a cold fluid (chilled water). T_1 and T_2 are the free-stream temperatures for the warm and cold working fluids respectively, while T_s is the temperature at the outside surface of the cold tube. Because the dew point temperature of the moist air is (usually) less than T_1, it is necessary to bias the surface temperature towards the temperature of the cold working fluid ($T_s \Rightarrow T_2$) to maximize dehumidification. This is equivalent to minimizing the water temperature rise through the coil. In the limiting case of zero water temperature rise, a situation approaching that which obtains in a DX coil, the coil characteristic becomes a straight line of maximum possible slope. In an operational situation, a lower

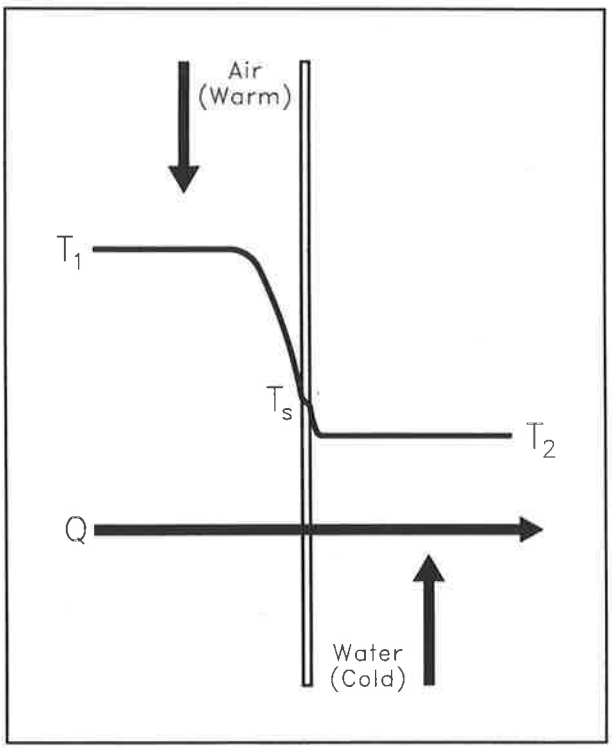


Figure 5.9. Temperature distribution across a tube wall separating cold and warm fluid streams.

bound of approximately 3°C is imposed on the water temperature rise to avoid difficulties in controlling the chiller.

To bias the surface temperature T_s towards the cold fluid temperature T_2, it is necessary to maintain a high heat transfer coefficient on the water side, and a low heat transfer coefficient on the air side. By a combination of reasoning and experimentation, it has been found that the desired situation may be achieved by designing for:

i. Low air face velocity. A face velocity at peak load of 1-1·5 m/s or less would be typical for an LFV/HCV system, as opposed to velocities of 2-2·5 m/s or higher, which are typically encountered in conventional practice.

ii. High coolant velocity. Velocities of 1·5-2·2 m/s would be appropriate to LFV/HCV operation. In conventional practice, peak water velocities of less than 1 m/s are not uncommon.

iii. High primary to secondary surface area. This can be achieved by using a low fin density. In practice, manufacturing difficulties impose a limit of 6 fins/inch (fpi) as the lowest fin density which can economically be used in air conditioning applications. Fin densities of 6 and 8 fpi will adequately cover most situations which are likely to be encountered; coils with fin densities in the range of 9-14 fpi are commonly specified in conventional practice.

The combination of low face velocity with low fin density will significantly reduce the air pressure drop through the coil, thus reducing fan power consumption, and potentially offering the opportunity to specify a smaller fan. The use of lower fin densities offers additional benefits in that such coils are less prone to inundation under conditions of high dehumidification, and are easier to clean.

Historically, the effect of low face velocity on dehumidification potential was considered first in the University of Adelaide studies, in isolation from the remaining parameters which affect coil performance. The results of these investigations are reported in a series of papers by Shaw (1982a,b; 1985). Apart from designing to minimize peak air face velocity, subject to space constraints, the designer has little opportunity to influence part-load performance directly using face velocity, which remains constant across the range of operation of a CAV unit and decreases for a VAV unit as the sensible load decreases. Manipulation of the coolant velocity by judicious circuiting does, however offer the designer rather more scope to control humidity levels.

Typically, the conventional designer will select a coil for a water temperature rise of 7-8°C under peak load conditions, this value being determined largely by rule of thumb, but having some substance in terms of chiller operation. The result of using such a high water temperature rise, in combination with a relatively high face velocity, is to produce a peak room condition which is significantly more humid than would be produced by an LFV/HCV coil selected for the same duty. If the LFV/HCV system incorporates staging (see below) to limit room humidity levels across the range of operation of the unit, further improvement may be possible by specifying a lower supply air temperature than would be

normal for the conventional system. Notwithstanding the foregoing discussion, it **is** possible to design systems which will satisfy comfort standards at peak load for all but the most demanding applications, using conventional design methods. The situation changes at part load. Throttling of the coolant flow invariably leads to an increase in water temperature rise, and a decrease in dehumidification potential. As noted previously, CAV systems fare much worse at part load than VAV systems, the latter benefiting from the low face velocity principle because the air flow is throttled and hence the face velocity falls as the load reduces. Neverhteless, in all cases the decrease in the air-side heat transfer coefficient is more than offset by a corresponding decrease in the water-side coefficient, with the result that room humidity levels will rise monotonically with reducing load. Thus, a conventional system which is marginally satisfactory at peak load, may well fail to satisfy comfort specifications during the far more common part-load conditions.

LFV/HCV systems are also susceptible to the same part load behaviour as described above. Comparing the LFV/HCV and the conventional systems at corresponding loads across their range of VAV operation, the LFV/HCV system will achieve a lower humidity at the peak load condition and, as the sensible load decreases will track its conventional counterpart, maintaining a near-constant difference in the room humidity. For most applications, the inherently superior dehumidification offered by the LFV/HCV unit will be sufficient to ensure that comfort standards are always met. Under particularly demanding conditions, such as when the dynamic load range is very large, certain load combinations can cause humidity constraints to be violated. Under such conditions the conventional designer has no option but either to ignore the fact that room conditions are no longer satisfactory, or to resort to overcooling and reheating in which case a double energy penalty is incurred. The LFV/HCV methodology, by contrast, offers an alternative mechanism for satisfying humidity constraints at part load without incurring an energy penalty. If the high coolant velocity can be renewed once it falls below a certain threshhold[31], the high dehumidification potential will be restored. In practice this is achieved by reducing the effective size of the coil by deactivating blocks of circuits, preferably from the downstream (with respect to the airflow) rows of the coil. To offset the sensible load with the smaller effective coil size the coolant velocity through the remaining circuits must increase and hence the active coil surface becomes colder and dehumidification potential is restored. This process is referred to as *staging*.

The consequences of staging can best be understood by reference to an example, that of a lecture theatre in the tropics which is subject to large variations in occupancy level[32]. For a fixed occupancy level the load is determined almost solely by transmission, solar loads being negligible for this case. Figure 5.11 shows for three different types of system the effect of falling load on room relative humidity for a maximum occupancy level of 150 students. The conventional systems behave as expected, relative humidity rising to particularly high levels at the lower end of the operating range of the conventional CAV

---

31 Signalled in practice by the water pressure drop across the coil, or some other surrogate.

32 This example, which represents an embryonic HDP system (see section 5.2.2), will be analyzed in further detail in chapter 13. It is introduced here to illustrate the effect of staging on the return air coil.

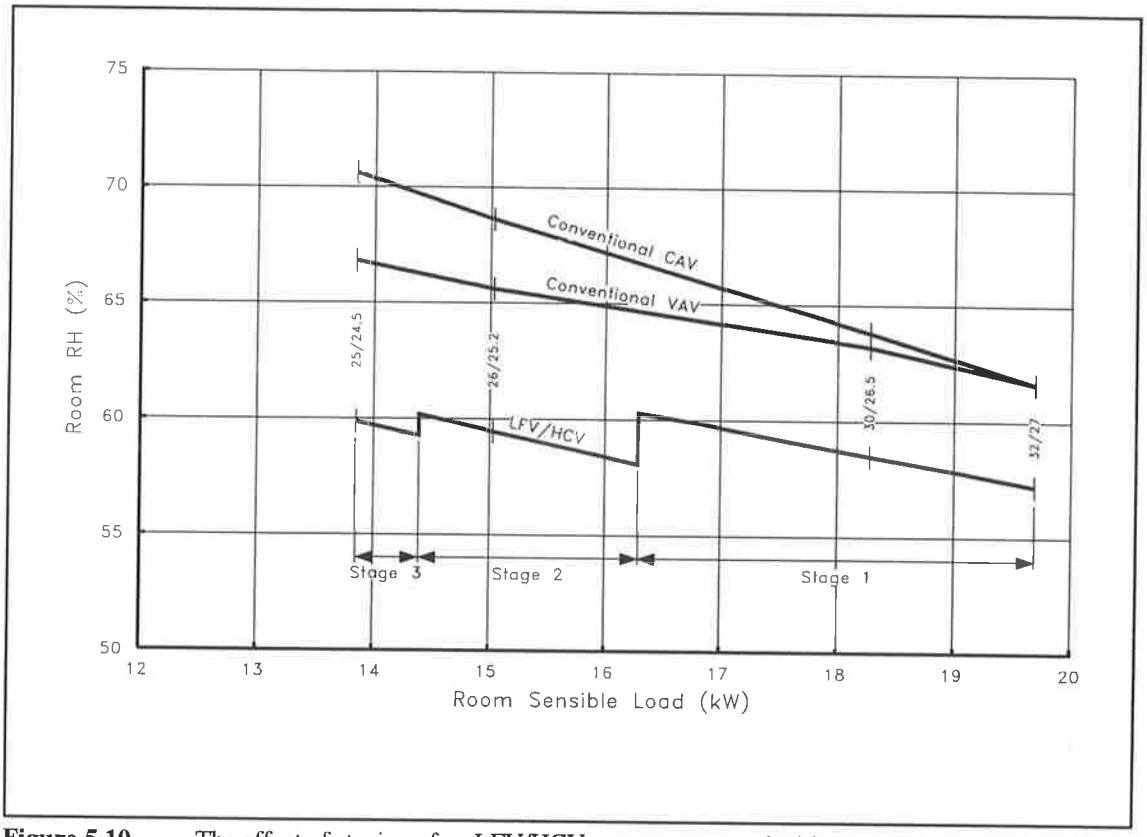


Figure 5.10. The effect of staging of an LFV/HCV system compared with conventional VAV and CAV systems. A lecture theatre in the tropics with a constant population of 150 students.

system. The LFV/HCV VAV system uses two coils, one handling the return air stream, and the other handling the outside air. Chilled water is fed to the outside air coil first, and then to a return air coil, water supply being controlled by a throttling valve. Staging is applied to the return air coil only, which is a three-row coil having a total of 18 circuits. On the first stage change, four circuits from the two downstream rows of the coil are deactivated using a simple on-off valve as the water velocity in the tubes falls below 0·8 m/s. In the third stage, a further two circuits are similarly deactivated. The resulting behaviour is shown in figure 5.10. Within a stage, dehumidification performance is as would be expected for a conventional VAV system. By introducing the two step changes in the active surface area, the relative humidity can be kept below 60% with the LFV/HCV system over most of the operating range. Because the dewpoint of the room air is kept low by the LFV/HCV system, a lower supply air temperature can be used throughout without fear of condensation forming on the supply air diffuser.

Note that this is a particularly demanding application. In most cases only one stage change is required.

5.2.2. HDP Systems.

The same considerations which led to the development of the LFV/HCV system of air conditioning have also been instrumental in formulating an alternative, and, as seen in the previous subsection, complementary methodology. This is known as the High Driving

Potential (HDP) method of air conditioning. At first glance the HDP methodology would seem to be a simple variation of the conventional precooling scheme, which is widely applied in tropical climates to make the humid outdoor air look like temperate climate air. However, the purpose of the two schemes is fundamentally different.

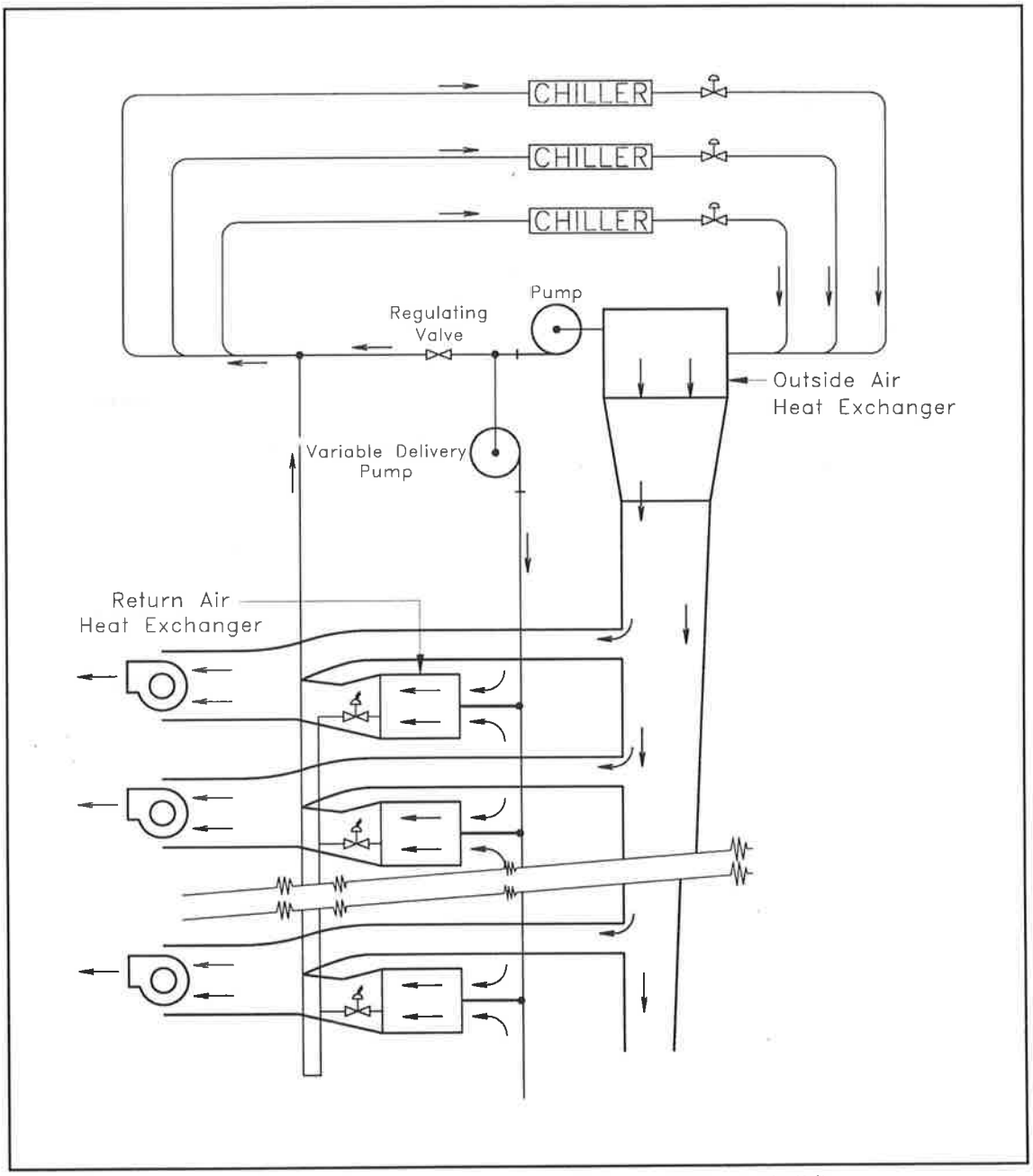


Figure 5.11. Schematic layout of an HDP system applied to a multistorey building.

In a conventional system the purpose of precooling is mainly to reduce the dry bulb temperature of the outdoor air, although invariably some dehumidification will also occur. The outdoor air is typically precooled to room temperature or a little lower, dry bulb temperatures within the range of 18-24°C being usual. The pretreatment plant is characterized by a modest chilled water flow rate, which results in a chilled water

temperature rise through the dehumidifier coil of about 8°C to satisfy the entry requirements of the chiller. The pretreatment is usually performed in one or more central units before distribution to the various air handling units located throughout the building. The outdoor air is mixed with the return air from the zones served by the unit, the combined airstream then being passed through a common dehumidifier coil. In applying only moderate precooling to the outdoor air, little use is made of the potential for dehumidification which exists between the outdoor air and the cold coil surface. As a consequence, precooling in the conventional sense has little to offer by way of improved dehumidification performance.

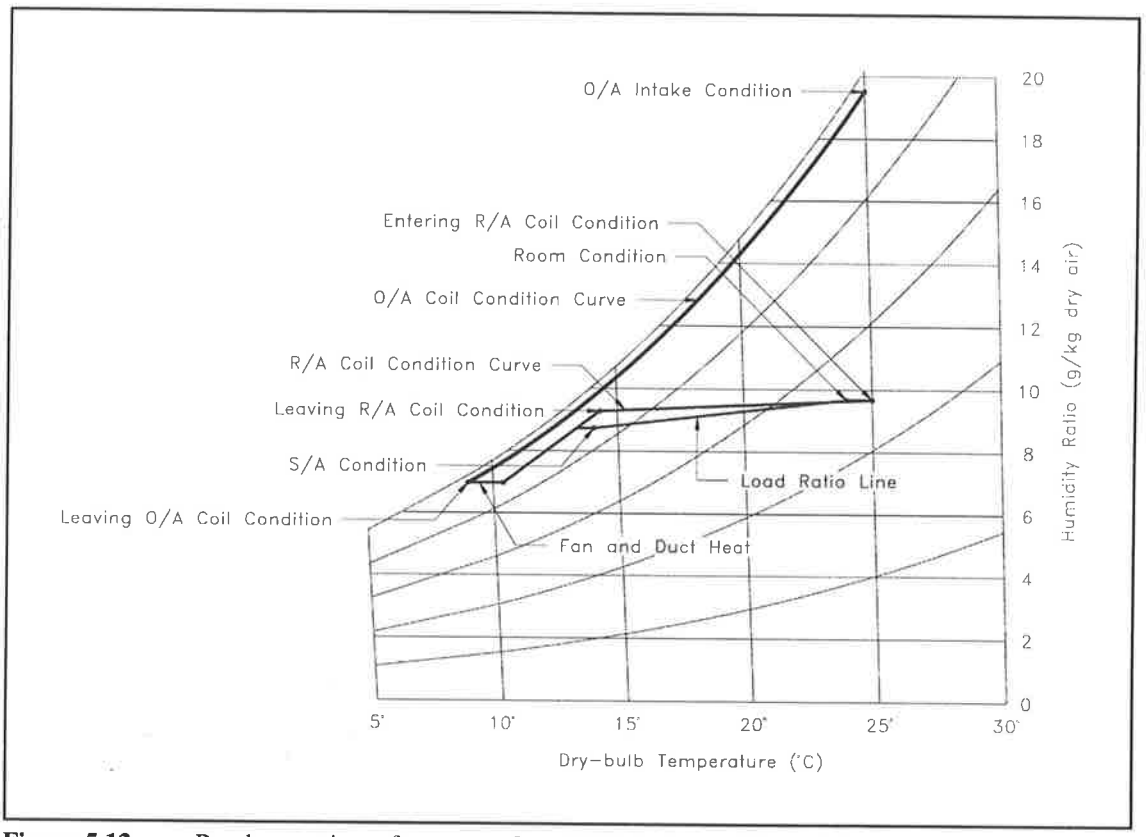


Figure 5.12. Psychrometric performance of an HDP system operating under part-load conditions.

In an HDP system, as with the pretreatment system, it is preferable, but not essential, to treat the outdoor air in a central plant before distributing it to the various terminal units. It is also possible to perform the treatment of the outdoor and return air streams within a common casing. This scheme will be described later in the discussion; for the moment we will assume that a central outdoor air treatment plant is in use. The strategy employed by the HDP system differs fundamentally from that of a conventional precooling scheme in that within the HDP system the outdoor air is cooled to a dry bulb temperature which is lower than the target supply air temperature for the terminal unit. The coil off temperature is typically within the range of 8-13°C. In the process of cooling to temperatures of this order, the outdoor air experiences considerable dehumidification. Within the terminal units this cold dry outdoor air is mixed with partially *treated* return air, and in the mixing process excess moisture and some excess sensible heat is transferred from the partially treated return air to the cold, dry, over-treated outdoor air. By this means the relatively

small quantity of outdoor (ventilation) air can offset a considerable part of both the latent and sensible zone loads, thus reducing the size of the terminal unit required to meet a specified duty.

The chilled water and air paths for a typical HDP system servicing a multistorey building are shown schematically in figure 5.11. The underlying psychrometric principles of such a system, when operating in a tropical climate, are illustrated in figure 5.12. The process lines shown are appropriate to a system operating at 50% of peak sensible heat load. The path of the outdoor coil condition curve effectively follows the saturation line, which is the steepest thermodynamically possible path. At the critical part-load condition, the dew point temperature of the air leaving the coil is 8·9°C, and the return air coil condition curve is very shallow, operating almost dry. The outdoor air coil removes 10·4 g moisture per kg of dry air, as opposed to a mere 0·75 g/kg in the case of the return air coil. Mixing the over-pretreated outdoor air with the under-treated return air produces supply air having a lower dew point temperature than is otherwise attainable using conventional technology. The deep dehumidification offered by the outdoor air heat exchanger thus offers an efficient replacement for techniques such as overcooling and reheating where the maintenance of indoor air humidities within the ASHRAE comfort zone is required during periods of high dew point ambient conditions.

To attain the deep dehumidification of the outdoor air, which is a central feature of the HDP methodology, it is necessary to maintain throughout the depth of the coil a relatively high differential between the partial pressure of the air entering the outdoor air dehumidifier coil and that of the water vapour at the wetted external surfaces of the coil. This is achieved by minimizing the chilled water temperature rise through the dehumidifier and maximizing the chilled water velocity. Typically the dehumidifier would be designed for a water temperature rise of only about 2°C. This is achieved by passing through the dehumidifier the whole quantity of water required to offset the simultaneous requirement of the terminal units. Furthermore, the chilled water path through the outdoor coil is kept short to minimize both the water pressure drop and the water temperature rise through the coil.

Herein lies an additional difference between the HDP system and a conventional precooling system. The temperature of the water leaving a conventional precooling unit will be so high that its potential for further cooling or dehumidification will effectively be exhausted. Chilled water must therefore be supplied to the outdoor air plant and to the terminal air handling units via separate parallel circuits. In the HDP system on the other hand, the water temperature rise within the outdoor air treatment plant is kept sufficiently small for the water leaving this unit to be supplied to the terminal units cooling the return air, the latent load of which will subsequently be substantially offset by the dry, cold treated outdoor air.

Consider now the behaviour of an HDP system as the load falls. Two limiting cases are possible. On the one hand the water flow through the outdoor air coil may reduce, always being equal to the sum of the flows required by the terminal units. In this situation the dehumidification potential of the outdoor air coil reduces with falling load. In demanding applications it may be necessary to employ staging within the outdoor air treatment plant

or in the terminal units, or both, if design specifications are to be met. Alternatively, if a chilled water bypass is installed, as shown in figure 5.11, the chilled water flow rate through the outdoor air coil can be maintained at the same high level, while the supply of chilled water to the terminal units is throttled. Falling zone loads are frequently associated with a fall in the enthalpy of the outdoor air, and when this coincides with constant or increasing zone latent loads, a particularly difficult situation may arise. If the chilled water flow through the outdoor air coil is maintained at a constant level, falling loads will be associated with a decrease in the dry bulb temperature of the treated outdoor air. In other words, the dehumidification potential of the treated outdoor air increases just when the extra dehumidification is required. As the load decreases, the latent load on the return air coil rapidly reduces to negligible levels. This is reflected in the behaviour of the room relative humidity, which shows an initial increase as the load reduces from peak and then, contrary to conventional experience, a continuous decrease with further reductions in the zone load as the effect of the treated outdoor air becomes dominant. This effect applies to both VAV and CAV terminal units. The choice of whether to specify a VAV or a CAV system then becomes a trade-off between capital costs and fan operating costs, rather than a decision based primarily on psychrometric considerations. Since under part-load conditions the coil in the terminal units is mainly concerned with offsetting *sensible* load, enhancement of dehumidification achieved by LFV/HCV technology in the terminal units confers little benefit. Low face velocities and low fin densities are still of value in minimizing fan power consumption and noise. It is desirable to maintain the Reynolds number of the water flow in the turbulent regime for control reasons, and this factor may indicate a need for staging. Clearly, the option of maintaining water flow through the outdoor air coil at a constant high level is a preferred characteristic.

Use of HDP technology confers additional benefits. Good ventilation practice and adherence to ventilation standards need no longer be seen as a massive energy penalty. One of the major problems of existing air conditioning systems is their common failure to satisfy the ventilation requirements during part-load operation. This problem is particularly prevalent in the case of variable air volume systems which serve a number of zones. VAV systems are otherwise preferred to CAV systems for reasons which have been enumerated above. The problem is particularly serious during part-load conditions for a building which has a constant population density, especially in the least loaded zones, in a library for example. A building that requires 15% outdoor air at peak sensible load would require 30% outdoor air when the sensible load falls to 50% of its peak value. If, as is common in the "sick" buildings of the past 25 years, the outdoor air dampers are fixed so that the ratio of ventilation to return air remains approximately constant during VAV operation, there is an inevitable tendency for stuffiness, especially in the least loaded zones. Thus the tendency of people to doze in the library may be related to inadequate ventilation. Such conditions are not likely to arise in connection with the new HDP method, as there is benefit in maintaining the quantity of treated outdoor air. This air contributes significantly to satisfying the room load over the full operating range. Plant rooms can then be smaller or even eliminated, as return air treatment units have very modest duties. Fan coil units can be wall mounted or, subject to maintenence constraints, could be accommodated in the ceiling plenum.

Use of the HDP principle in conjunction with a central outdoor air treatment plant confers major benefits resulting from the reduced size of the terminal air-handling units, reduced alienation of rentable floor space, and a significant downsizing of chilled water risers and pumps. The significant reduction in chilled water pumping energy at part-load confers benefits in terms of operating cost. This will be partially offset by the cost of moving the pretreated air around the building, although the quantities involved are modest by comparison with conventional central plant systems. Note that no return air riser is required and spill can be effected via washrooms and toilets at each level.

Occasions will of course arise when technical or contractual considerations preclude the use of a central outdoor air treatment plant. In such circumstances the benefits listed in the preceding paragraph are not wholly attainable. It is however still possible to take advantage of the unique psychrometric performance of HDP systems by mounting separate outdoor air and return air coils in a common casing. The chilled water supplied to the air-handling unit passes first through the outdoor air coil, and then through the return air coil. The treated airstreams are mixed before entering the fan. At part-load the water supply to the unit may be throttled, or alternatively the flow through the outdoor air coil may be held constant at its peak value, while water in excess of that required to offset the zone load is bypassed around the return air coil. The psychrometric performance of such units is then almost identical to that pertaining to an HDP system with a central outdoor air treatment plant.

Chapter 6. Simulation of Air Conditioning Systems.

The preceding chapter has briefly reviewed the characteristics of a range of all-air system types which may be used to achieve a desired set of thermal comfort conditions within one or more zones. A major objective of the current work is to develop the appropriate software tools to enable the designer to trial a series of potential design strategies for a particular problem. The purpose of the present chapter is to introduce and describe a set of basic abstractions and algorithms which will simulate hardware items from which the previously-reviewed systems are constructed, together with the thermodynamic processes comprising the air conditioning cycle.

The initial focus of the chapter is on the zone. For a given project the zone comprises a conditioned space, which has associated with it a target set of thermal environmental specifications. The efficiency with which competing systems will maintain the specified conditions under the influence of changing occupancy patterns, outdoor air conditions and solar loads is thus a major yardstick by which systems may be compared (first cost being the other major yardstick). The emphasis here is on developing a self-contained set of data structures which will adequately describe the zone and the psychrometric processes occurring therein, together with the data structures required to measure and control zone conditions.

In the sections which follow, data structures are developed to model a range of conventional and novel air conditioning systems. The structures described offer the designer considerable flexibility in the configuration and evaluation of alternative air conditioning strategies for a given situation. Finally, a set of algorithms to model the steady-state performance of an air conditioning system are presented.

6.1. The Zone.

A *zone* is a conditioned space in a building. According to ASHRAE (1987), the term "*zone* implies the provision or the need for separate thermostatic control, while the term *room* implies a partitioned area that may or may not require separate control." The two terms are used interchangeably in the present work, with some preference being shown for the former. The requirement that zones are subject to thermostatic control is relaxed; in respect of CAV operation, the situation whereby some (but not all) zones served by a unit are not so controlled is admissible. Thermal conditions within the zone must, however, be measurable, at least from a conceptual point of view.

The term *zone* is also used in a different, although related sense in the present work, in that it may be used to refer to the abstract entity used to model the conditioned space. This is implemented as the C++ class ***Zone***, which maintains internal variables describing the environmental state of the space, together with the sensors required to measure and control conditions in the space, and member functions to set and access information relating to the zone, and to perform computations associated with processes occurring within the zone. Objects of class ***Zone*** are self-contained in respect of their interaction with the outside

world. From the point of view of the *Zone*, external objects may alter sensor setpoints, the flow rate and the condition of the air supplied to the zone. From the viewpoint of coordinating objects, the zone is a 'black box' which will react in a certain way in response to the supplied air stream, the response being measurable by the deviations of sensor readings from their setpoints.

Zone loads may be specified, and are stored internally in a structure of class *Load*, which contains fields for the sensible, latent and total components of the load, and for the sensible heat ratio, defined as

$$SHR = \frac{q_s}{q_s + q_l} \quad . \tag{6.1.1}$$

Zone loads are normally calculated externally using a load calculation programme, such as TEMPER (Australian Construction Services, 1987). Class *Zone* does not currently maintain the thermal data required to enable zone loads to be calculated internally, although such information could readily be incorporated into the existing framework.

Environmental conditions within the zone are determined by the state of the air supplied to and extracted from the zone, denoted by the suffixes *i* and *o* respectively in the following. The latter quantity is regarded as representative of thermal comfort conditions within the zone, and is monitored to provide the basis for control strategies concerning the zone[33]. In the basic model considered herein, questions of air distribution within the zone are not addressed.

The flow rate of air through the zone may be expressed in terms of either mass flow rate or volume flow rate. The latter quantity is ambiguous, as it must be referenced to the appropriate density, volume flow rate so defined being referred to as the *actual* volume flow rate. Clearly, the actual volume flow rate of air leaving a zone will differ from that of the air entering the zone. In an effort to overcome this ambiguity a third quantity, the *standard* volume flow rate, is commonly used within the industry. This is defined as the volume flow rate referenced to ASHRAE standard conditions, which specify a density of 1·204 kg/m³, corresponding approximately to saturated air at 16°C, or to dry air at 21°C, at standard atmospheric pressure (101·325 kPa). Since each volume flow rate may readily be converted to mass flow rate, which is an unambiguous quantity, a simple formula is available for converting between the two volume quantities:

$$Q_{a,std} = \frac{Q_{a,act}\rho_{act}}{1{\cdot}204} \quad . \tag{6.1.2}$$

Internally, calculations are performed preferentially in terms of mass flow rate or where necessary, as in the case of fan calculations, in terms of actual volume flow rate. The use of standard flow rate is nevertheless widespread in the industry, and in most cases results are reported in terms of this quantity. For CAV operation, the rate at which air is supplied

---

[33] See also section 3.1.

to the zone remains fixed. The appropriate standard volume flow rate[34] is stored as a member variable of class *Zone*, which also provides member functions to set and access this quantity.

In order to control infiltration and exfiltration of air to and from the zone, it is customary to maintain a small differential (usually positive) between the operating pressure of the zone, and ambient pressure. The magnitude of this pressure differential becomes important when calculating duct flow rates, pressure losses and damper settings. Consequently, the required pressure differential is stored as a member variable of class *Zone*[35].

Class *Zone* provides a full complement of member functions for reading and setting its component variables and structures. Also provided are member functions for performing calculations referring to psychrometric processes occurring within the zone, and structures for measuring and monitoring conditions within the zone. These latter aspects will be considered in the following.

6.1.1. Zone Processes.

The psychrometric processes affecting a parcel of air during its passage through the zone are customarily represented on the psychrometric chart by a straight line, known as the *load ratio line*, connecting a point representing the entering air state with one representing the leaving air state. Such a line is shown in figure 6.1. In analyzing the thermodynamics of the situation shown, it is useful to consider a point lying at the intersection of the line of constant humidity ratio passing through the entering air condition, and the line of constant dry bulb temperature passing through the leaving air condition. In terms of the psychrometric points shown in figure 6.1, the latent and sensible components of the load,

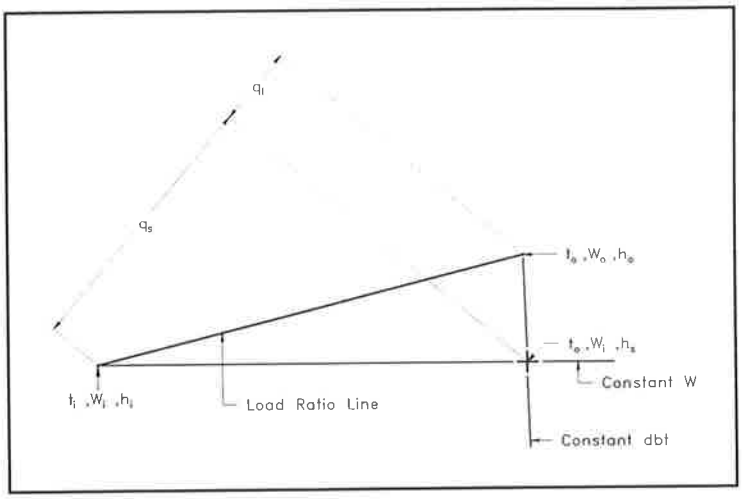


Figure 6.1. The load ratio line.

$$q_l = \dot{m}(h_o - h_s) \quad , \tag{6.1.3}$$

---

34 In fact it is the *actual* volume flow rate which remains fixed during CAV operation. The need to reference this quantity to a changing air state during the simulation process would however result in operational difficulties, and *standard* flow rate is used instead. The resulting errors will be minimal.

35 By default, this is set to zero when an object of class *Zone* is created.

$$q_s = \dot{m}(h_s - h_i) \qquad \textbf{(6.1.4)}$$

respectively, where the latent load may be expressed in terms of the sensible component by rearranging equation (6.1.1) so that

$$q_l = q_s\left[\frac{1}{RSHR} - 1\right] \qquad \textbf{(6.1.5)}$$

where now RSHR is the room sensible heat ratio.

The enthalpy values in equations (6.1.3) and (6.1.4) are a function of the corresponding dry bulb temperature and humidity ratio. Substituting the functional relationship of equation (4.1.25), the above equations become

$$q_l = \dot{m}(W_o - W_i)(2501{\cdot}8 - 1{\cdot}8144t_o) \quad , \qquad \textbf{(6.1.6)}$$

$$q_s = \dot{m}(1{\cdot}006 + 1{\cdot}8144W_i)(t_o - t_i) \quad . \qquad \textbf{(6.1.7)}$$

Combining these gives

$$q_s = \dot{m}\left[1{\cdot}006 + 1{\cdot}8144\left(W_o - \frac{q_l}{\dot{m}(2501{\cdot}8 + 1{\cdot}8144t_o)}\right)\right](t_o - t_i) \quad . \qquad \textbf{(6.1.8)}$$

Invoking equation (6.1.5), this may be rearranged to solve two problems of significant practical importance:

1. In a variable air volume (VAV) system the dry bulb temperature of the supply air and the dry bulb temperature of the air leaving the room are maintained at constant values by a thermostat downstream of the fan in the first instance, and by a thermostat in the conditioned space in the second instance. Before entering the zone, the supply air will be subjected to a small temperature rise as a result of heat transferred through the duct walls. Provided this temperature rise can be calculated, the temperature rise through the zone, $\Delta t = t_o - t_i$, will be known. The mass flow rate of air required to offset the zone loads subject to temperature rise Δt will then be

$$\dot{m} = \frac{q_s}{1{\cdot}006 + 1{\cdot}8144W_o}\left[\frac{1}{\Delta t} + \frac{1 - RSHR}{RSHR(2501{\cdot}8 + 1{\cdot}8144T_o)}\right] \quad . \qquad \textbf{(6.1.9)}$$

2. For a constant air volume (CAV) system, the zone dry bulb temperature and the mass flow rate of air remain constant; Δt changes to accommodate varying loads. In this case,

$$\Delta t = \left[\frac{(1{\cdot}006 + 1{\cdot}8144W_o)\dot{m}}{q_s} - \frac{1 - RSHR}{RSHR(2501{\cdot}8 + 1{\cdot}8144t_o)}\right]^{-1} \quad . \textbf{(6.1.10)}$$

In the above, the supply air humidity ratio W_i can be substituted for the room humidity ratio W_o, where it occurs, by invoking equation (6.1.6). The form given is however the more useful for the purposes of the iterative schemes to be developed later in the chapter.

Two further operations involving the load ratio line, which will be performed extensively in the simulation of air conditioning cycles are the following:

1. The state of the air entering the zone is specified. The state of the air leaving the zone can be found by setting the leaving condition equal to the entering condition, and:

   a. Incrementing the enthalpy by the amount $\Delta h_1 = q_s/\dot{m}$ while keeping the humidity ratio constant, and finding the corresponding dry bulb temperature.
   b. Incrementing the enthalpy by the amount $\Delta h_2 = q_l/\dot{m}$ while keeping the dry bulb temperature constant, and finding the corresponding humidity ratio.

2. The state of the air leaving the zone is specified. The state of the air entering the zone can be found by setting the entering condition equal to the leaving condition, and decrementing the enthalpy in stages in the inverse of the order above.

Member functions of class *AirState* to perform the above operations are described in section 4.3.3.

6.1.2. Sensors.

The preceding discussion identifies three properties of the zone which require monitoring and/or control, namely the dry bulb temperature, the relative humidity, and the pressure. This section describes classes to monitor and control the former two properties. The desired pressure differential is simply specified; the duct simulation algorithms of chapter 9 assume that this differential will always be maintained.

Dry bulb temperature and relative humidity are measured by objects of class *Thermostat* and class *Humidistat* respectively, both of which are derived from a common base class *Sensor*. In its present form this class is quite rudimentary. A setpoint and a deadband may be specified within a constrained range imposed by the derived classes. Additional fields contain the current value of the measurand, and a flag indicating whether the sensor is enabled or not. The purpose of this flag, which may be set, is simply to indicate to objects accessing the sensor whether the measurand is to be controlled or not; the monitoring function of the sensor is always available. Member functions provide facilities for setting and reading the setpoint and deadband and the status of the enabled/disabled flag, for updating and reading the measurand, and for returning the deviation of the measurand from the setpoint and its square, and the magnitude of the extent to which the measurand exceeds the upper limit or falls below the lower limit of the deadband.

As it stands, this class is adequate for the steady-state models considered herein. With the addition of fields specifying the dynamic characteristics, the class could readily be adapted for use in transient models.

Objects of class *Thermostat* and class *Humidistat* are passed on creation a pointer to the object of class *AirState* which they are to monitor. By default, objects of class *Thermostat* are *enabled* upon creation; the default state for objects of class *Humidistat* is *disabled*. Class *Thermostat* adds little further to the base class, apart from modifying the vocabulary of the member functions. Objects of class *Humidistat* are primarily concerned with detecting relative humidity levels which fall outside a given range, which is specified internally in terms of the deadband. In this work we are concerned solely with the upper bound of the humidistat range which, for an *enabled* humidistat, must lie between 45% and 65% relative humidity.

6.2. Data Structures for System Simulation.

Computer simulation provides a means of predicting the performance of an air conditioning system. Chapter 5 has defined the characteristics of the systems which we seek to model; the model must possess sufficient flexibility and generality to simulate the range of configurations and control strategies embodied in those systems. In accordance with the principles of the object-oriented design methodology, such a model may be developed by selecting a suitable set of coordinating classes, and defining the operations whereby they interact. The core class categories required to implement the model are shown in figure 6.2. In the remaining sections of this chapter we seek to develop the classes associated with the class categories of figure 6.2. These class categories are not self-sufficient. In order to complete the implementation the classes described in the following will need to access a number of additional classes representing various equipment items and subsystems. These will be identified in context, and where such classes are sufficiently simple, they will be described in the following. Three such categories of classes require separate detailed treatment however, namely those describing cooling coils, duct systems and fans. They will be the subject of the following four chapters. In this study the emphasis will be placed on developing a detailed model of the *air distribution* system. A set of class categories paralleling those of figure 6.2, and describing components such as chillers and pipe networks would be required to model the chilled water distribution system. Such a development will not be undertaken here. However, where appropriate the classes representing various equipment items will be designed in such a manner as to facilitate

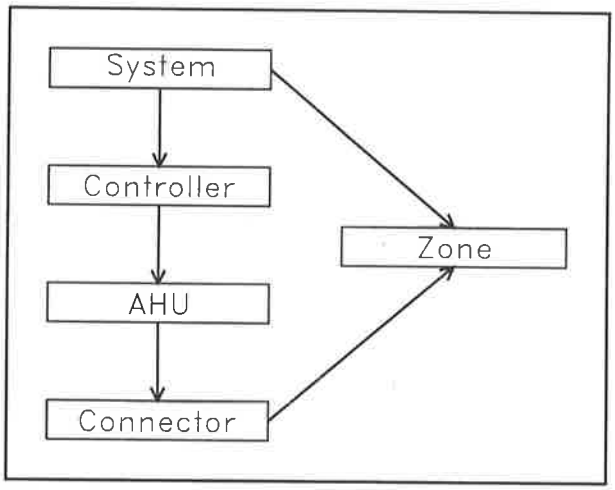


Figure 6.2. Core class categories required to implement the air conditioning system model.

their use within a model of the chilled water distribution system. Thus for instance, in the present model duct systems are represented by a class representing a specialization of a base class for generalized networks, which could also be used as a base class for pipe networks.

In essence, the class categories of figure 6.2 perform the following functions:

i. The ***system*** is the user interface to the model. All requests to update and run components of the model are processed by this class category, which also provides a central conduit for querying the status of the various system components.

ii. Objects belonging to classes in the category ***AHU*** provide a description of the *physical* attributes of each air handling unit in the system, including in particular their configuration.

iii. Each object of class category ***AHU*** has associated with it an object belonging to the class category ***controller***. Such objects are concerned with the control operations relevant to the air handling unit with which they are associated, and as such provide an interface whereby the ***system***, and ultimately the user, can access each air handling unit.

iv. Each air passageway into or out of an air handling unit will have associated with it an object belonging to class category ***connector***. Each such object in turn will have associated with it a duct system, which may or may not be specified in detail, and optionally a cooling coil and a filter.

v. Class category ***zone*** contains one class, *Zone*. This has been described in detail in section 6.1.

6.2.1. The System.

The system defines the simulation domain of the model. Conceptually the system refers to a set of zones, together with the air handling units required to service those zones, and their associated duct and pipework systems. User access to the system is provided by a C++ class *SystemControl*, which has the following attributes:

i. There can be at most one instance of class *SystemControl*, referenced through the **static** member pointer `SystemControl::s`, which is initially set as a null pointer. If the constructor for *SystemControl* is called, and a non-null value for `s` is detected, an exception will be generated. Otherwise, `s` will be set to point to the newly generated instance of *SystemControl*.

ii. The following properties of the system are stored as member variables:
   a. Atmospheric pressure.
   b. Ambient air state (as a variable of class *AirState*).
   c. Time.

   Member functions to set and read these variables are also provided.

iii. Associative arrays provide access to all major components of the system, including in particular:
   a. Air handling unit controllers.
   b. Duct systems.
   c. Zones.

iv. The constructor for class *SystemControl* may take as an input parameter a reference to an input stream (see Stroustrup, 1991), from which the specifications may be read. This initiates a sequence in which the constructors for the various component items are called as specified to set up the system. The basic data structures used to read the system specifications from an input stream are described in section 2.4. The process is considered further in chapter 12.

v. A command-line interpreter enables commands to be read and executed. The commands provide a means to alter system parameters, and to run various aspects of the simulation. This facility is also described further in chapter 12.

6.2.2. Air Handling Units and Controllers.

Two basic types of air handling unit are considered in this study:

i. An *Outdoor Air* ahu takes in outdoor air at ambient conditions, and distributes it to one or more distributed units, after cooling in a prescribed manner. The central outdoor air pretreatment plants for conventional precooling systems and the outdoor air treatment plants for HDP systems fall into this category.

ii. A *Distributed* ahu takes in outdoor air and (optionally) return air, and distributes the mixture, cooled to a supply air temperature determined by the system control strategy, to one or more zones. The outdoor air may be either at ambient condition, or as supplied from an outdoor air treatment unit. The ahu may also contain facilities for treating the outdoor air and return air streams individually, or for combining the two streams, and then treating them. The definition of this type of ahu is thus sufficiently broad to cover both stand-alone multizone units, and the terminal units of an HDP system.

To model these a base class *AHU* is defined. From this are derived classes *D_AHU* and *OA_AHU*, which provide facilities to store, set and read the configuration and operating parameters for distributed and outdoor air air-handling units respectively. Within the context of the system model, each object of class *AHU* (or a derived class) has associated with it one and only one controller of the appropriate type[36]. Controllers are instances of class *DController* (controlling a distributed ahu), or of class *OController* (controlling an

---

[36] As implemented each object of a class derived from base class *Controller* in fact contains a variable of the appropriate subclass of *AHU* as a member variable.

outdoor air ahu), both of which are derived from a base class *Controller*. The relationships between the various subclasses of class *Controller* and those of class *AHU* are as shown in figure 6.3. The purpose of base class *Controller* is merely to store and manipulate basic bookkeeping information associated with the controller. In keeping with the remarks made earlier, the derived classes provide the primary means of accessing the objects representing the air handling units, together with associated entities. The classes derived from base class *Controller* also contain member functions to perform simulation functions for the subsystem associated with each controller. The algorithms for performing these operations are the subject of section 6.3. Further discussion of the structure of class *Controller* and its subclasses will be deferred to that section.

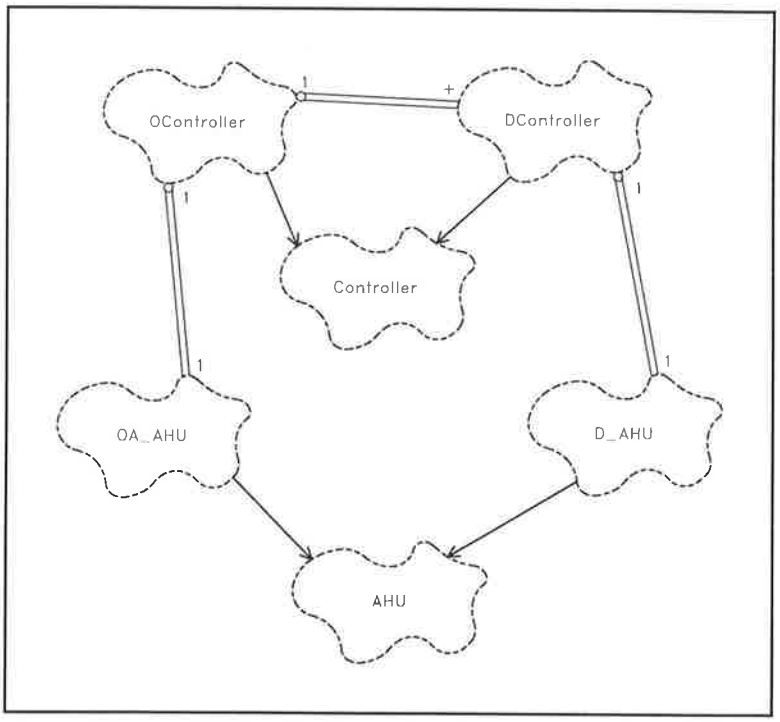


Figure 6.3. Class diagram showing relationships between subclasses of class *AHU* and class *Controller*.

Stripped to its bare essentials, an air handling unit comprises a fan, controls and a number of connectors to air intake or output subsystems, each of which will optionally have associated with it a ductwork system, and a cooling coil and filter. Member variables within base class *AHU* store the air state upstream and downstream of the fan, and the mass flow of air through the fan. Other member variables of importance are the following:

`f` : An object of class *FanUnit* representing the fan and its associated structures. The interaction of this class with its coordinating classes will be considered in chapter 10.

`fp` : A variable of the enumerated type

```
enum fpos { Unspecified, DrawThrough, BlowThrough };
```

Variable `fp` is initialized to a value of `Unspecified`. The fan position will subsequently be defined by the configuration of the connectors which are attached to the unit. These are in fact fundamental in determining the configuration of and control strategies available for a unit. The rules for combining the connectors for a unit are thus of considerable interest. These will be considered in the next section.

`connected` : A boolean variable which specifies whether a sufficient set of connectors have been attached to an ahu. This variable is initialized to *false*.

6.2.3. Connectors.

The connectors to an air handling unit are represented by objects belonging to subclasses of a base class *Connector*. The following variables are members of class *Connector*[37]:

i. A pointer to a duct system (initialized to null).
ii. A pointer to a filter (initialized to null)[38].
iii. A pointer to a coil bank (initialized to null).
iv. Pointers to the air state entering and leaving the connector (variables of class *AirState*).
v. Mass flow of air through the connector.

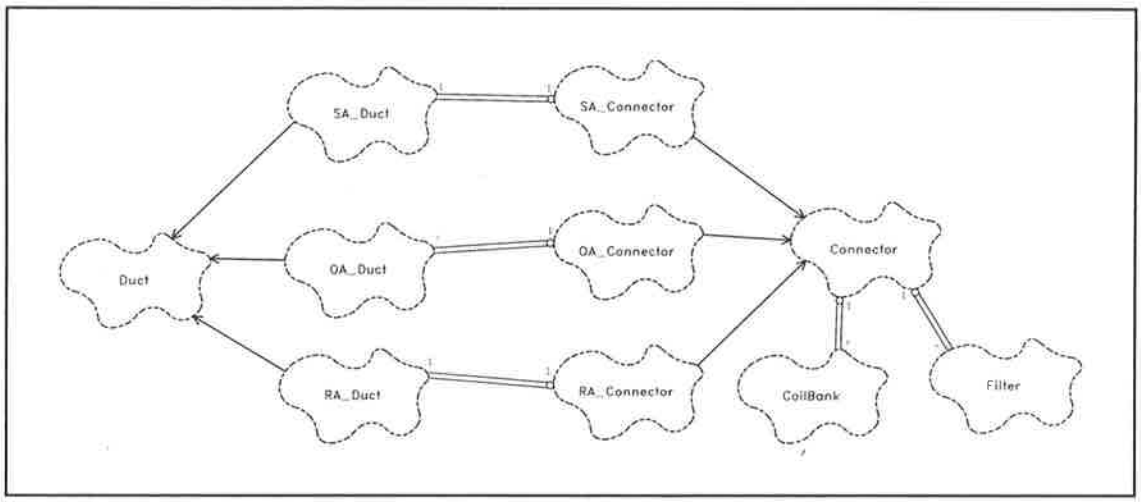


Figure 6.4. Subclasses of class *Connector* and their relationship to other classes.

A connector which has been *attached* to a unit is not *modifiable*; pointers (i) to (iii) above cannot be altered in this state. If it is desired to alter any of these pointers, the connector must first be *detached*. This enables the software to identify correctly the configuration of the unit at all times. In addition to the above, class *Connector* contains additional variables for internal use only, together with a set of functions to set and reset and read its attributes. In the context of the air handling unit model it is the subclasses of class *Connector* rather than the base class which are of interest. These differ from the base class mainly in the manner in which they deal with the duct system. A duct system of the appropriate type (a subclass of class *Duct*) must be attached to the connector, except in certain cases specified below. The duct system need not be fully specified (see chapter 9). The relationships connecting the various classes involved are shown in figure 6.4. The subclasses of class *Connector* concerned are:

---

[37] This inventory will be expanded in section 6.5, which considers the generalization of chilled water circuiting configurations.

[38] An object of class *Filter* (see Chapter 10).

i. Class *SA_Connector*. A supply-air connector connects to the root of a duct system distributing treated air to a set of zones in the case of a distributed ahu, or to the root of a duct system distributing treated air to a set of distributed air handling units in the case of an outdoor air ahu. This subclass provides the following attributes which are additional to those of the base class:

   a. A thermostat (an object of class *Thermostat*; see section 6.1.2), which is used to monitor (and in the case of VAV operation, control) the supply air temperature.

   b. A singly-linked list of indices to the zones or distributed air handling units (as appropriate), which are fed by the connector.

   c. A reheat coil (an object of class *ReheatCoil*; see section 6.3.1) which is used to control humidity within the zones served by the unit by applying reheat on a *global* basis to the supply air. By default, the reheat coil is *off*.

ii. Class *OA_Connector*. In the case of an outdoor-air treatment plant, or a distributed unit drawing air from outside the building, the connector is attached to the root of an outdoor-air duct system (see chapter 9). In the case of a terminal unit the connector is supplied with air from one of the terminal nodes of the supply-air distribution duct system attached to the supply-air connector of a central outdoor-air treatment plant. In this case, the entering air condition for the outdoor-air connector of the terminal unit may be set, but this connector may not access the duct system which serves it.

iii. Class *RA_Connector*. A return-air connector attaches to the root of a return air duct system, and only has significance in the case of distributed air handling units.

The characteristics of an air-handling unit are determined by the configuration of the connectors which are attached to it. The range of supported configurations is considered separately in the following for the case of distributed and outdoor-air units.

6.2.3.1. Distributed ahu.

The range of supported cooling coil configurations for a distributed ahu is as shown in figure 6.5[39]. Two basic divisions are evident; configurations (a), (b) and (c) are blow-through configurations, while the remainder are draw-through configurations. In the model supported, a blow-through unit will have one or more supply-air connectors attached to it, while a draw-through unit will have one and only one supply-air connector. This will be indicated by the setting of the flag `fp` in the base class *AHU* (section 6.2.2). In addition, to distinguish between the various configurations shown, the derived class *D_AHU* defines the following member variables:

---

[39] Frequent reference is made to figure 6.5 in the following. For convenience therefore, this figure is reproduced on a lift-out card which will be found inside the back cover.

`oas` : Indicates the source of the outdoor air fed to the unit. This is a variable of the enumerated type

```
enum oa_source { Undefined, O_A, D_A };
```

This variable is initialized as `Undefined`. `O_A` indicates that the outdoor air is ingested directly from the exterior of the building; `D_A` that it is supplied from a central OA treatment plant.

`combined` : A boolean variable indicating that the outdoor air and return air streams are combined before treatment using a single coil. Configurations (a) and (b) and (e) and (f) are `combined`; the remainder are not.

Two further member variables are defined to specify the chilled water circuiting for configuration (g):

`vfirst` : A boolean variable indicating whether the chilled water is to be fed to the outdoor air coil first (*true*), or otherwise (*false*). Default *true*.

`qfs` : A variable of class ***CfixedFlow***, which has two member variables:

`Qfixed` : A boolean variable, the value of which has the following significance:
false : The total flow rate through the ahu is modulated; the cooling coil(s) draw the chilled water flow rate they require to offset the load.
true : The total flow rate through the ahu is fixed.
The operation of these member variables is modified by local control actions within the outdoor air and/or return air connectors (see section 6.5.3 for details). The default is *false*.

`Qf` : Fixed water flow rate (L/s).

We can now write a set of rules for attaching connectors to and detaching them from a distributed ahu:

i. A distributed ahu is `connected` if it has the following connectors attached to it:
   a. One and only one outdoor air connector.
   b. A maximum of one return air connector.
   c. At least one supply air connector.

ii. A distributed ahu can have an arbitrary positive number of supply air connectors attached to it. A pointer to a supply air connector is stored on a singly-linked list (a member variable of class ***D_AHU***) when the connector is attached, and removed when it is detached. The following constraints apply when attaching or detaching supply air connectors:
   a. A supply air coil which references a coil bank cannot be attached to a unit designated as `DrawThrough`. If such a connector is attached to a unit

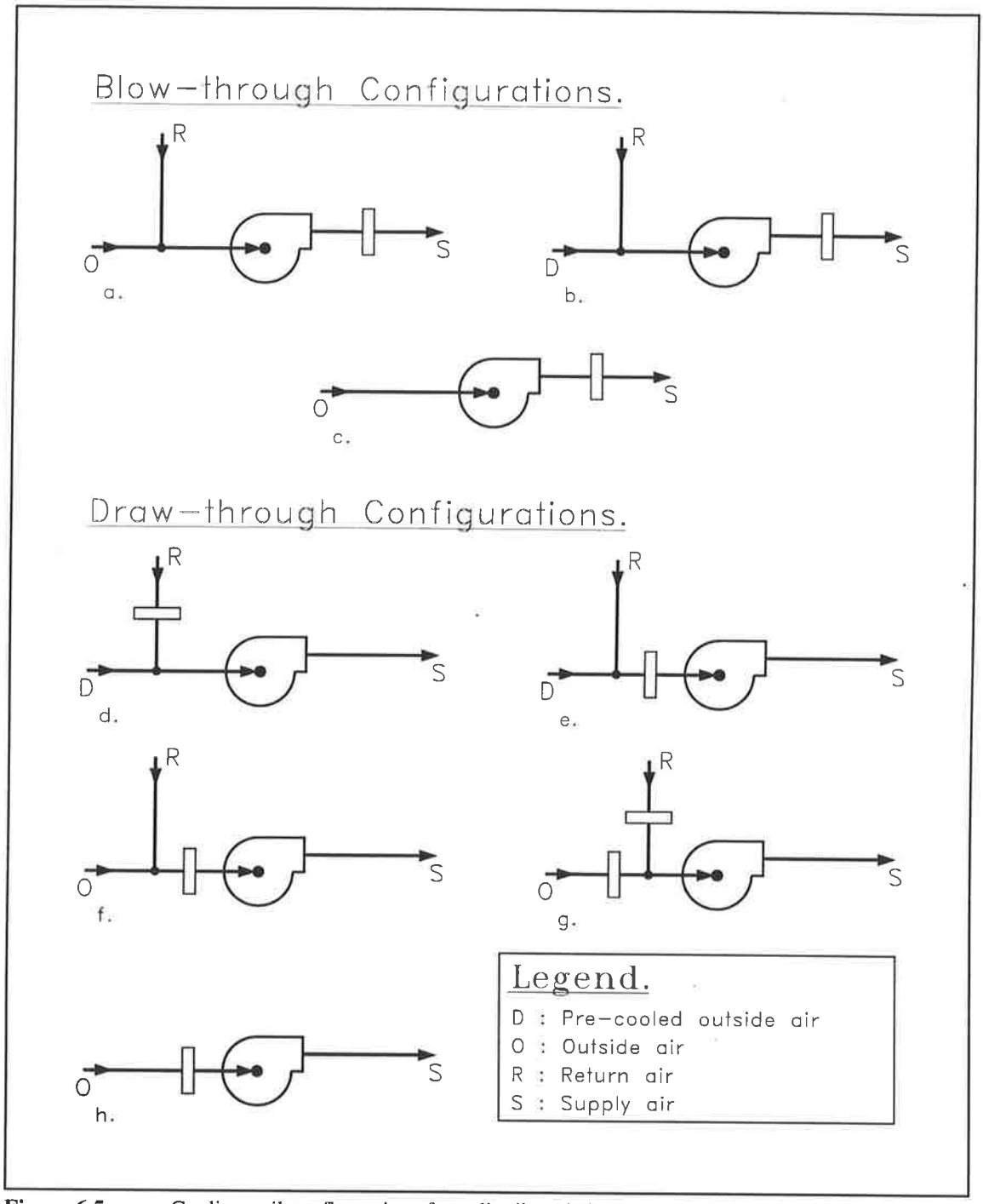


Figure 6.5. Cooling coil configurations for a distributed air handling unit.

designated as `Unspecified`, that unit will be redesignated as a `BlowThrough` unit.

b. A supply air connector which does not reference a coil bank cannot be attached to a `BlowThrough` unit.

c. When all supply air connectors have been removed from a `BlowThrough` unit, the state of the unit will be returned to `Unspecified`.

iii. A return air connector can only be attached to a unit if an outdoor air connector is simultaneously attached. Similarly, if either an outdoor air connector or a return air connector is detached from the unit, the other connector of the pair will be detached simultaneously. A distributed ahu must have one and only one outdoor air connector, and can have at most one return air connector. The operations of attaching and detaching outdoor air and return air connectors are thus subject to the following constraints:

a. An outdoor air connector which references a coil bank cannot be attached to a unit designated as `BlowThrough`. The same applies to a return air connector. If a connector of either type referencing a coil bank is attached to a unit designated as `Unspecified`, that unit will be redesignated to be a `DrawThrough` unit.

b. If both the outdoor air connector and the return air connector are detached from a `DrawThrough` unit, the state of the unit will be returned to `Unspecified`.

c. By convention, a unit to which both an outdoor air connector and a return air connector are attached, but for which only the outdoor air connector references a coil bank, will have the `combined` flag set to *true*. From figure 6.5 it will be seen that this convention permits `combined` units to be identified unambiguously.

6.2.3.2. Outdoor Air ahu.

The range of configurations which we need to make available for an outdoor air ahu is much smaller than for its distributed counterpart, as shown in figure 6.6. An outdoor air ahu is `connected` if it has the following connectors attached:

a. One and only one supply air connector. In principle a central outdoor air treatment unit could support multiple supply air connectors. However, this possibility is not currently provided for.

b. One and only one outdoor air connector.

Either the supply air connector (`BlowThrough`) or the outdoor air connector (`DrawThrough`) must reference a coil bank.

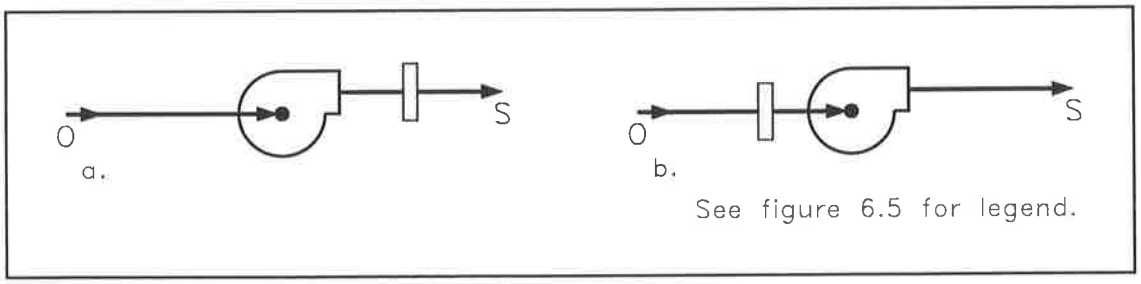


Figure 6.6. Cooling coil configurations for an outdoor air ahu.

6.3. Algorithms for System Simulation - Distributed Units.

Luxton and Shaw (1991) drew attention to "the thermodynamic requirements that if equilibrium is to be achieved in a system the rate at which heat and moisture are removed by the dehumidifier coil must equal the rate at which they are added to the conditioned space". For a single-zone air conditioning system this implies that the equilibrium condition within the zone will be uniquely determined provided

i. The load on the zone remains constant.

ii. The condition within the zone is constrained to lie on some locus on the psychrometric chart. Most often this locus will be established by thermostatic control so the constrained locus becomes a line of constant dry-bulb temperature.

iii. The cooling coil has sufficient capacity to offset the sensible load in the zone.

A further constraint may be introduced into the system, provided an extra degree-of-freedom is also introduced. A practical example, and the only one with which we will be concerned in the present study, is that of using reheat to control the relative humidity within a zone so that it remains below an upper limit. As a corollary to the above, we note that if the conditions within the zone do not lie at the equilibrium point, then the action of the control system will be to drive the zone condition towards its equilibrium point.

If the zone approaches its equilibrium condition along the locus determined by a line of constant dry-bulb temperature, the process is known as the *moisture staircase* phenomenon. This process has been graphically illustrated by Sekhar (1990), and by Luxton and Shaw (1991). It can be demonstrated by *assuming* a zone relative humidity, and plotting on a psychrometric chart the zone condition which satisfies the assumed relative humidity and the desired dry-bulb temperature. This point is used as a starting point, and the various processes occurring within an air conditioning system are plotted on a psychrometric chart. The return air is mixed with the outdoor air in the specified proportions, the coil condition curve plotted, and the coil exit condition required to offset the zone load is located. Finally, the load ratio line is plotted, thus determining a new estimate for the zone condition. In almost all cases, this will differ from the point originally chosen, although still lying on the locus. If the process is repeated several times, using the last estimate of the zone condition as the starting point for each successive construction, the sequence of points generated will be found to approach an equilibrium point monotonically. The equilibrium point is insensitive to the initial estimate.

The procedure outlined above is tedious for a single zone, even if a coil simulation programme is available. The complexity introduced by multiple zones renders hand computation totally impractical for most purposes. The problem is, however, amenable to a computerized solution. In designing the required algorithms, it is useful to decompose the problem into two nested loops:

i. The *inner loop* performs one step of the moisture staircase simulation, using the current estimate of the zone condition(s) to derive an updated estimate.

ii. Within the *outer loop*, a programmed sequence of moisture staircase steps is performed, iteration continuing until the sequence converges to the desired equilibrium condition.

The outer loop is most easily programmed using the method of *successive substitution* (Stoecker, 1989), in direct emulation of the hand computation procedure described above. Unfortunately, this method suffers from two drawbacks; it is usually slow to converge and often it may not converge at all, preferring to oscillate between two states which straddle the true solution point.

The remainder of this section is devoted to the development of a set of algorithms which will provide a stable and efficient solution to the problem of finding the system equilibrium point for a wide range of system configurations. The case of a system with an upper bound placed on the relative humidity permitted in some or all zones is also considered. The algorithms described in the following are implemented as member functions of class *DController*.

6.3.1. Reheat Coils.

Reheat coils are an integral component of many of the control systems which we seek to model in this study. In keeping with the philosophy adopted throughout, reheat coils are modelled as objects of a class *ReheatCoil*, which has the following member variables[40]:

`enabled` : A boolean variable which flags whether the reheat coil is available for use (*enabled*) or not (*disabled*).

`status` : A variable of enumerated type

```
enum ReheatStatus { off, on };
```

A reheat coil must be *enabled* before `status` can be set to *on*.

`p` : Power consumption (kW).

`dT` : Reheat temperature (°C).

In addition, member functions are provided to *enable* or *disable* the reheat coil, to toggle its status to *on* or *off*, to access the status and enabled flag, and to access and set the power and reheat temperature.

---

40 This class is in effect, although not in fact, an abstract class in the sense that it seeks to model only those aspects of the reheat coil which are of fundamental importance to the current application. Reheat coils are available employing any one of a number of different heating media (electric power, steam, hot water), and it is conceivable that details specific to a physical coil (dimensions or hot water flow rate, for instance) might be of importance in another application. This possibility is readily handled by using *ReheatCoil* as a base class for a set of classes providing member functions and variables of a more specific nature.

6.3.2. Class *DController.*

Each object of class *D_AHU* has associated with it an object of class *DController*, which provides the control functions for that air handling unit (figure 6.3). The following member variables are of particular importance in connection with the algorithms which will be developed in the following:

`cmode` : A variable of the enumerated type

```
enum DController::controlmode { VAV, CAV };
```

By default `VAV` is specified. Note that the control mode is properly a property of the controller, rather than of the ahu; most if not all of the configurations which can be assembled using the classes provided can be run in either a VAV or a CAV mode of operation.

`ZInfo` : An associative array defined by the statement

```
Map<unsigned, ZoneInfo*> ZInfo;
```

Each zone served by the unit has one entry in this array. The *key* contains an index to the zone in question. The *value* is a pointer to an object of class *ZoneInfo*, which facilitates access to and manipulation of the various zones served by the unit. The interface for class *ZoneInfo* is defined as follows:

```
class ZoneInfo
{
  int nc;
  double ZSupDuctDt;
  ReheatCoil ZReheat;
  double ZRetDuctDt;
  Zone* ptr;
public :
  // Constructor -----------------------------------------------
  ZoneInfo (Zone* z,
            const int ic = 0)
  : ZSupDuctDt (0.0), ZRetDuctDt (0.0), nc (ic), ptr (z) {}
  // Destructor ------------------------------------------------
  ~ZoneInfo () {}
  // Access and set duct temperature gains ---------------------
  double SADuctDt () const { return ZSupDuctDt; }
  double RADuctDt () const { return ZRetDuctDt; }
  void SADuctDt (const double dt) { ZSupDuctDt = dt; }
  void RADuctDt (const double dt) { ZRetDuctDt = dt; }
  // Access reheat information ---------------------------------
  ReheatCoil& r () { return ZReheat; }
  boolean ReheatOn () { return (boolean) (ZReheat.Enabled () &&
                        ZReheat.Status () == ReheatCoil::on); }
  double ReheatPower () const { return ZReheat.Power (); }
  double ReheatDt () const { return ZReheat.Temperature (); }
  // Access and set textual information ------------------------
  char* Name () { return ptr->Name (); }
  char* Comment () { return ptr->Comment (); }
  void Name (const char* nstr) { ptr->Name (nstr); }
  void Comment (const char* cstr) { ptr->Comment (cstr); }
```

```
    // Access and set connector number ---------------------------
    int Connector () const { return nc; }
    void Connector (const int n) { nc = n; }
    // Access referenced zone ------------------------------------
    Zone& Z () { return *ptr; }
};
```

Member variable `ZoneInfo::ZReheat` is a rehcat coil used to achieve *temperature* balance among the zones served by a multizone CAV system by reheating the supply air to an individual zone on a *local* basis. This should be distinguished from the variable `SA_Connector::r` (section 6.2.3), also a reheat coil, which is used for *humidity* control. The remaining member variables of this class essentially serve a housekeeping purpose. The associative array defined above provides a convenient means of visiting all zones served by the unit in sequence.

Class *DController* defines among its member functions a constructor and a destructor, together with functions to read and set as appropriate the operating variables for the controller and its associated ahu. The **public** interface of class *DController* also declares three sets of member functions which are of central importance to the modelling process. These perform the following functions:

i. Finding the equilibrium operating conditions for the system. In other words, performing a programmed sequence of moisture staircase steps until convergence is obtained.

ii. Initializing the components of the model to a state which is numerically, if not necessarily physically feasible.

iii. Setting the control mode to VAV or CAV, as desired.

The **public** functions which provide these services are supported by an extensive set of **private** member functions. These are described in detail further on in this section, in terms of both their algorithmic content, and of their implementation within a computational setting.

6.3.3. The Outer Loop.

The shortcomings of *successive substitution* as a practical means of solving the outer loop of the moisture staircase problem have been remarked upon above. An alternative solution procedure is required which improves upon this most basic and intuitive method in terms of both efficiency and robustness. Such a procedure may readily be derived for a single-zone system. Let the equilibrium condition of the zone be defined by its dry-bulb temperature (t) and its humidity ratio (W), of which t is fixed by thermostatic control, and W is to be found. Let $\tilde{W}_a$ be an initial estimate for W, and execute one iteration of the inner loop using this as a starting point such that an updated estimate $\tilde{W}_a{}'$ is produced. Assume that the value $\tilde{W}_a$ is such that $\tilde{W}_a > \tilde{W}_a{}'$. Now let $\tilde{W}_b > \tilde{W}_a$ be a second estimate which, upon completion of one iteration of the inner loop, produces an updated estimate $\tilde{W}_b{}'$ such that $\tilde{W}_b{}' < \tilde{W}_b$. Clearly then

$$\tilde{W}_a < W < \tilde{W}_b \quad . \tag{6.3.1}$$

To find the equilibrium condition for the single-zone case then, we wish to find some estimate $\tilde{W}$ of W such that

$$f(\tilde{W}) = \tilde{W} - \tilde{W}' = 0 \quad . \tag{6.3.2}$$

Provided we can write a function which will take $\tilde{W}$ as input, and return $f(\tilde{W})$, equation (6.3.2) can be solved efficiently using a root-finding procedure. The zero algorithm developed by Brent (1971) is in fact used throughout this work where it is desired to find the root of a univariate function. Further details of this algorithm, together with algorithms to find a bracketing interval $[\tilde{W}_a, \tilde{W}_b]$ which satisfies equation (6.3.1), will be found in section (7.3.6).

The single-zone case is of limited interest. The method described above readily generalizes however to handle units serving multiple zones. In so doing it is necessary to consider two separate cases:

a. Draw-through units, or blow-through units having a single supply-air connector, and
b. Blow-through units having multiple supply-air connectors.

The major complication in case (a) arises from the fact that we have to satisfy equation (6.3.2) for each and every zone served by the unit. We could of course elect to solve equation (6.3.2) for one particular zone, or alternatively to recast the problem so that we minimize $f^2(\tilde{W})$, summed over all zones. The problem can be more simply and unambiguously solved however by noting that, at equilibrium, equation (6.3.2) is satisfied not only at the zones, but also at every other point in the air conditioning cycle. Hence, for this case we may take $\tilde{W}$ to be the humidity ratio of the supply air (the air leaving the supply-air connector), and hence solve equation (6.3.2) as above.

For case (b) we need to satisfy equation (6.3.2) for each and every supply-air connector, where $\tilde{W}$ now refers to the humidity ratio of the supply air. The solutions are not independent since the return air streams are mixed, resulting in a common coil-on condition for the various cooling coils. There exist algorithms for solving systems of nonlinear equations (see for example Moré and Cosnard, 1980). However, the process of finding the root of a multidimensional problem is inherently less stable than that of solving a multidimensional minimization problem (Press et al., 1988). The current problem can readily be recast as a minimization problem. If there are m supply-air connectors, we seek to minimize the objective function

$$f_1(\tilde{W}_1, \ldots, \tilde{W}_m) = \sum_{i=1}^{m} (\tilde{W}_i - \tilde{W}_i')^2 \quad . \tag{6.3.3}$$

A wide range of solution methods are available for such problems (Arora, 1989; Gill et al., 1981), of which the quasi-Newton methods would seem to be favoured in current practice. In the present implementation the objective function (6.3.3) is minimized using the smsno

routine from the sumsl package (Gay, 1983), which uses a quasi-Newton method to minimize a function for which the derivatives cannot readily be calculated[41].

The fundamental difference between the two cases considered above is that in case (a) we are concerned with a single supply-air humidity ratio, in case (b) there may be many. By setting $m = 1$ in equation (6.3.3), case (a) could be treated as a special instance of case (b). There are however good reasons for not doing so. In the first place, the zero algorithm converges at a rate which is matched by few if any minimization codes. A more important reason emerges when we consider the case where humidity control is applied to some or all of the zones served by a unit. This in effect imposes a constraint on the solution. In case (a) it is convenient to solve the unconstrained problem first, and only then solve the constrained problem, if any of the constraints are violated. In case (b) it is conceivable that at equilibrium constraints may be active for some supply-air connectors, and not for others. The effect of the constraints must therefore be considered at each step of the iteration.

Since there are basic differences between the solution procedures used for the two cases above, it is desirable that these be clearly identified in the following. We thus introduce the labels **MS1** and **MS2** for the two procedures, defined as follows:

MS1	Solution procedure for a draw-through unit, or a blow-through unit with a single supply-air connector.

MS2	Solution procedure for a blow-through unit with multiple supply-air connectors.

The moisture staircase iteration is controlled by the top-level function `DController::Stair ()`. The logic for this function is shown in figure 6.7. The following notes refer to that diagram:

1. Before the moisture staircase iteration is initiated, certain initialization procedures must be followed. These are described in detail in section 6.3.6 and in chapter 10. To clarify the development which follows, it is appropriate to provide at this point the following notes on the initialization procedure:

    a. The various zones served by a particular supply-air connector comprise a *group* of zones. Before commencing the moisture staircase iteration, internal *consistency* must be established between the zones belonging to each group, and that consistency must be maintained throughout the iterative process. The condition of the zones within a group will be internally consistent if the condition of the air entering and leaving each zone could have been produced by a common air state at entry to the

---

[41] The original package is written in FORTRAN. This has been translated into ANSI Standard C (Kernighan and Ritchie, 1988).

supply-air duct serving the zones, subject to the duct heat gain and space load imposed on the air flow to each zone.

b. The air supplied to the various zones is subject to parasitic heat gains (or occasionally losses) as a result of heat transmission through the duct walls. For a zone index i, the supply-air temperature gain will be denoted by the nomenclature $\Delta t_{s,i}$. Similarly, the return-air path for zone i will be subject to a temperature gain or loss $\Delta t_{r,i}$. In the preliminary stages of designing a system, an estimate of the supply-air temperature gain must be supplied. Once the duct layout is specified, the temperature gain can be calculated using methods to be developed in chapter 9. Regardless of whether the temperature gains are specified or calculated, at the commencement of the moisture staircase iterative loop, the supply and return air duct temperature gains will be known. Note that as various solution points are trialled during the moisture staircase iteration the mass flow rate of air will adjust to suit the changing air density, with obvious consequences for the temperature gains. Such changes in the temperature gains will however be of very minor importance, and will be ignored in the following.

```
{
    Initialize staircase iteration;                                    (1)

    if DrawThrough or (BlowThrough and One SA_Connector only) then
                                                        (Procedure MS1)
    {
        Bracket solution;                                              (2)
        Unconstrained solution using zero to solve equation (6.3.2);   (3)
        Solve for humidity constraints;                                (4)
    }
    else                                                (Procedure MS2)
    {
        Initialize data arrays;                                        (5)
        Unconstrained solution using sumsl to solve equation (6.3.3);
    }
}
```

Figure 6.7. Logic for function Stair.

c. The air stream is also subject to parasitic heating as a result of the dissipation of fan power. We denote the resulting temperature gain by the nomenclature Δt_f. The fan temperature gain can be calculated if the duct layout and fan characteristics are known (see chapters 9 and 10). Otherwise, an estimated value must be supplied. As will be shown, once the fan temperature gain has been calculated for a given operating point, it

can be readily modified to correct for small variations in air flow rate around the nominal operating point. Thus, it is only necessary to compute the required parameters once prior to beginning the moisture staircase iteration. In the following the fan temperature gain appropriate to the current operating point will be accessed by calling function `FanUnit::TemperatureGain ()` for member variable `D_AHU::f`.

d. Before entering the moisture staircase loop, all reheat coils, both global and local, are set *off*.

2. Function zero requires that the user specify an interval within which the solution for the objective function is known to lie. Section 7.3.6.2 describes algorithms (**BI1** and **BI2**) to find a bracketing interval for an arbitrary function, given an initial estimate of the solution. A simpler procedure may be used to find the bracketing interval in the present case. Given an initial estimate of W, we evaluate the objective function (6.3.2) to give $d\tilde{W} = f(\tilde{W})$. Now, if $d\tilde{W} > 0$, we can repeat the iterative loop

$b \Leftarrow \tilde{W}$;
$\tilde{W} \Leftarrow \tilde{W}$ - 0·005;
$a \Leftarrow \tilde{W}$;
$d\tilde{W} \Leftarrow f(\tilde{W})$;

until $d\tilde{W} \leq 0$, in which case the solution lies in the interval $[a,b]$. The bracketing procedure for an initial estimate of $d\tilde{W} < 0$ may be derived in a similar manner.

3. To solve equation (6.3.2) it is necessary to implement in software a function which will evaluate $f(\tilde{W})$, given $\tilde{W}$ as sole argument. This function must furthermore be in such a form that it can be specified as an argument to zero (see footnote 42 on page 113). An algorithm to evaluate $f(\tilde{W})$ is referred to in the following as algorithm **OF1**, and is described in detail in section 6.3.4.

4. In procedure **MS1** the humidity constraints are applied *a posteriori*. An algorithm to perform this operation is referred to as algorithm **MH1** in the following. This is also described in section 6.3.4.

5. The minimization routine smsno requires an initial estimate of the solution vector to initiate the iterative procedure. Thus, the solution vector is initialized as

$$\tilde{x} = [\tilde{W}_1, ..., \tilde{W}_m] \quad . \qquad \textbf{(6.3.4)}$$

6. Function smsno performs *unconstrained* minimization. To solve equation (6.3.3) it is necessary to implement in software a function which will evaluate $f_1(\tilde{W}_1, ..., \tilde{W}_m)$, given an estimate for the solution vector $[\tilde{W}, ..., \tilde{W}_m]$, and which is in a form suitable to be passed as an argument to smsno. The proposed function must furthermore be capable of detecting and handling any active humidity constraints *internally*. An algorithm to perform this function is described in section 6.3.5, and is referred to in the following as algorithm **OF2**.

6.3.4. Solution Procedure MS1.

In procedure **MS1** the moisture staircase problem is solved in two steps. Each step may be treated as a subproblem, and is handled by an appropriate algorithm. The algorithms in question (**OF1** and **MH1**) have been introduced above in the context in which they occur. The problems which they seek to describe may be stated formally as:

OF1	For a unit configured in a manner appropriate for solution using procedure **MS1**, compute and return the value of the objective function $f(\tilde{W}) = \tilde{W} - \tilde{W}'$, where $\tilde{W}$ is an estimate of the supply-air humidity ratio, and $\tilde{W}'$ is the revised estimate after one moisture staircase iteration.

MH1	For a unit configured in a manner appropriate for solution using procedure **MS1**, for which an unconstrained solution of the moisture staircase problem has been found, check to see whether any humidity constraints have been violated. If they have, solve the corresponding humidity-constrained problem, using the unconstrained problem as a starting point.

The description of the solution of the moisture staircase problem using procedure **MS1** continues with an examination of the algorithms which solve these problems.

6.3.4.1. The Objective Function.

A software function to implement algorithm **OF1** takes some estimate $\tilde{W}$ of the supply-air humidity ratio as its sole argument, and returns the corresponding value of $f(\tilde{W})$. The software to perform this operation is configured as a top-level module which passes control to a sequence of subsidiary modules, as required[42]. The manner in which the various steps

---

[42] Function zero takes as an argument a pointer to the objective function. While in principle it is possible to take the address of a member function written in C++, the reference manual (Ellis and Stroustrup, 1990) is somewhat obscure in its description of the method involved. Thus, syntax which works with one compiler (Zortech C++ version 3.0) has been found to cause compilation errors with another (Borland C++ version 3.1). The problem may be solved in a portable and unambiguous manner by implementing the objective function as a **static** member function of class *DController*. Static member functions enjoy unimpeded access to the members of the defining class. However, wherever a static function requires access to a nonstatic member, the member must be prefixed by a pointer indicating the particular object to which it belongs. In the current case the **static** member variable `DC` is used for this purpose. This is a pointer to an object of class *DController*, which is set by `DController::Stair ()` to point to the currently accessed object (**this**). The static function may of course call nonstatic member functions, always identifying the object to which the nonstatic function belongs by means of an appropriate pointer. It is advantageous to perform the minimum amount of processing in the static function, before passing control to a nonstatic member function, or sequence of such functions. This reduces the complexity of the notation used, and with it the possibility of coding errors.

Similar considerations to the above apply wherever a pointer to a member function is to be passed as an argument to another function.

of the algorithm are shared between the coordinating modules is largely a matter of programming convenience. The programming details will not be considered in the following, except insofar as they enhance understanding of the algorithmic content.

Algorithm **OF1** is structured as shown in figure 6.8. The algorithm comprises a sequence of three major steps. These will be considered in turn below.

6.3.4.1.1. Setting Zone Conditions.

For a VAV system the dry-bulb temperature of the supply air is subject to thermostatic control. Specifying the humidity ratio will then uniquely identify the position of the supply air point on a psychrometric chart. The air flow to each zone will adjust to meet the thermostat set point for the zone, while offsetting the appropriate space loads. Having set the supply-air condition, the condition of the air entering and leaving each zone, together with the mass flow of air through each zone, may be calculated. The calculation procedure for each zone is independent of that for any other, and the conditions and air flow rates for all zones may be calculated in sequence. The steps required to achieve this for a single zone, referred to as algorithm **ZCV1** in the following, are as shown in figure 6.9. Having calculated the mass flow rate for each zone, the supply-air mass flow rate for the air-handling unit becomes quite simply

$$\dot{m}_{SA} = \sum_{1}^{n} \dot{m}_i \qquad \textbf{(6.3.5)}$$

where n is the number of zones served by the unit.

The situation becomes somewhat more complex in the case of a CAV system. We assume that a strategy is in place to minimize the reheat used by the off-peak zones. The strategy most commonly used in practice is that of setting the reheat for the most heavily loaded zone, as determined by a load analyzer, to zero. The technique used here differs slightly from that just described. Here we require that at all times:

a. *All* zones will operate at their

Given:

W_s : Supply air humidity ratio
t_s : Supply air dry-bulb temperature
$\Delta t_{s,i}$: Supply air duct heat gain for zone i

{

(Set condition of air entering zone)
$t_i \Leftarrow t_s + \Delta t_{s,i}$;
$W_i \Leftarrow W_s$;

(Find mass flow of air through zone)
$t_o \Leftarrow$ *Zone thermostat set point;*
$\dot{m} \Leftarrow q_s/[(1{\cdot}006 + 1{\cdot}8144\,W_i)(t_o - t_i)]$; (Eqn. 6.1.7)

(Set condition of air leaving zone)
$h_o \Leftarrow h_i$;
$h_o \Leftarrow h_o + q_s/\dot{m}$; (Eqn. 6.1.4)
$h_o \Leftarrow h_o + q_l/\dot{m}$; (Eqn. 6.1.3)

}

Figure 6.9. Procedure to establish the air flow rate and zone conditions for a single zone served by a VAV system (algorithm **ZCV1**).

```
{
    i ⇐ Index of zone with maximum sensible load;
    finish ⇐ false;

    while not finish do
    {
        t_o ⇐ Thermostat set point for zone i;                    (t_o is leaving dbt)
        t_i ⇐ t_o - q_s,i/[ṁ_i(1·006 + 1·8144 W_s)];             (Eqn. 6.1.7 - t_i is entering dbt)
        t_i ⇐ t_i - Δt_s,i;                                      (Supply air condition now specified)

        Δt_max ⇐ 0;
        i_max ⇐ 0;

        for each zone do
        {
            j ⇐ Zone index;
            t_i ⇐ t_s + Δt_s,j;                                  (Set zone entering condition)
            W_i ⇐ W_s;
            h_o ⇐ h_i;                                           (Set zone leaving condition)
            h_o ⇐ h_o + q_s,j/ṁ;                                 (Eqn. 6.1.4)
            h_o ⇐ h_o + q_l,j/ṁ;                                 (Eqn. 6.1.3)
            Δt ⇐ t_o - Thermostat set point for zone j;
            if Δt > Δt_max and j ≠ i then
            {
                Δt_max ⇐ Δt;
                i_max ⇐ j;
            }
        }

        if i_max = 0 then                                        (Check if dbt for any zone exceeds set point)
            finish ⇐ true;
        else
            i ⇐ i_max;
    }

    for each zone do
        Adjust reheat;                                           (Algorithm ZC1)
}
```

Figure 6.10. Procedure to establish the zone conditions for a CAV system (algorithm **ZCC1**).

respective thermostat set point. This may be achieved by the use of reheat, if necessary.

b. One zone at least will satisfy requirement (a) with its reheat set to zero.

```
{
    Δt ⇐ t_o - Thermostat set point;
    if |Δt| < ε then                                   (ε is a specified tolerance)
        Reheat ⇐ off;
    else if Reheat enabled and Δt < 0 then
    {
        Δt ⇐ |Δt|;
        t ⇐ t_i;
        h ⇐ h_i;
        t_o ⇐ t_o + Δt;                                (Shift exit condition to set point)
        Calculate Δt_z;                                (Eqn. 6.1.10)
        t_i ⇐ t_o - Δt_z;                              (Adjust entry condition accordingly)
        h_i ⇐ h_o - q_t/ṁ;
        Reheat ⇐ on;
        Reheat Temperature ⇐ t_i - t;
        Δh ⇐ h_i - h;
        Reheat Power ⇐ ṁ × Δh;
    }
}
```

Figure 6.11. Procedure to adjust the reheat for a zone (algorithm *ZC1*).

It is to be expected that the proposed method will simulate the behaviour resulting from the use of a load analyzer, in most instances. Where this is not the case, the system using the load analyzer must have some zones operating at temperatures in excess of their thermostat set point. In such a case, strict enforcement of requirements (a) and (b) above leads to a more objective assessment of the energy required by the system to maintain the specified zone conditions. To implement this strategy the reheat for each zone is initially set *off*. It is then necessary to find a zone which will operate at its thermostat set point, while all other zones operate at or below their respective set points. To this end, the zone subject to the maximum sensible load is selected as the initial trial zone, the supply-air temperature which achieves the thermostat set point for this zone is found, and the trial supply-air humidity ratio ($\tilde{W}_s$) is determined. With the supply-air condition fixed, the corresponding condition for each zone is found. If all zones are operating at or below their respective thermostat set points, the task is finished. Otherwise, the zone for which the operating temperature exceeds its set point temperature by the maximum amount is selected as a new trial zone, and the above loop is repeated. Once requirements (a) and (b) are satisfied, the reheat for those zones requiring it is set *on*, and the reheat power requirements are found.

The above procedure has been implemented as algorithm **ZCC1**, the logic for which is shown in figure 6.10. It will be noted that the final step in the algorithm is that of adjusting the reheat for each zone. This is done by invoking algorithm **ZC1** (figure 6.11), which operates on an object of class *Zone* and an associated object of class *ReheatCoil*.

6.3.4.1.2. Calculating Return-Air Conditions.

The condition of the return air is found by incrementing the dry-bulb temperature of each air stream by its return-air duct temperature gain, and mixing streams on a cumulative pairwise basis according to the relationships of section 4.3.4.

6.3.4.1.3. Setting the Chilled Water Flow Rate.

In this step of the processing we set the condition of the air entering the coil (or coils), and then seek to find the chilled water flow rate[43] which will minimize the deviation of an appropriate system temperature from its thermostat set point (the temperature *error*). The details of the procedures used to achieve these objectives will depend upon whether we are dealing with a blow-through system, or a draw-through system. Accordingly, we define two procedures, **MET1** and **MET2**:

MET1	For a *blow-through* unit, set the coil-on conditions, and find the chilled water flow rate which will minimize the deviation of an appropriate system temperature from its thermostat set point.

MET2	For a *draw-through* unit, set the coil-on conditions, and find the chilled water flow rate which will minimize the deviation of an appropriate system temperature from its thermostat set point.

The temperature error which we seek to minimize in procedures **MET1** and **MET2** has not yet been defined. The appropriate system temperature and associated thermostat will be identified according to whether the system is operating in a VAV control mode, or a CAV control mode. Objective functions are implemented for each case, taking the estimated chilled water flow rate as an argument, and returning the corresponding value for the temperature error. Procedure **OF3** (section 6.3.4.1.6) implements the objective function appropriate to VAV operation, while **OF4** (section 6.3.4.1.7) serves the same purpose for CAV operation.

6.3.4.1.4. Procedure MET1 - Blow-through Units.

The major steps involved in this procedure are shown in figure 6.12. The following notes refer to that figure:

1. For systems having a return-air connector (configurations (a) and (b) of figure 6.5), the condition of the air entering the air-handling unit is determined by psychrometrically mixing the air streams leaving the return-air and outdoor-air

---

[43] Over certain portions of the operating range of an air conditioning system, modulating the chilled water flow rate may not be the only means of achieving the zone operating conditions. An alternative scheme is described in section 6.5.

connectors according to the relationships of section 4.3.4. For systems employing 100% outdoor air (configuration (c)), the entering air condition for the ahu is set equal to the condition of the air leaving the outdoor-air connector.

```
{
    Set condition of air entering ahu;                                  (1)
    Set ahu leaving condition equal to entering condition;
    Increment air condition leaving ahu by fan temperature gain;

    for each S/A connector do                                           (2)
    {
        Set coil entering air condition equal to ahu leaving condition;
        Set mass flow rate of air through coil;                         (3)
        Find bracketing interval for chilled water flow rate;           (4)
        Call zero to minimize deviation of control temperature;         (5)
    }

    if mode = VAV then
        Adjust zone conditions;                                         (6)
    else
        Adjust reheat for each zone;                    (Algorithm ZC1)
}
```

Figure 6.12. Logic for procedure **MET1**.

2. In the model, procedure **MS1** is used only for systems having a single supply-air connector. Since procedure **MET1** will also be used in connection with procedure **MS2**, it has been set up to handle the general case of a system having an arbitrary number of supply-air connectors.

3. The mass flow rate of air entering the coil is set equal to the flow rate through the supply-air connector. This is equal to the sum of the requirements for all zones (equation 6.3.5) served by the connector. In the case of a VAV system, this will be established at the beginning of the moisture staircase step, when the zone conditions are calculated. In the case of a CAV system, the mass flow rate of air to each zone is specified *a priori*, and assumed to remain constant with varying air density. This is not strictly correct; the fan in fact provides constant *actual* volume flow rate with varying air density. The assumption of constant mass flow rate does however accord with conventional practice, where it is customary to specify a *standard* volume flow rate, without reference to the conditions to which it applies.

4. To complete the moisture staircase step it is necessary to modulate the chilled water flow rate to null the deviation of some system temperature from its thermostat set point. Thus we require an objective function which will take a trial value of the chilled water flow rate ($\tilde{Q}_w$) as input, and return the corresponding temperature deviation ($\Delta t = t - t_{set}$). As in previous cases, a preliminary step in solving this problem involves finding a pair of estimates for the independent variable ($\tilde{Q}_w$) which will bracket the solution point. Given an initial estimate for $\tilde{Q}_w$, a suitable

bracketing interval may be found as follows. If $\Delta t > 0$, $\tilde{Q}_w$ is repeatedly doubled[44] until a trial value for which $\Delta t \leq 0$ is found. Likewise, if $\Delta t < 0$ for the initial trial value, $\tilde{Q}_w$ is repeatedly halved until a bracketing interval has been established.

The above procedure is simplistic in that it assumes that the coil has sufficient capacity to achieve the design zone conditions while offsetting the zone loads. Strategies to handle the case where this is not so are considered in section 6.5.

5. For a VAV system one seeks to minimise the deviation of the supply-air temperature from its thermostat set point. For a CAV system, control is exerted by the zone thermostats. The desired control strategy is invoked by specifying a pointer to the appropriate objective function (**OF3** for VAV systems or **OF4** for CAV systems) as an argument to zero.

6. The new zone conditions are calculated by following the steps of algorithm **ZCV1** (figure 6.9), with the exception that we do not recalculate the mass flow rate of air. Consequently, the zone temperatures at the end of a moisture staircase step will not exactly match their thermostat set points. The deviation approaches zero as the equilibrium condition is approached.

6.3.4.1.5. Procedure MET2 - Draw-through Units.

Draw-through units exhibit a greater variety of configurations than is the case for blow-through units (figure 6.5). Thus, while the steps involved in establishing the chilled water flow rate for a draw-through unit closely parallel those for a blow-through unit, there are significant differences in detail between the two algorithms. The steps involved in procedure **MET2** are as shown in figure 6.13, to which the following notes apply:

1. "Combined" refers to units for which the outdoor-air and return-air streams are mixed before passing through a common coil (configurations (e) and (f) of figure 6.5).

2. Mixing is performed using the relationships of section 4.3.4.

3. Units for which `D_AHU::combined` is *true* have a coil bank specified for the outdoor-air connector, but none specified for the return-air coil bank (see section 6.2.3.1).

4. The procedure for finding the bracketing interval is identical to that used in procedure **MET1**. Note however that in the present instance we are modulating the *total* chilled water flow for the unit, rather than the flow for an individual connector.

---

[44] During the initialization phase (section 6.3.6), an initial estimate for $\dot{Q}_w$ is calculated. For successive trial points on the moisture staircase, the final estimate for $\dot{Q}_w$ for the preceding point is used as the initial estimate for the present point.

```
{
        if combined then                                                    (1)
        {
                Psychrometrically mix air streams leaving R/A and O/A connectors;   (2)
                Set mixed air stream as coil-on condition for O/A coil;     (3)
                Set mass flow rate of air through O/A coil;
        }
        else
        {

                if O/A coil bank present then
                {
                        Set coil-on condition equal to condition entering O/A connector;
                        Set mass flow rate of air through O/A coil;
                }

                if R/A connector and R/A coil bank present then
                {
                        Set coil-on condition equal to condition entering R/A connector;
                        Set mass flow rate of air through R/A coil;
                }

        }

        Find bracketing interval for chilled water flow rate;               (4)
        Call zero to minimize deviation of control temperature;             (5)

        if mode = VAV then
                Adjust zone conditions;                                     (6)
        else
                Adjust reheat for each zone;                    (Algorithm ZC1)

}
```

Figure 6.13. Logic for procedure **MET2.**

5. The objective functions specified are the same as those used by procedure **MET1.**

6. Note (6) for procedure **MET1** applies here also.

6.3.4.1.6. Objective Function OF3 - VAV Systems.

In a VAV system we seek to minimize the deviation of the supply-air temperature from its thermostat set point. We therefore require an objective function which will take the estimated chilled water flow rate ($\tilde{Q}_w$) as its sole argument, calculate the corresponding coil-off condition, and hence the supply-air temperature, and return the deviation ($\Delta t = t_s - t_{set}$) of the supply-air temperature from its thermostat set point. To achieve these aims a procedure is required which will set the chilled water flow rate for the various coil banks in the unit equal to the trial flow rate, find the corresponding coil-off condition, and hence set the condition of the air leaving the supply-air connector. The

```
{
    if combined then                                   (Configurations (e) and (f))
    {
        CW flow rate through O/A connector ⇐ Q_w;
        Solve for coil-off condition;
        ṁ ⇐ Air flow rate through S/A connector;   (ṁ is air mass flow rate through ahu)
        a_in ⇐ Coil-off condition;                  (a_in is ahu entering air state)
    }
    else if outdoor air from central O/A unit then          (Configuration (d))
    {
        CW flow rate through R/A connector ⇐ Q_w;
        Solve for coil-off condition;
        ṁ ⇐ Sum of air flow rates through O/A and R/A connectors;
        a_in ⇐ Psychrometric mixture of coil-off conditions;          (Section 4.3.4)
    }
    else
    {
        if no R/A connector then          (Configuration (h); 100% outdoor air)
        {
            CW flow rate through O/A connector ⇐ Q_w;
            Solve for coil-off condition;
            ṁ ⇐ Air flow rate through O/A connector;
            a_in ⇐ Coil-off condition;
        }
        else                                                    (Configuration (g))
        {
            if chilled water to O/A coil bank first then
            {
                CW flow rate through O/A connector ⇐ Q_w;
                Solve for coil-off condition for O/A coil bank;
                R/A connector CW t_in ⇐ O/A connector CW t_out;
                CW flow rate through R/A connector ⇐ Q_w;
                Solve for coil-off condition for R/A coil bank;
            }
            else                          (Chilled water to R/A coil bank first)
            {
                CW flow rate through R/A connector ⇐ Q_w;
                Solve for coil-off condition for R/A coil bank;
                O/A connector CW t_in ⇐ R/A connector CW t_out;
                CW flow rate through O/A connector ⇐ Q_w;
                Solve for coil-off condition for O/A coil bank;
            }
            ṁ ⇐ Sum of air flow rates through O/A and R/A connectors;
            a_in ⇐ Psychrometric mixture of coil-off conditions;      (Section 4.3.4)
        }
    }
    a_out ⇐ a_in;                                  (a_out is ahu leaving air state)
    Increment dbt for a_out by fan temperature gain;
    S/A connector leaving condition ⇐ a_out;
}
```

Figure 6.14. Procedure **UCW1** to evaluate the supply-air condition corresponding to a specified chilled water flow rate for a draw-through unit.

problem of calculating the coil-off condition which will be produced by a specified coil-on condition and air flow rate, and water inlet temperature and flow rate, are considered in detail in Chapters 7 and 8 (this is problem **FP**, defined in section 7.3.2). For a blow-through unit the objective function solves for and returns the error associated with one supply-air connector only, which is specified before invoking the objective function. In solving for a blow-through unit with multiple supply-air connectors, the supply-air temperature error must be minimized separately for each connector. The basic steps involved are:

i. Set the chilled water flow rate to the coil bank equal to the trial value ($\tilde{Q}_w$)[45].

ii. Solve for the coil-off condition.

iii. Set the condition of the air leaving the associated supply-air connector equal to the coil-off condition.

In accordance with the greater diversity of configurations available for draw-through units, a more complex algorithm is required. Procedure **UCW1** (figure 6.14) will calculate the supply-air condition corresponding to a trial chilled water flow rate for those draw-through configurations shown in figure 6.5. Once the supply-air condition has been calculated, the deviation of the supply-air temperature from its set point may be found.

6.3.4.1.7. Objective Function OF4 - CAV Systems.

The algorithm to evaluate the objective function for a CAV system is structured in a similar manner to that described above for a VAV system insofar as the chilled water flow rate is modulated to adjust the supply-air temperature. The difference is that now we are not controlling the supply-air temperature directly; rather, we seek to minimize an error based on the zone temperatures. Once the supply-air condition has been adjusted to match a trial water flow rate, the zone conditions are calculated. The steps involved are the same as those described for algorithm **ZVC1** (figure 6.9), with the exception that we no longer adjust the mass flow rate of air at each step. This has been set beforehand, and remains constant. For a single-zone system, the appropriate objective function is simply the deviation of the temperature of the air leaving the zone from its thermostat set point. A method for calculating an objective function suitable for use with multizone units can be derived on the basis of the discussion of section 6.3.4.1.1. That is, we seek to find a chilled water flow rate for which, with the local reheat for each zone turned *off*, one zone temperature will satisfy its thermostat set point, and all other zone temperatures will lie at or below their respective thermostat set points. A suitable objective function for a group of n zones may be calculated as follows:

---

[45] In the algorithms descibed in this chapter find a solution for system performance when the chilled water temperature is *given*. As will be described in Chapter 12, the Zebra Kernel provides a mechanism for the efficient exchange of data and command sequences with coordinating tasks in a multi-tasking environment. Using this mechanism, it is readily possible to implement an appropriate search algorithm which can seek the optimum value of a specific operating parameter, or set of parameters, within a feasible solution space.

a. If $n = 1$, Δt is the dry-bulb temperature error for the single zone:

$$\Delta t = t_o - t_{set} \quad . \tag{6.3.6}$$

b. Where $n > 1$, Δt takes the following form:

i. If any zone dry-bulb temperature exceeds the set point for that zone, then let

$$\Delta t_i = \max(0,\ t_{o,i} - t_{set,i}) \tag{6.3.7a}$$

and

$$\Delta t = \max_{1}^{n} \Delta t_i \quad . \tag{6.3.7b}$$

ii. If all zone dry-bulb temperatures are less than or equal to their set points, let

$$\Delta t_i = t_{set,i} - t_{o,i} \tag{6.3.8a}$$

and

$$\Delta t = -\min_{1}^{n} \Delta t_i \quad . \tag{6.3.8b}$$

6.3.4.2. Handling Humidity Constraints.

Once the unconstrained problem has been solved, a check is carried out to see whether any humidity constraints have been violated. A humidity constraint will have been violated if the humidistat for a zone is *enabled* and *on*, and if the relative humidity in the zone exceeds its upper set point. Such a constraint is said to be *active*. If a constraint has been violated, the problem can be rectified by applying an additional sensible load (reheat) to the supply air, forcing the cooling coil to overcool the air to the dew-point temperature required to satisfy the constraint. In the model proposed each supply-air connector has associated with it a reheat coil which can be used to control the humidity levels in the zones served by the connector. An algorithm to solve the constrained problem follows if we observe:

a. The solution to the constrained problem will always require a greater flow rate of chilled water than the corresponding unconstrained problem.

b. If we identify a zone i for which the magnitude of the humidity ratio violation ($\Delta W_i = W_i - W_{i,set}$) exceeds that of any other zone served by the same supply-air connector for the unconstrained case, then all constraints will be satisfied with minimum expenditure of energy when the constraint for zone i is *exactly* satisfied ($\Delta W_i = 0$).

c. There is one and only one supply-air humidity ratio which will exactly satisfy an active humidity constraint.

Note that the above principles will require modification when applied to a unit having multiple supply-air coils. The necessary modifications will be dealt with in section 6.3.5.

If any constraints are active, we can identify a zone i which satisfies condition (b) above. The following algorithm will then solve the constrained problem:

i. The condition of zone i is relocated to the point determined by its thermostat setting, and by the upper bound on its associated humidistat setting.

ii. The conditions in the other zones are set *consistent* with the new condition for zone i. This establishes a target supply-air humidity ratio ($W_{s,\,target}$), which must be met if the humidity constraint is to be satisfied. In other words, we need to modulate the chilled water flow rate until the condition

$$f_w(\tilde{Q}_w) \equiv W_s - W_{s,target} = 0 \qquad \textbf{(6.3.9)}$$

is satisfied.

iii. Equation (6.3.9) can be solved using zero, provided we implement in software a function which will return $f_w(\tilde{Q}_w)$ for some trial value $\tilde{Q}_w$. A suitable function can readily be implemented by adapting the procedures described in sections 6.3.4.1.6 and 6.3.4.1.7 to serve the present purposes. Before the equation can be solved an appropriate bracketing interval must be found. A lower bound ($\tilde{Q}_a$) is established by the unconstrained solution. Now, if we let $\tilde{Q}_b^{(0)} = \tilde{Q}_a$, and define a sequence such that $\tilde{Q}_b^{(j+1)} = c.\tilde{Q}_b^{(j)}$ where $c > 1$, then if $\tilde{Q}_b$ is the first term in the sequence for which $f_w(\tilde{Q}_b) \leq 0$, the solution will lie in the interval $(\tilde{Q}_a, \tilde{Q}_b]$.

iv. For a multizone CAV system, algorithm **ZCC1** (figure 6.10) is used to set the zone conditions, and consequently to adjust the *local* zone reheat coils to the settings required to attain the dry-bulb temperatures requested by the thermostat set points. Note that we will still need to supply additional *global* reheat at the supply-air connector, as described in step (v) below. For each zone of a VAV system, algorithm **ZCV1** (figure 6.9) may be used to determine the zone conditions.

v. Let $t_{s,\,c}$ be the temperature of the air leaving the supply-air connector, and $t_{s,\,z}$ the supply-air temperature required to satisfy the current zone conditions. Then the reheat to be applied at the supply-air connector is

$$\Delta t_r = t_{s,z} - t_{s,c} \qquad \textbf{(6.3.10)}$$

and the reheat power consumption can be computed accordingly.

6.3.5. Solution Procedure MS2.

Solution procedure **MS2** is applicable in the case of blow-through units having multiple supply-air connectors. If no humidity constraints are active, a solution can be found by minimizing the objective function defined by equation (6.3.3). If any humidity constraints are active however, we cannot readily impose them after solving the unconstrained

problem, as was done in procedure **MS1**. Each coil is subject to an air-on condition, which, except in the case of a unit using 100% outdoor air, will be determined by the operating points of all the coils. Thus, solutions cannot be obtained independently for the various coils. The problem could be recast in a form suitable for soltuion by a *constrained* optimization routine. However, while this is in principle a viable option, the difficulties involved in formulating a suitable objective function are formidable. It is simpler and computationally more efficient to redefine the objective function for the *unconstrained* problem to account for the humidity constraints implicitly, and solve as previously described using smsno. Algorithm **OF2** implements such an objective function. Its purpose may be formally described as follows:

OF2 For a blow-through unit having m supply-air connectors, compute and return the value of an objective function, $f_1(\tilde{W}_1,...,\tilde{W}_m)$, which must be minimized. For an unconstrained problem, the objective function will be as defined in equation (6.3.3). Where constraints exist and may be active, the objective function will be modified to handle constraint violations.

6.3.5.1. The Objective Function.

For a blow-through unit having m supply-air connectors, procedure **OF2** will take as input the current estimate of the solution vector $[\tilde{W}_1,...,\tilde{W}_m]$, and return the corresponding value of the objective function

$$f_1(\tilde{W}_1,...,\tilde{W}_m) = \sum_{j=1}^{m} (\Delta\tilde{W}_j)^2 \quad , \tag{6.3.11}$$

where the interpretation to be placed on $\Delta\tilde{W}_j$ for a particular supply-air connector j will depend on whether or not that connector is a member of the *constraint set*. For a connector which is not a member of the constraint set, $\Delta\tilde{W}_j$ is as defined previously in equation (6.3.3):

$$\Delta\tilde{W}_j = \tilde{W}_j - \tilde{W}_j' \quad . \tag{6.3.12}$$

A supply-air connector is a member of the constraint set if at least one of the zones which it serves has a humidistat which is *enabled*. Assume that the dry-bulb temperature in each zone satisfies the thermostat set point for that zone. For each zone subject to humidity control within a group served by a specific supply-air connector, the deviation of the zone humidity ratio from its humidistat set point will be

$$\Delta W_i = W_i - W_{i,set} \quad . \tag{6.3.13}$$

Within this set we can identify a zone i, which we will refer to as the *reference* zone, for which if $\Delta W_i \leq 0$, we are guaranteed that the humidity constraints for all other zones served by that connector will be satisfied. If now $\Delta W_i = 0$ for the reference zone, the corresponding supply-air humidity ratio, $W_{sm,j}$ will be uniquely determined. The supply-air constraint for supply-air connector j will be *active* if and only if the supply-air humidity ratio

Input:

m : Number of supply-air connectors.
$\vec{x}$: Trial solution vector.

{

Check trial values within bounds; (1)
for i = *1* to m do
{
$\tilde{W}_i \Leftarrow x_i$;
Set conditions at zones served by supply-air connector i to match $\tilde{W}_{s,i}$ *and thermostat setting;* (2)
}
Compute coil-on conditions; (3)
$\sigma^2 \Leftarrow 0$;
for i = *1* to m do
{
Find chilled water flow required to offset sensible loads; (4)
$\tilde{W}_i' \Leftarrow$ *S/A humidity ratio for connector i;*
if *Connector i in constraint set* then
{
$W_{max} \Leftarrow$ *Maximum permissible S/A humidity ratio for connector i;*
if $\tilde{W}_i' > W_{max}$ then
{
Constraint active;
$\Delta W_i \Leftarrow \tilde{W}_i - W_{max}$;
Modulate water flow until coil-off humidity ratio is W_{max}; (5)
Adjust zone conditions to point determined by S/A humidity ratio of W_{max} *and thermostat set points;* (6)
$\Delta t_r \Leftarrow t_{s,z} - t_{s,c}$; (7)
}
else *Constraint not active;*
}
else *Constraint not active;*
if *Constraint not active* then
{
$\Delta W_i \Leftarrow \tilde{W}_i - \tilde{W}_i'$;
Adjust zone conditions to point determined by coil-off condition;
}
$\sigma^2 \Leftarrow \sigma^2 + (\Delta W_i)^2$;
}
}

Returns: σ^2.

Figure 6.15. Steps in the implementation of algorithm **OF2**.

$$W'_{s,j} > W_{sm,j} \quad . \tag{6.3.14}$$

We can now develop the modifications to the objective function which are required in the case of supply-air connector j, which is a member of the constraint set. Let $\tilde{W}_j$ be the current estimate of the supply-air humidity ratio, and $\tilde{W}_j'$ be the corresponding estimate

after one moisture staircase iteration, where the coil-off condition has been adjusted so as to offset the zone sensible loads, without reheat. Then, there are two possible outcomes:

i. $\tilde{W}_j' \leq W_{sm,j}$. The humidity constraint is *inactive*, and ΔW_j is as defined by equation (6.3.12).

ii. $\tilde{W}_j' > W_{sm,j}$. The humidity constraint is *active*. In this case

$$\Delta \tilde{W}_j = W_j' - W_{sm,j} \quad . \tag{6.3.15}$$

The chilled water flow rate to the coil is now modulated until the coil-off humidity ratio is $W_{sm,j}$, and the *global* reheat required to offset the zone sensible loads is calculated. The zone conditions can subsequently be adjusted accordingly.

Algorithm **OF2** is thus structured as shown in figure 6.15. A detailed description of algorithm **OF2** will necessarily draw heavily upon material already presented in section 6.3.4. Appropriate references to that material will be made in the notes of section 6.3.5.1.2. Before invoking smsno to minimize the objective function of algorithm **OF2**, it is necessary to set up the constraint set. The procedures for doing this will be described first.

6.3.5.1.1. The Constraint Set.

Given a set of n zones served by a supply-air connector, that connector will be a member of the constraint set if the humidistat is *enabled* for at least one of the n zones. Assume that a subset of $n' \leq n$ of the zones served by the connector have *enabled* humidistats. It is then possible to select one among that subset of zones for which, if the humidity constraint for that zone is satisfied at equilibrium, then we are assured that the humidity constraints for all other members of the subset will also be satisfied. We refer to this zone as the *reference* zone. If the condition at that zone is now set to the psychrometric point determined by its thermostat set point and the upper limit of the humidistat range, the corresponding supply-air humidity ratio (W_{sm}) will place an upper bound on the range of supply-air humidity ratios which will satisfy the humidity constraints for the n' zones. The reference zone and its associated maximum supply-air humidity ratio must of course be redetermined if any of the zone loads or control set points are altered.

Algorithm **ZCV** finds the reference zone and the associated maximum supply-air humidity ratio for a system. Details of the algorithm are shown in figure 6.16.

6.3.5.1.2. Notes on Algorithm OF2.

1. The software implementation of algorithm **OF2** accepts an argument `nf` (the invocation count) which is normally returned unchanged to smsno (see Gay, 1983). Returning a value of `nf` = 0 will indicate to smsno that one or more members of the trial solution vector are out of bounds, in which case the length of the search step will be shortened. Thus, lower and upper limits may be placed on the acceptable trial values for $\tilde{W}_i$. A bound is also imposed on the magnitude of the change in $\tilde{W}_i$ at each iteration.

```
{
    i ⇐ Index of first zone on S/A connector list with enabled humidistat;
    finish ⇐ false;
    while not finish do
    {
        t_o ⇐ Thermostat set point for zone i;
        W_o ⇐ Humidistat upper set point for zone i;
        if mode = VAV then
        {
            t_i ⇐ t_s + Δt_s,i;                          (Air-in dbt for zone i)
            Calculate ṁ for zone;                        (Eqn. 6.1.9)
        }
        else
        {
            Calculate Δt for zone;                       (Eqn. 6.1.10)
            t_i ⇐ t_o - Δt;
        }
        h_i ⇐ h_o;
        h_i ⇐ h_i - q_l/ṁ;                               (Eqn. 6.1.3)
        h_i ⇐ h_i - q_r/ṁ;                               (Eqn. 6.1.4)
        W_i ⇐ Value determined by (t_i, h_i);
        W_s ⇐ W_i;                                       (t_s, W_s fix supply-air)
        if mode = CAV then
            Set zone conditions determined by supply air;   (Algorithm ZCC1)
        i_max ⇐ 0;
        ΔW_max ⇐ 0;

        for each zone index j ≠ i do                     (Loop on zones served by connector)
        {
            if mode = VAV then
                Set zone condition determined by supply air;
                                                         (Algorithm ZCV1)
            if humidistat enabled and constraint violated then
            {
                ΔW ⇐ W_o - W_set;                        (Magnitude of violation)
                if ΔW > ΔW_max then
                {
                    ΔW_max ⇐ ΔW;
                    i_max ⇐ j;
                }
            }
        }

        if i_max = 0 then                                (Check if any constraints violated)
            finish ⇐ true;
        else
            i ⇐ i_max;
    }
    i_ref ⇐ i;                                           (Reference zone)
    W_sm ⇐ W_s;                                          (Supply-air humidity ratio limit)
}
```

Figure 6.16. Procedure to find the reference zone for a system (algorithm **ZCV**).

2. In the present case the zone conditions for the various supply-air connectors must be set independently. That is, given a vector $\vec{x}$ which specifies a trial value for the supply-air humidity ratio for each connector ($\tilde{W}_{s,i}$), the zones served by any particular connector are set to the condition determined by the appropriate value of $\tilde{W}_{s,i}$, and by their thermostat set points. Algorithms to achieve this are presented in section 6.3.4.1.1.

3. The procedure for calculating the return-air condition is as described in section 6.3.4.1.2, the psychrometric mixing being performed over all zones served by the unit. The return air is then psychrometrically mixed with the ventilation air such that

$$\dot{m} = \dot{m}_r + \dot{m}_v \qquad \textbf{(6.3.16)}$$

where $\dot{m}_r$ is the portion of the return air which is recirculated (total - spill air), and $\dot{m}_v$ is the ventilation air, which is usually prescribed as a standard volume flow rate. The mixing relationships are described in section 4.3.4. The dry-bulb temperature of the mixture is incremented by the fan temperature gain to find the coil-on condition, which will be identical for all supply-air connectors.

4. The required chilled water flow rate may be found by invoking zero to find the root of a suitable objective function. Algorithm **OF3** (section 6.3.4.1.6) and **OF4** (section 6.3.4.1.7) fulfil this function for VAV and CAV systems respectively.

5. To find the chilled water flow rate to achieve a target coil-off humidity ratio, an objective function f_w defined in equation (6.3.9) must be solved using zero. Steps (ii) and (iii) of section 6.3.4.2 refer. Since the chilled water consumption necessary to satisfy the humidity constraint will exceed that required to offset the sensible loads in this case also, the procedure outlined in step (iii) of section 6.3.4.2 can be used to bracket the solution here.

6. Step (iv) of section 6.3.4.2 refers.

7. Step (v) of section 6.3.4.2 refers.

6.3.6. Initialization.

Prior to initiating the moisture staircase iteration, the internal state of certain components of the system must be initialized, either to establish consistency among various groups of components, or to provide a plausible set of initial conditions for an iterative solution. In the following development it is assumed that the supply and return air duct temperature gains, together with the fan temperature gain, have been specified beforehand by the user. The temperature gains must be explicitly specified if the programme is not supplied with sufficient details of the duct layout and fan operating characteristics to permit them to be calculated. This will be the normal situation early in the design process, and even at a more advanced stage the user may elect not to provide this information. A set of algorithms is presented in chapters 9 and 10 to compute the above-mentioned temperature gains, where the duct systems and fan characteristics are fully specified. These chapters

will also describe the modifications which must be made to the following procedures to handle the case where duct and fan temperature gains are to be calculated internally.

System initialization is performed by a sequence of steps within function `DController::Stair ()`, preceding the moisture staircase iteration. These are described below.

1. *Internal consistency* is established between the zones within each group served by a particular supply-air connector. It will be recalled that the air conditions within a group of zones served by a single supply-air connector will be internally consistent if that set of conditions corresponds to a unique supply-air condition, while accounting for the diversity of space loads and supply-air temperature gains associated with the zones in the group. Implicitly, it is necessary to estimate an appropriate initial value for the supply-air condition, and in the case of a VAV system, this is the first step in establishing internal consistency among the zones. We set the dry-bulb temperature of the supply-air to the value specified by its controlling thermostat (t_{SA}), and the dew-point temperature to some value t_{SA} - Δt, where $\Delta t > 0$ and Δt must be sufficiently large to ensure that the coil-off condition corresponding to the initial supply-air condition lies below the saturation line. A value of Δt = 2·5°C is recommended. This will be adequate to cope with the situation in a draw-through unit, where the coil-off condition is necessarily closer to the saturation line than is the supply-air condition. With the supply-air condition established, the corresponding zone conditions can be found using algorithm **ZCV1** (figure 6.9).

   For a CAV system the supply-air dry-bulb temperature is not known *a priori*. A supply-air condition which satisfies the above requirements may be found using the zone conditions as a starting point. At the start of the initialization procedure, the condition of the air leaving each zone will lie on a locus determined by its respective zone thermostat setting, the exact point being established either by specifying a default relative humidity (60%), or by the outcome of a previous simulation. In the general case, internal consistency will not pertain among the zones in a group at this stage, but that does not matter for the present purposes. Let $t_{SA,i}$ be a supply-air temperature established for zone i by calculating the entering dry-bulb temperature using equation (6.1.10), and subtracting the supply-duct temperature gain. Then, a *first estimate* for the supply-air temperature can be found as

$$t_{SA} = \min_{i=1}^{n} t_{SA,i} \quad . \qquad \textbf{(6.3.17)}$$

   As in the case of the VAV system, we set the corresponding dew-point temperature to some value t_{SA} - Δt. This establishes a supply-air humidity ratio (W_{SA}) which will ensure the coil-off conditions lie below the saturation line. Now, for each zone the humidity ratio, $W_i = W_{SA}$. Rearranging equation (6.1.10) so that t_i is expressed as a function of t_o and W_i, the set of $t_{SA,i}$ can be recalculated, and an improved estimate for the supply-air temperature found using equation (6.3.17). With the suppy-air condition established , the zone conditions with reheat *off* are

readily found (see section 6.3.4.1.1), and the reheat values adjusted to give a consistent set of zone conditions by invoking algorithm **ZC1** (figure 6.11).

2. The above procedure establishes the mass flow through each zone, and the mass flow through each supply-air connector follows from equation (6.3.5). The mass flow through the fan is

$$\dot{m}_f = \sum_1^m \dot{m}_{SA,j} \qquad \textbf{(6.3.18)}$$

and through the return-air connector,

$$\dot{m}_{RA} = \dot{m}_f - \dot{m}_{OA} \quad . \qquad \textbf{(6.3.19)}$$

3. The return-air condition is found as described in section 6.3.4.1.2.

4. Methods are presented in chapters 7 and 8 to find the condition of the air and water leaving a coil, provided the entering conditions have been specified. The methods described in chapter 7 deal with simple coil components, and prescribe an iterative scheme which require that initial estimates be provided for the condition of the working fluid leaving the coil (these can be set equal to the entering conditions), and for the total refrigerating capacity of the coil. These methods are generalized in chapter 8 to handle composite coil banks of some complexity. An algorithm is presented which will initialize the various components of a coil bank given an estimate of required total refrigerating capacity. The mass flow rates and entering conditions of the working fluids need to be set beforehand.

   The mass flow rate of the air through the various coils in the system can be determined as in (2) above, while procedures to determine the coil-on conditions for given outdoor-air and return air-conditions have already been described in section 6.3.4.1.4 for blow-through units, and in section 6.3.4.1.5 for draw-through units. With the supply-air condition estimated as in (1) above, we also have an *estimate* for the required coil-off condition[46], which is sufficiently accurate to provide an approximate initial estimate for the coil bank capacity. For configurations having a single cooling coil, the coil capacity is readily found given coil-on and coil-off conditions (section 7.8 refers). Where the outdoor air and return air are treated through separate coils (configuration (g) of figure 6.5), the coil-off condition on which an estimate is based is produced by mixing the two air streams in the appropriate proportions. The coil-off conditions for the individual coils are not uniquely determined. The ambiguity is resolved for initialization purposes by *assuming* that the air leaving the outdoor-air coil is saturated, and at the same dry-bulb temperature as the mixture, thus fixing the coil-off condition for the return-air coil, and permitting the capacities for both coils to be estimated.

---

[46] Adjusted in the case of draw-through units to allow for the fan temperature gain.

The entering water temperature is specified (see note 13). If we assume a water temperature rise (2·5 °C or 5 °C in the present work), an initial estimate for the mass flow rate of chilled water follows from equation 8.4.4.

5. The supply-air reheat coil, which controls humidity levels, is set *off*.

6.4. Algorithms for System Simulation - Outdoor Air Units.

The following discussion refers to a class of unit which is designed to ingest outdoor air at ambient conditions, cool it to a temperature which is more or less determined by the control strategy in use, and distribute it to a network of terminal units. Such units are used in conventional precooling schemes. They are also a central component of the high driving potential (HDP) system, configurations (d) and (g) figure 6.5, which is described and compared with its conventional counterpart in section 5.2.2.

The intention herein is to develop a coordinating set of classes which are sufficiently broad in content to simulate the range of configurations and control strategies with which we will be concerned. From a control point of view, the primary difference between a subsystem controlling a distributed unit and one controlling an outdoor-air unit is the absence in the latter of the feedback loop provided by the return air. This simplifies the control strategies considerably. It also reduces the range of possible unit configurations to just two, as shown in figure 6.6. It is conceivable that a blow-through unit *could* have multiple supply-air connectors, but the model will provide for one only, with little loss of generality.

In the model developed in the following, the rate at which air is supplied by the central outdoor-air unit will always exactly balance the ventilation requirements of the terminal units. Mechanisms to achieve this balance are discussed in chapters 9 and 10. No feedback from the terminal units is provided in respect of the supply-air temperature. Two alternative mechanisms are provided to control the chilled water flow rate:

1. The chilled water flow rate is preset and is maintained constant. This is the preferred control method for an HDP system circuited as shown in figure 5.11. In this arrangement, in which the outdoor-air unit is circuited *in series* with the terminal units, the chilled water flow through the outdoor air unit must be at least equal to the total requirements of the terminal units *at all times*. This can be guaranteed by presetting the flow rate to a value which is equal to or greater than the amount demanded by the terminal units under simultaneous peak conditions. If the unit is designed to deliver supply air at 8-10 °C at peak, when supplied with chilled water at 6-7 °C, the supply-air temperature will vary over a very narrow range as the ambient temperature decreases.

2. The supply-air temperature is controlled by a thermostat modulating the chilled water supply. This is identical to the control mechanism used by a distributed VAV unit, and is appropriate for use with a conventional precooling system. In this arrangement, the outdoor-air unit is circuited *in parallel* with the terminal units, and its chilled water flow may be controlled independently of the requirements of

the terminal units. Indeed, when the temperature differential between the chilled water and the supply air is large, as on a peak or near peak day, the degree of pre-cooling will vary substantially as the ambient temperature changes, unless some explicit form of control is provided.

The remainder of the present section is devoted to describing a class which supports the facilities required to implement the above-mentioned control strategies.

6.4.1. Class *OController.*

By analogy with the model developed to simulate a distributed unit and its control mechanisms, each object of class *OA_AHU* will have associated with it an object of class *OController* (figure 6.3). In connection with the discussion to follow, these member functions are of particular importance:

`cmode` : A variable of the enumerated type

```
enum OController::controlmode { WaterFlowRate, ExitDbt };
```

If `WaterFlowRate` is specified for `cmode`, the chilled water flow rate will be maintained constant at a constant preset level at all times. If `ExitDbt` is specified, chilled water flow rate will be modulated to maintain a specified supply-air dry-bulb temperature. The default control mode is `WaterFlowRate`.

`DInfo` : An associative array defined by the statement

```
Map<unsigned, DControlInfo*> DInfo;
```

Each terminal unit served by this outdoor-air unit has one entry in this array. The *key* contains an index to the terminal unit in question. The *value* is a pointer to an object of class `DControlInfo`, which facilitates access to and manipulation of the various terminal units served by this unit. The interface for class *DControlInfo* is defined as follows:

```
class DControlInfo
{
  int m;                // Multiplicity of unit
  double DSupDuctDt;    // Supply air duct temperature gain
  double DSupPipeDt;    // Chilled water pipe temperature gain
  DController* ptr;     // Pointer to referenced DController
public :
  // Constructor -----------------------------------------------
  DControlInfo (DController* d)
  : m (1), DSupDuctDt (0.0), DSupPipeDt (0.0), ptr (d) {}
  // Destructor ------------------------------------------------
  ~DControlInfo () {}
  // Access and set duct temperature gain ----------------------
  double SADuctDt () const { return DSupDuctDt; }
  void SADuctDt (const double dt) { DSupDuctDt = dt; }
  // Access and set CW pipe temperature gain -------------------
  double CWPipeDt () const { return DSupPipeDt; }
  void CWPipeDt (const double dt) { DSupPipeDt = dt; }
```

```
  // Access and set multiplicity of unit -----------------------
  int Multiplier () const { return m; }
  void Multiplier (const int multiplier) { m = multiplier; }
  // Access and set textual information ------------------------
  char* Name () { return ptr->Name (); }
  char* Comment () { return ptr->Comment (); }
  void Name (const char* nstr) { ptr->Name (nstr); }
  void Comment (const char* cstr) { ptr->Comment (cstr); }
  // Access referenced DController ----------------------------
  DController& DC () { return *ptr; }
};
```

The intention of this class is entirely analogous with that of class *ZoneInfo*, described in section 6.3.2. In respect of its member variables, we note the following:

`m` : In a multistorey building it is not unusual to encounter the situation where a number of adjacent stories are identical in their floor plan, construction and intended use. While some variation in the loads to which the various floors are subjected is inevitable, floors which show this type of similarity are often treated as identical for design purposes. Specifying multiplicity $\bar{m}$ for a terminal unit indicates that the unit, and its associated zones are replicated $\bar{m}$ times throughout the building, and that each instance is subject to an identical loading cycle. By specifying a multiplicity $\bar{m} > 1$, one of course foregoes the opportunity to differentiate between units in terms of the temperature of the supply air and chilled water delivered to the unit. The temperature gains which give rise to this differentation will however be poorly defined early in the design cycle.

`DSupDuctDt` : Provides a measure of the temperature gain experienced by the air in its passage through the supply-air duct.

`DSupPipeDt` : In the HDP system of figure 5.11, the chilled water will also experience a temperature rise between the outdoor air unit and a terminal unit. This may be calculated using methods analogous to those developed in chapter 9, although that facility is not implemented in the current release of the software. Alternatively, the user may specify a chilled water pipe temperature gain.

`c` : A variable of the enumerated type

```
enum circuiting { Series, Parallel };
```

This variable specifies whether the outdoor-air unit is circuited in series with, or in parallel with the terminal units it serves. A series arrangement is the preferred option, and is specified as the default.

Class *OController* provides a comprehensive set of member functions, including a constructor and a destructor, together with routines to read and set as appropriate the operating variables for the controller and its associated ahu. Member function

```
void OController::Solve ();
```

may be invoked to solve for the supply-air condition for the outdoor-air unit, and subsequently to solve in sequence for each associated terminal unit. This procedure is described in the next section.

6.4.2. Solution Procedure for an Outdoor Air Subsystem.

The steps implemented by member function `OController::Solve ()` are the following:

1. The mass flow of air through the unit is found as the sum of the requirements of the terminal units it serves. If there are n terminal units, or sets of identical units, and each has multiplicity $\bar{m}_i$ and ventilation mass flow requirement $\dot{m}_{a,i}$, the mass flow through the outdoor-air unit will be

$$\dot{m}_a = \sum_{i=1}^{n} \bar{m}_i \times \dot{m}_{a,i} \qquad \textbf{(6.4.1)}$$

   where the sum may be evaluated by accumulating over the elements of array `OController::DInfo`.

2. The current operating conditions are set for the unit. These are:

   a. The mass flow of air, found as above.
   b. The coil-on condition, corresponding to the ambient air condition for a draw-through unit, or the ambient air condition adjusted for the fan temperature gain in the case of a blow-through unit.
   c. The chilled water entry temperature, which is specified.
   d. The chilled water flow rate if `OController::cmode = WaterFlowRate`; the target supply-air temperature if `OController::cmode = ExitDbt`.

   As in the case of a distributed unit, the entering air and water conditions for the coil bank must be set, together with the mass flow of air and water (estimated if necessary), and the coil bank initialized by invoking function `CW_coilBank::Initialize (qtd)` (section 8.4.4), where the argument specifies an argument for the required cooling capacity. It is convenient to consider the initialization and solution procedures together for each control strategy.

   a. Control mode `ExitDbt`. The dry-bulb temperature at coil exit is determined by the target supply-air temperature in the case of a blow-through unit, and by that temperature decremented by the fan temperature gain for a draw-through unit. If we *assume* that the air leaving the coil is saturated, the coil will be operating between two known conditions, and the corresponding refrigeration capacity can be found (see section 7.8). For initialization purposes, the fact that the coil-off condition determined in this manner may not be physically feasible is immaterial.

An iterative procedure must be employed to minimize the supply-air temperature error, and hence solve for the coil-off condition. This process is identical to that described in section 6.3.4.1.6.

b. Control mode `WaterFlowRate`. In this case the supply-air dry-bulb temperature is not known, but an initial estimate for the coil capacity can be made by invoking function `CW_CoilBank::EstimateCapacity ()`, which is described in section 8.4.5. The coil-off condition is then found by invoking function `CW_CoilBank::Solve ()`, and the supply-air temperature set accordingly, with appropriate adjustment being made for the fan temperature gain in the case of a draw-through unit.

3. For each terminal unit attached to the outdoor air unit:

a. The outdoor air condition is set equal to the supply-air condition for the outdoor-air unit, adjusted to allow for the appropriate duct temperature gain.

b. If the outdoor-air unit is circuited in series with the terminal units (`OController::c= Series`), the entering chilled water temperature for the terminal unit is set equal to the leaving temperature for the outdoor-air unit, incremented by the appropriate pipe temperature gain.

c. A moisture staircase iterative loop is initiated for the terminal unit by invoking function `DController::Stair ()`, after initializing the coil(s).

6.5. Extended Chilled Water Circuiting Strategies.

Recent design studies involving certain classes of HDP unit have shown a need to extend the chilled water circuiting concepts described in the foregoing. The extensions to the basic methodology are handled by modifying the definition of the *Connector* class described in section 6.2.3. These extensions are described in the following. At the same time, it is opportune to consider a question which has been left open in the above; how to handle the situation where a specified coil has insufficient capacity to achieve design zone conditions while offsetting the zone loads. This situation may be handled by treating it as an *exception* and leads naturally into a discussion of the concept of *staging* whereby the capacity of a coil may be altered physically in service to match the loads.

6.5.1. Exception Handling.

Although not providing the motivation for the work described in this section, the exception handling mechanism is central to much of the following discussion and it is appropriate to consider it first.

In the context of our present discussion, the condition which is of major concern is that in which a coil has insufficient capacity to offset the zone loads while maintaining the zone dry-bulb temperature at the level specified by its thermostat setting. As will be shown in section 13.1, the cooling capacity of a coil becomes essentially independent of coolant

velocity as the coolant velocity increases beyond a certain point. In these circumstances, the simple water flow rate doubling procedure described in section 6.3.4.1.4 is inadequate for finding the upper bound to the bracketing interval; the water flow rate may be increased indefinitely without finding a suitable upper limit. If we assume for the moment that this condition can be detected, we must determine how it should be handled.

The C++ exception handling mechanism[47] offers a convenient means of achieving this end. In this mechanism, exceptions are *thrown* at the point at which they are detected, and are *caught* by an *exception handler* implemented elsewhere within the software. The question as to what may constitute an exception is considered by Stroustrup (1990). Basically, an exception may be defined by the programmer to be any condition which is most conveniently handled by breaking out of the normal flow of programme control.

To see how this may be implemented in the present context, consider the following example. In searching for a bracketing interval for the chilled water flow rate in algorithm **MET1** (figure 6.12), it is found that the coil has insufficient capacity for the duty demanded of it. Then, an exception may be thrown thus:

```
if (...) // Capacity exceeded
 throw EDCapacityExceeded (this);
```

where what is thrown is an object of class *EDCapacityExceeded*, defined in outline as

```
class EDCapacityExceeded
{
  Dcontroller* d;
public :
  EDCapacityExceeded (Dcontroller* dc = 0) { d = dc; }
  ~EDCapacityExceeded () {}
  Dcontroller* Controller () const { return d; }
  ExceptionHandler1 ();
};
```

Bear in mind that algorithm **MET1** will be implemented as a member function of class *DController*. Thus, the argument (`this`) passed to the object of the exception class in the throw statement is a pointer to the *DController* for which the exception has been raised, and can thus be identified by the code which catches the exception. Note also that it is convenient to implement certain components of the exception handling code as member functions of the exception class.

The throw statement in fact has no effect unless the exception is raised within a `try` block, thus:

```
DController& d = ...
```

---

[47] The C++ exception handling mechanism is described in detail by Ellis and Stroustrup (1990), and by Stroustrup (1991). At the time of writing (1995) this feature is not supported by all popular implementations of the C++ language, but it is supported by the Version 10.0 of the Watcom C++ compiler which was used for the present work. Significantly, the exception handling mechanism for the Windows NT operating system is modelled closely on the C++ paradigm.

```
try {
   ⋮
   d.Stair ();
   ⋮
}
catch (EDCapacityExceeded e)
{
   ⋮
   e.ExceptionHandler1 ();
   ⋮
}
```

The block prefixed by the `catch` statement is an error handler for objects of class *EDCapacityExceeded*, and must immediately follow the `try` block. Handlers for exceptions of different types may be ganged together following the `try` block, thus:

```
try {…}
catch (EDCapacityExceeded e) {…}
catch (EDExcessCapacity e) {…}
```

It is important to note that the code which raises the exception may be nested to an arbitrary depth within the `try` block. Thus, the `throw` statement in the above example is invoked by a function which is called by another function, which is in turn passed as a parameter to procedure zero. In this situation, the C++ exception handling mechanism probably represents one of the most elegant and efficient means of returning control to a point where it can be handled.

In the current DOS-based version of the Zebra Kernel, the handler for exceptions of this type simply prints a message to the screen, and awaits input to the command-line interpreter. The user may respond by trying a larger coil, or by terminating execution. In the Windows-based version the coordinating task, which may for instance be an expert system, will be notified via a DDE[48] link that the coil is too small, and may be programmed to react accordingly.

6.5.2. Detecting Exception Conditions.

We return now to a problem posed at the beginning of the last section, that of identifying a situation in which a coil which has insufficient capacity for the duty demanded of it has been specified. In general, the choice of a coil (or coils) for a particular application may be subject to constraints on one or more of the following parameters:

i. The quantity of water consumed by the coil(s) to meet the specified conditions;
ii. The water pressure drop through the coil(s);
iii. The velocity of the water within the tubes of the coil(s).

Constraints (i) and (ii) may be imposed due to economic considerations. Increased water flow rates demand an increase in the size of the pumping equipment and pipework to handle the volume of water required, and may also increase pumping costs. Increased

---

[48] Dynamic Data Exchange.

pressure drops increase pumping costs, and may also require larger pumping equipment. Excessive water velocities increase erosion within the tubes and unnecessarily shorten the life of the equipment. For a given circuiting configuration, these three constraints (if constraints are imposed on all three parameters) are not independent, and each may be expressed in terms of the other two. Therefore, all three constraints may be expressed as a constraint on water flow rate, water velocity or water pressure drop. When reduced to a common means of comparison, one of the three constraints will dominate, and the dominant constraint may differ among coils which all meet essentially the same duty requirements. For instance, increasing the number of passes per circuit may decrease the water flow rate and velocity but have an adverse impact on pressure drop.

It would not be appropriate for the Zebra Kernel to reject possible design solutions on the grounds that one or more constraints imposed by the designer has been exceeded. It is the responsibility of the designer (or an expert system or optimization scheme operating in his stead) to determine how constraint violations should be handled. The values of the parameters in question, even if exceeding design specifications, may well provide guidance towards a suitable solution, and the Zebra Kernel should always attempt to return a solution *if one exists*. The purpose of exception handling within the Zebra Kernel is to detect situations where *no solution is possible*. An appropriate criterion is suggested by the data presented in figure 13.2, which refers specifically to coils having ⅝" tubes[49], and while the data have been derived for a specific circuiting configuration, the general conclusions seem to have universal validity. From that data it may be seen that increasing chilled water velocity beyond approximately 2·5 m/s will do little to increase the cooling capacity of a coil, regardless of the air face velocity, and in fact effectiveness of water velocity as a control medium decreases with decreasing air face velocity.

Thus, assume that we seek a bracketing interval for the chilled water flow rate of $[\tilde{Q}_a, \tilde{Q}_b]$, and we impose a limit ($\tilde{Q}_{\max}$) on the chilled water flow rate through the coil. Then, if $\tilde{Q}_{\max}$ is appropriately selected, and for some trial point on the moisture staircase we are unable to find some $\tilde{Q}_b$ such that $\tilde{Q}_b \leq \tilde{Q}_{\max}$, we may assume that the coil specified has insufficient capacity for the required duty, and raise an exception. On the basis of the simulations cited above, a value of $\tilde{Q}_{\max}$ corresponding to a water velocity within the range of 2·5 to 4 m/s will be appropriate for most cases. In view of the fact that a slight variation in the water flow rate required to meet the desired zone condition at various points on the moisture staircase is to be expected, it may be advisable to select $\tilde{Q}_{\max}$ close to the upper end of this range. In any case, the range specified is in its entirety higher than is usually permissible in practice.

Referring now to note (4) in section (6.3.4.1.4), the necessary modification to the bracketing interval search procedure is obvious. If we invoke the objective function with the initial trial value $\tilde{Q}_w$, and a temperature deviation $\Delta t^i < 0$ is returned, we can set $\tilde{Q}_b = \tilde{Q}_w$, and find some $\tilde{Q}_a < \tilde{Q}_b$ as previously. If however $\Delta t^i > 0$, it suffices to set $\tilde{Q}_a = \tilde{Q}_w$, and evaluate the objective function for $\tilde{Q}_b = \tilde{Q}_{\max}$. If the temperature deviation

---

[49] The appropriate water velocties may be expected to be a function of the tube diameter.

returned $\Delta t^{i+1} < 0$, the bracketing interval has been found. Otherwise, the coil capacity is too small, and an exception is raised.

The moisture staircase algorithms described in the earlier sections of this chapter are based on the assumption that at equilibrium, the zone temperatures are identically equal to their thermostat set points. If the coil has insufficient capacity to achieve these conditions, no solution is possible, and an exception is raised. In a real-life situation of course, an equilibrium condition will always be reached. If the coil is undersized, the moisture staircase iteration will no longer be constrained to a locus described by the zone thermostat settings. The locus will instead be determined by the capacity of the system, and zone temperatures will in general exceed their thermostat settings. In this case we may develop the outline of a solution procedure for a single-zone system by analogy with the discussion of section 6.3.3. In this case, let $Q_w^m < \tilde{Q}_{\max}$ be the maximum permissible water flow rate for the system. We seek then to minimize the function (cf equation 6.3.2)

$$f(\tilde{W},\tilde{t}) = (\tilde{W} - \tilde{W}')^2 + (\tilde{t} - \tilde{t}')^2 \qquad \textbf{(6.5.1)}$$

subject to the constraint that $Q_w = Q_w^m$. The generalization of this function follows from the discussion of section 6.3.3.

To solve equation (6.5.1) it is necessary to know the functional relationship which exists between the zone temperatures and their loads. For a VAV system, we also need to know the fan control characteristics. The former information must be supplied from some source external to the Zebra Kernel, such as a load calculation programme. Assume that such information may optionally be available, possibly through a DDE link to a load calculation programme. The exception handling mechanism described in section 6.5.1 may be extended to handle this latter case thus:

```
Dcontroller& d = …
try
{
  try
  {
    :
    d.Stair ();
    if (…)      // Chilled water flow exceeds maximum permissible
      throw EDCapacityExceeded (d);
    :
  }
  catch (EDCapacityExceeded e)
  {
    :
    if (…)      // If access to load calculation programme is available
    {
      :
      // Solve equation (6.5.1)
      :
    }
    else
      throw;
  }
}
catch (EDCapacityExceeded e)
{
  :
```

```
    e.ExceptionHandler1 ();
    :
}
```

There are several points to note from the above example:

1. If the coil capacity is exceeded, this will be detected as before, and an exception thrown during the execution of function `d.Stair ()`. It may also happen that the moisture staircase routine iterates to completion, but that the water flow rate exceeds that available ($Q_w > Q_w^m$). An exception is also raised in this case.

2. All exceptions are handled in the first instance by the exception handler immediately following the inner `try` block. An exception is regarded as having been handled once execution enters an exception handler. This inner exception handler invokes the code to solve equation (6.5.1), provided access can be gained to a load calculation programme, or some other means of determining the functional relationship between the zone temperatures and their loads. If such access is not available, the exception is re-thrown to be caught by the exception handler immediately following the outer `try` block.

3. It is worthwhile to solve equation (6.5.1) even where a system with insufficient capacity to meet the specified zone conditions is unacceptable. The extent to which the equilibrium zone temperatures deviate from their thermostat set points provides a figure of merit for the system which may guide the designer towards a more satisfactory solution.

6.5.3. Self-contained HDP Units.

In the description of the principles of the HDP system contained in section 5.2.2 the discussion centred on systems employing a central outdoor air treatment plant. In the current section the emphasis will be upon self-contained HDP units, in which separate outdoor and return air coils are mounted within a common casing, and fed in series from a common chilled water source. Some of the control strategies to be described may however be adapted for use in HDP systems employing a central outdoor-air treatment plant, or even in conventionally circuited systems.

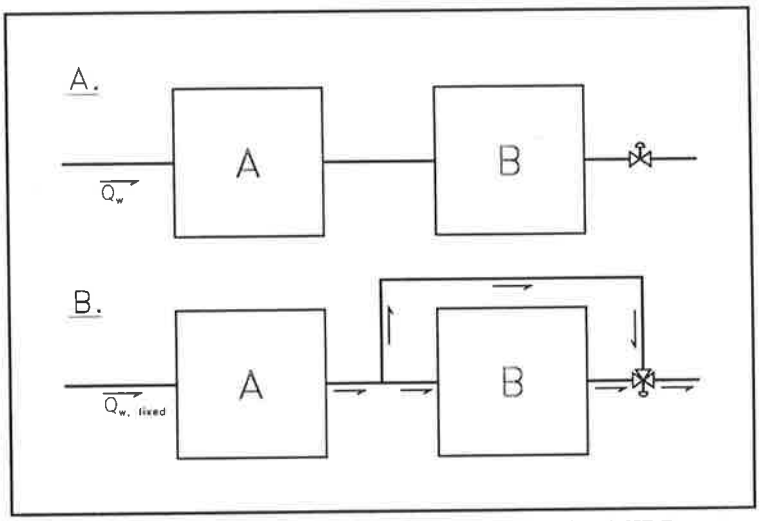


Figure 6.17. Circuiting options for a self-contained HDP system with all circuits fed.

In the following discussion the two coils will be referred to as A and B, where coil A is closer to the chilled water source than coil B, and thus possesses

the higher dehumidification potential. In practice coil A will almost always be the outdoor air coil, although it is possible to envisage situations in which the latent heat component of the return air is sufficiently high to justify reversing the usual arrangement.

At peak load and high part-load conditions it is assumed that all circuits of both coils are active, and that the supply air temperature is controlled by modulating the water flow rate. Two arrangements are possible, as shown in figure 6.17. In diagram A the water flow rate through both coils is modulated using a single valve. In diagram B the water flow rate through the system, and hence through coil A is constant, while that through coil B is modulated by a three-way valve. If coil A is the outdoor air coil, the arrangement is thermodynamically equivalent to an HDP system employing a central outdoor air treatment plant. Note that in diagram B it is possible for the flow to either A *or* B[50] to be modulated in the manner shown.

As the load falls it may be desirable to reduce the active coil surface area, thus maintaining the water flow rate in, and hence the dehumidification potential of the remaining active circuits. Two strategies are available to achieve this end:

(i).a. Coil A or coil B (but not both) may be deactivated, provided the necessary bypass circuiting is in place.

(i).b. The active size of either coil may be reduced by selectively deactivating circuits.

The above strategies may be combined with one another on falling load, or with other control actions, to provide a comprehensive control strategy. The additional control actions which may be initiated as the load falls are the following:

(ii).a. In an arrangement such as that shown in diagram B, the fixed water flow rate may be reduced to a lower (fixed) level.

(ii).b. A coil bypass may be activated to bypass a fixed percentage of the (usually fixed) total water flow around a coil.

To illustrate the type of combined strategy which is possible, consider the following scenario. A system such as that shown in diagram B of figure 6.17 is designed to maintain a zone temperature of 24 °C at peak load, at which point the chilled water flow rate through both coils is 1 L/s. As the load falls, the flow through coil B is modulated to maintain the correct zone temperature, until a point is reached at which this coil provides no effective cooling capacity. At that point it is deactivated entirely. With the capacity of the system now fixed, the zone temperature can no longer be maintained with a further fall in load, and is allowed to fall until it reaches 21 °C. At that point the fixed water flow rate is

---

[50] In principle complete control over zone temperature and humidity may be achieved by separately modulating the chilled water flow through coils A and B. The feasibility of such a system is currently under investigation, and will not be considered further in this thesis.

reduced to say 0·7 L/s. If with further fall in load the room temperature again falls to 21 °C, reheat can be employed to maintain the zone temperature at that level[51].

In order to accommodate the additional control actions described above, certain enhancements to the control structures and algorithms described in the foregoing are required. The control logic necessary to deactivate coils either entirely or partially (control actions (i).a and (i).b above) is contained within class *CoilBank*, and will be described in Chapter 8. To handle the remaining control actions ((ii).a and (ii).b), the following member variables are added to class *Connector*:

`modulated` : A boolean variable which flags whether the chilled water flow through the coil bank served by the connector (if any) is modulated.

`bypass` : The fraction [0...1] of the total chilled water flow to the ahu which bypasses the coilbank (if any) served by the connector. This variable only has real signifance if the water flow rate to the ahu is fixed; otherwise it is set to 0 if the coil is active, and 1 if it is deactivated. For an ahu with fixed water flow rate, `bypass` may be queried to find the portion of flow bypassing a coil if the flow through the coil is modulated, and may be set to specify the portion of flow bypassing the coil where this parameter is fixed. It defaults to 0.

These member variables operate in conjunction with the appropriate members of class *CoilBank*, and with member variable `D_AHU::qfs` (see section 6.2.3.1) to specify the control actions imposed on the predefined coil circuiting.

The chilled water flow rate through the air handling unit is regarded as *modulated* if the flow through *either* coil is modulated. This condition is handled using algorithm **MET1**, modified as described in sections 6.5.1and 6.5.2 above. There is one additional exception condition which needs to be tested for. If the flow through one coil is fixed, and the flow through the other coil is modulated, the capacity of the fixed flow coil alone may be sufficiently high that overcooling beyond the thermostat set point will occur even with the modulating valve shut. This may be an acceptable control situation, but it is one which needs to be detected. The procedure to detect this condition is analogous to that described above for detecting an undersize coil, and will result in an exception of class *EDExcessCapacity* being thrown. The exception handler behaves in a similar manner to that described previously for exceptions of class *EDCapacityExceeded*. As in that case it is intended that, when the ability to access a load calculation programme becomes available, the relevant exception handling mechanism will be modified to use that information to determine the equilibrium condition which would be achieved by a system under those circumstances. The appropriate control action may then be taken by the user or by the programme; this issue will be touched on again briefly in the next section.

---

[51] A circuiting strategy such as that described might be employed in a residential application in the tropics, where the system would be required to provide air conditioning on a 24-hour basis. Under such circumstances, the quantity of reheat required would be minimal. In an actual installation using a system of this type, the necessary reheat was supplied by the spill air using an air-to-air heat exchanger.

If the flow is not modulated, it is assumed that the coil has sufficient capacity to cool the zones below the target temperature, in which case reheat must be applied to maintain the temperature at the setpoint. This mode of operation is only suitable for use with constant air volume systems. The required reheat is established on a global basis using the procedure established in section 6.3.4.2, and may be applied to either the supply air or the outdoor air[52]. If the coil combination in use has insufficient capacity for this mode of operation, the calculated reheat has a *negative* value. This causes an exception of class *ENegativeReheat* to be thrown. The remarks of the preceding paragraph apply to exceptions of this class also.

6.5.4. Staging.

The concept of *staging* as described for instance in Sekhar (1990) and Shaw et al. (1992) refers to the ability to deactivate various circuits of a coil as the load falls, and to reactivate the same circuits on rising load. A more general definition is preferred in the current work. A control strategy for an ahu is defined by a sequence of *stages*. Each stage must specify the control actions appropriate to the stage in question, together with a trigger variable and the threshold values on that variable which will cause a transition to a higher or lower stage. The control action is specified by a combination of the actions described in the preceding section. The trigger variables currently supported are chilled water velocity and water pressure drop[53]. To support the complete range of circuiting strategies currently under consideration, it would be desirable to support the use of room dry-bulb temperature as a trigger variable for those instances in which the chilled water flow rate is not modulated, and in which the zone conditions cannot otherwise be met using reheat. As has been noted above, to use this trigger variable presupposes that the functional relationship between zone temperature and load is available from some external source. Since the mechanism for providing this information is not yet in place, support for this trigger variable has been omitted from the current release of the software.

Staging is appropriately enough supported by class *Stage*, which defines member variables specifying the following properties:

---

52 The technique of using the spill air to supply the reheat has been noted above. In this case, application of the reheat to the outdoor air rather than the supply air has certain advantages. The mass flow rate of the air streams on both sides of the heat exchanger are the same, and hence the area of the heat transfer surfaces should be almost the same, a situation which will normally obtain in practice unless extended heat transfer surfaces are used. The flow rate of the outdoor air is considerably smaller than that of the supply air, and the heat transfer potential between the treated outdoor air and the spill air considerably exceeds the potential existing between the supply air and the spill air. Both of these considerations indicate that a smaller heat exchanger if the reheat is applied to the outdoor air rather than the supply. Note however that the same quantity of heat must be transferred, whether it is applied to the outdoor air or the supply air.

53 While the changeover points are usually selected on the basis of water velocity, the system is more likely to be implemented using water pressure drop as a surrogate for velocity. Water pressure drop is extremely temperature dependent, largely through the effect of temperature on the water viscosity. By providing the user with the option of selecting either water velocity or pressure drop as the trigger variable the actual changeover performance of the system can be compared with the ideal performance.

i. The stage number. By convention, in stage 1 the coil(s) are active and fully circuited. Stage number increases in sequence as the system capacity reduces. In the following a stage having a higher number than another is referred to as a lower stage.

ii. The trigger variable, specified as a variable of the enumerated type

```
enum c_param { WV = 0x010, WPD = 0x020 };
```

It is also necessary to specify for which coil the trigger variable will be measured. This is specified by a variable of the enumerated type:

```
enum c_coil { OAC = 0x001, RAC = 0x002 };
```

iii. The values of the trigger variable at the upper and lower breakpoints for the stage. The lower breakpoint defaults to zero, and the upper breakpoint defaults to infinity[54].

iv. Circuiting details for the outdoor air and return air connectors, defined as variables of class *Circuiting*. This class defines member variables specifying:

a. The coil circuiting in terms of circuits deactivated, and from where in the coil they are deactivated.
b. Whether the coil is active or not.
c. Whether the water flow through the coil is modulated or not.
d. The fraction of water which will bypass the coil.

v. A variable of class *CfixedFlow* (see section 6.2.3.1) which specifies whether the water flow rate through the ahu is fixed, and if so, at what level.

vi. The supply air temperature (°C). This variable is only relevant to VAV operation.

Staging is controlled by member variables and functions of class *DController*. The stages themselves are accessed by means of an associative array, defined as

```
Map<unsigned, Stage*> Dcontroller::SInfo;
```

If this array is empty, staging is not implemented[55]. In building up the array, certain constraints are imposed upon the stages:

i. The stages are numbered consecutively, starting from stage number 1.

---

[54] In fact the ANSI Standard C variable `DBL_MAX` is used. This variable is defined in header file `float.h`.

[55] This state may be tested for using a public member function

```
boolean Dcontroller::Staged ();
```

ii. The trigger variable must be the same for each stage. Clearly this constraint will need to be relaxed for stages in which a variation in the dry-bulb temperature is a valid control action.

iii. The lower breakpoint for each stage must be lower than the upper breakpoint for the next lower stage. This constraint can be checked when the array is assembled, but in itself it may not be sufficient to ensure that an overlap between one stage and the next occurs. Failure to provide an adequate overlap will result in unstable operation on the changeover. Adequacy of the overlap is perhaps most easily checked by simulating the coil operation using ZEBRA in coil simulation mode.

iv. The upper breakpoint for stage 1 must be infinity, and the lower breakpoint for the lowest stage must be zero.

The following member functions are provided to perform stage-related operations:

i. `boolean DController::SetStage (const int ns);`

This function sets the circuiting for the system to the configuration appropriate to the stage specified by the argument. A value of *true* is returned if the operation was successful, or if staging was not implemented for the unit. A value of *false* signifies that the requested stage was unattainable.

ii. `boolean DController::IncreaseCapacity ();`

Increases the cooling capacity of the system by effecting a changeover to the next higher stage. The return value has the same significance as for function `SetStage`.

iii. `boolean DController::DecreaseCapacity ();`

Decreases the cooling capacity of the system by effecting a changeover to the next lower stage. The return value has the same significance as for function `SetStage`.

iv. `int DController::CheckStage ();`

Checks whether the system is operating within the limits appropriate to the current stage. The return value has the following significance:

-1 : A decrease in capacity is required.
0 : The stage is correct.
+1 : An increase in capacity is required.

The final step is to incorporate the staging strategy into the exception handling mechanism:

```
Dcontroller& d = …
for (boolean finish = false; !finish;)
{
  finish = true;
```

```
  try {
    :
    d.Stair ();
    if (d.Staged ())
    {
      int cv = d.CheckStage ();
      switch (cv)
      {
        case -1 :
          if (d.DecreaseCapacity ())
            finish = false;
          break;
        case 0 :
          break;
        case +1 :
          if (d.IncreaseCapacity ())
            finish = false;
          else
            throw EDCapacityExceeded (this);
          break;
      }
    }
  }
  catch (EDCapacityExceeded e)
  {
    Dcontroller& d = e.Controller ();
    if (d.Staged ())
    {
      if (d.IncreaseCapacity ())
        finish = false;
    }
    if (!finish)
      e.ExceptionHandler1 ();
  }
    :
}
```

The following notes refer to the above code fragment:

1. The `try` block and its associated exception handlers are now enclosed within a loop which executes continually until the boolean variable `finish` is assigned a value of *true*. The first statement within the loop assigns a value of *true* to `finish`, and thus the loop will execute once only unless a condition is detected which will cause a value of *false* to be assigned to `finish`.

2. As before, an exception may be thrown from within the moisture staircase routine (`DController::Stair ()`) if a situation is detected in which a solution is not possible. Provided the moisture staircase iteration runs to completion, and staging is implemented, the trigger variable is checked to ensure that it is within the range of the currently selected stage. If the trigger variable is within range, no more action is required, and control of execution passes out of the loop. If a stage change

is indicated, this is performed if possible[56], and variable `finish` is assigned a value of *false*, initiating another pass through the loop.

3. The handler for exceptions of class `EDCapacityExceeded` now checks whether it is possible to make a transition to a higher stage. If this operation succeeds, the changeover is performed, and variable `finish` assigned a value of *false*, thus initiating another pass through the loop. If the operation fails, the "standard" exception handling routine is invoked.

When the Zebra Kernel is loaded, the current stage for each unit will be stage 1 unless another stage is explicitly requested. In a sequence of runs involving a loaded Zebra Kernel process, each run will be initiated using the stage settings calculated in the preceding run.

6.6. Summary.

The Zebra Kernel uses a modelling strategy based upon finding a steady-state closure for the psychrometric processes occurring within an air conditioning system. The present chapter has described the structure and implementation of a set of algorithms designed to achieve this purpose. The emphasis throughout has been on identifying and selecting the key abstractions required to provide the necessary degree of flexibility in the range of configurations which may be simulated, while minimizing redundancy in the code used to obtain this objective. In its present form the code has been used to model a wide range of systems of both conventional and novel configurations, with and without constraints imposed on room humidity. The algorithms described are implemented using a dual-loop structure which seeks *separate*, but not entirely independent solutions for the equilibrium temperature and humidity conditions. Current research is examining the possible advantages of techniques which will seek to *simultaneously* minimize the deviations of system temperatures and humidities from their equilibrium conditions, subject to the appropriate constraints.

The software provides a framework which must be augmented using suitable models for the plant items and duct and pipe networks. This separation permits those entities to be modelled to an arbitrary degree of complexity, as will be described in the following chapters. In itself the Zebra Kernel provides a test bed which may be used to examine the effect upon system performance of varying system parameters, such as chilled water temperature, and thus assist the designer in the search for an optimal combination of equipment selections and control strategies.

---

[56] If an upward stage change is indicated and no higher stages are available, an exception will be thrown. This situation should not arise. If the capacity of the system in the lowest stage exceeds that required to maintain the thermostat set point while offsetting the loads, an appropriate exception will also be thrown from within routine `DController::Stair ()`, as described in section 6.5.3 above.

Chapter 7. Cooling Coil Simulation.

The cooling coil is arguably the most important equipment item in the air conditioning cycle. Ideally, the purpose of the coil is to offset the sensible and latent loads within the conditioned space in the precise proportion in which they occur. An understanding of the processes occurring within the coil, and the ability to predict correctly the response of a coil to a set of inputs, are thus fundamental aspects of the design process. Neglect of these basic considerations is undoubtedly one of the major reasons for installed systems failing to meet specified performance criteria.

The purpose of the present chapter is to develop and describe a model for predicting the performance of simple and composite coil banks based on available test data. The model developed herein is required to operate within the framework of the system models described in the preceding chapter. Computational efficiency is thus an essential requirement of the model. Discussion is restricted to the steady-state characteristics of the coil. It has been shown by Van Aken (1993) that the time constant of a coil will be considerably less than that of the air mass contained within the building fabric surrounding the conditioned space. Also, the dynamic performance of the coil within a systems context cannot be considered without also considering the time constants of other components, such as the fan and ductwork, which may be expected to be of the same or greater order of magnitude as that of the coil. Thus, while there are circumstances in which the dynamic characteristics of the coil are of importance, they can be neglected within the context of the present study.

A final constraint on the modelling procedure relates to the range of coil geometries to be considered. The strategy adopted is to develop a general computational framework which is essentially independent of the coil geometry, and subsequently to develop techniques for rating and predicting the performance of a range of plate fin and tube coils within the context of the general framework. Given the widespread acceptance of the plate fin and tube configuration within the air conditioning and process industries, this restriction does not severely limit the applicability of the model. In any case, adaptation of the model to handle alternative coil geometries is relatively straightforward.

7.1. Physical Characteristics.

While the intention in the current chapter is to develop a computational framework for coil performance prediction which is generally applicable, much of the discussion, together with the applications considered herein, relates specifically to a class of plate fin and tube heat exchangers which have been tested in the controlled environment wind tunnel at the University of Adelaide. It is therefore opportune at this point to describe the coil geometry and to introduce the associated dimensional notation.

The basic configuration of the coil is as shown in figures 7.1 and 7.2. Typically, the coil will have copper tubes and aluminium fins, although other materials such as copper and copper-nickel are also occasionally used for the fins. The fins are pressed from aluminium

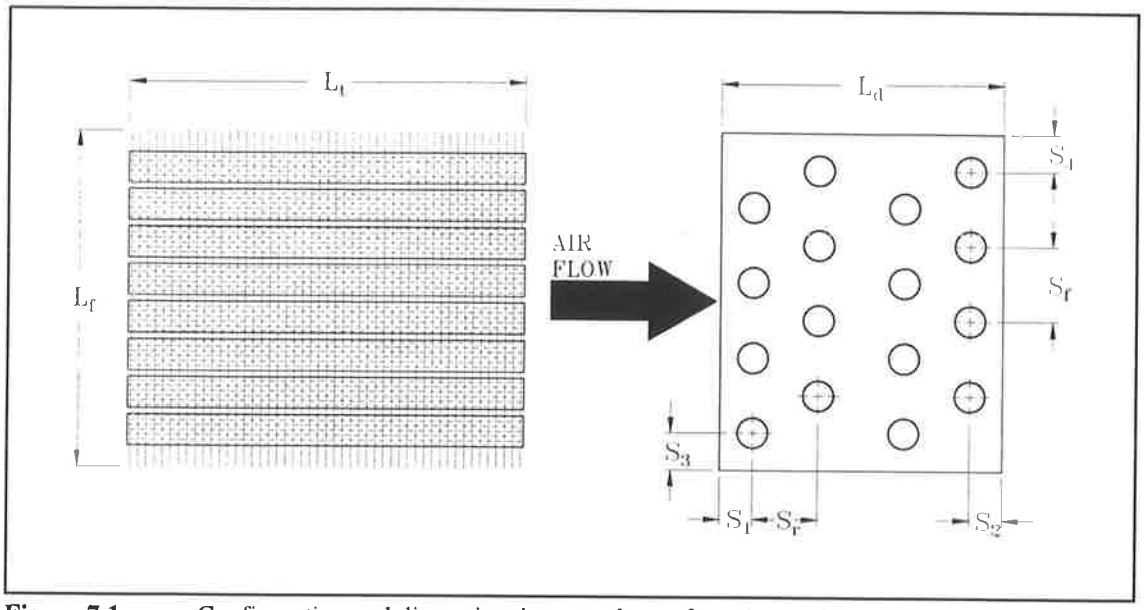


Figure 7.1. Configuration and dimensional nomenclature for a 4 row, 4 tube high cooling coil.

plate, with a series of holes with drawn down collars to maintain both the tube spacing and the fin spacing, as shown schematically in figure 7.2. The coil is assembled by expanding the tubes, usually by internal hydraulic pressure to achieve good thermal contact in the holes. The fins are rippled in the vertical direction, and along the leading and trailing edges[57]. Testing has been restricted to coils in which the tubes in succeeding rows are staggered, as shown in figure 7.1[58]; coils with an alternative inline configuration are available, but are less commonly met with in practice.

Using the dimensional nomenclature presented in figures 7.1 and 7.2, a number of further geometrical properties of the coil can be derived. If N_t is the height of the coil, expressed as the number of tubes in each row, and N_r is the number of rows of tubes in the coil, then the total number of tubes in the coil,

$$N = N_t \times N_r \quad . \tag{7.1.1}$$

Let $\dot{N}_f$ be the fin density (fins per unit length of tube). Then, the total number of fin intersections per tube,

$$N_f = trunc(\dot{N}_f . L_t) \tag{7.1.2}$$

where ***trunc()*** represents the largest integer less than or equal to the argument.

The overall coil height,

---

57 Contrary to a widely held belief, this is not done to promote turbulent flow through the coils (flow is always laminar regardless of the ripples); the vertical ripples provide stiffness and increase the surface area, while the leading and trailing edge ripples serve to minimize impact damage.

58 This differs from the corresponding illustration in ARI Standard 410-81 which shows a fin configuration which is not practically feasible! The definitions in this section refer to figure 7.1.

$$L_f = (N_t - \tfrac{1}{2})S_f + S_3 + S_4 \qquad \textbf{(7.1.3)}$$

and the coil depth,

$$L_d = (N_r - 1)S_r + S_1 + S_2 \quad . \qquad \textbf{(7.1.4)}$$

The *fin root radius* is defined as the radius of the external surface of the collar:

$$x_b = 0{\cdot}5(D_o + 2Y_f) \qquad \textbf{(7.1.5)}$$

from which the dimensions of the free-flow passage may be expressed in terms of the height of the passage between adjacent pairs of tubes,

$$F_h = S_f - 2x_b \qquad \textbf{(7.1.6)}$$

and the width of the passage between adjacent pairs of fins,

$$L_c = \frac{1}{\dot{N}_f} - Y_f \qquad \textbf{(7.1.7)}$$

where Y_f is the fin thickness.

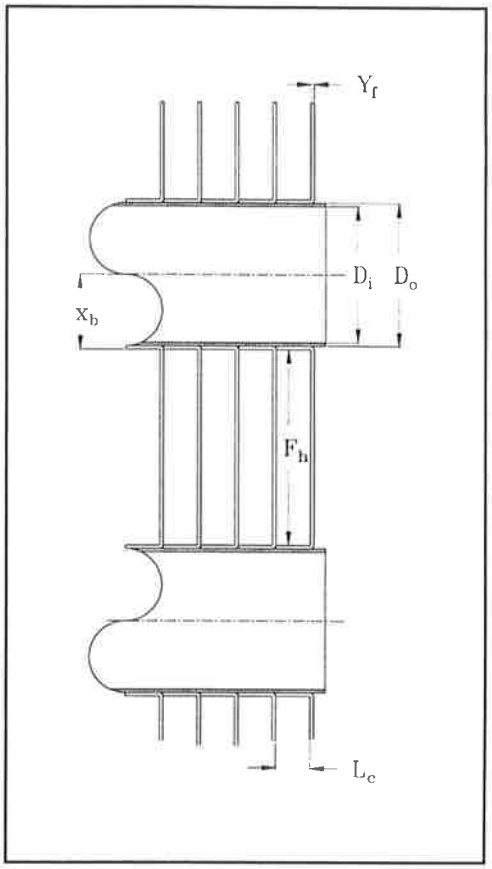


Figure 7.2. Tube and fin geometry for a plate and fin cooling coil. Thickness of fins and tube wall exagerrated for clarity.

The *coil face area* is defined as the total area (metal together with free-flow passages) presented to the air stream,

$$A_f = L_f \times L_t \qquad \textbf{(7.1.8)}$$

while the *minimum free-flow area* comprises the area presented to the impinging air stream by the free-flow passages alone, and is readily derived by subtracting the area presented by the metal from the total face area:

$$A_m = A_f - N_f(Y_f L_f - 2N_t x_b) - 2N_t L_t x_b \quad . \qquad \textbf{(7.1.9)}$$

The external surface area of the coil is conveniently divided into two components, the *primary surface area* consisting of the external surface of the fin collars, together with the external surface area of any exposed section of the tube surface,

$$A_p = N_f(2\pi x_b L_c) + \pi D_o\left(L_t - \frac{N_f}{\dot{N}_f}\right) \qquad \textbf{(7.1.10)}$$

and the *secondary surface area* comprises the surface area of the fins,

$$A_s = 2N_f[L_d L_f - \pi N x_b^2 + Y_f(L_f + L_d)] \qquad \textbf{(7.1.11)}$$

from which the *total external surface area*,

$$A_o = A_p + A_s \quad . \qquad \textbf{(7.1.12)}$$

The *total internal surface area*,

$$A_i = \pi N D_i L_t \tag{7.1.13}$$

and the ratio of total external to total internal surface area,

$$B = \frac{A_o}{A_i} \quad . \tag{7.1.14}$$

7.2. Processes within the Cooling Coil.

The processes occurring within a plate fin-and-tube heat exchanger are governed by a coupled system of partial differential equations, describing the mechanical energy, thermal and species concentration fields (Kays and Crawford, 1993). In the absence of dehumidification, the problem is reduced to one involving the thermal and mechanical energy fields only. If in addition the temperature gradients are sufficiently small that the associated density variations can be neglected, as will always be the case for the systems with which we will be concerned, the equations describing the mechanical energy and thermal fields may be decoupled and solved separately[59]. This is extremely advantageous from a computational point of view. The distribution of the local heat transfer coefficient will essentially be determined by the flow field. It is thus a valid procedure to solve first for the flow field, and only then to solve for the thermal field.

If condensation is occurring, the situation becomes considerably more complex. Under conditions similar to those described above, decoupling between the species concentration field and the thermal and mechanical energy fields will be achieved. Problems however occur in defining the boundary conditions at the wetted surface where condensation is occurring. While the interfacial temperature may be evaluated on the basis of thermodynamic considerations (e.g. Luxton and Shaw, 1991), the position of the interface remains indeterminate. The boundary conditions will not be fully specified unless the topology of the interface can be accurately characterized.

The author is unaware of any experimental investigations of the detailed flow field in a heat exchanger subject condensation, and indeed, the complexity of the flow field make it unlikely that any such investigations have yet been undertaken. However, some progress has been made in measuring the flow field and the distribution of the heat transfer coefficient for the non-condensing situation. There is also a considerable body of knowledge available concerning the mechanisms of the condensation process, based on both theoretical and experimental studies. It is reasonable to assume that a more complete understanding of the processes occurring in condensing flow in a heat exchanger will emerge from a confluence of these two lines of investigation. The purpose of the present

---

[59] This is achieved by applying the Boussinesq approximation (Sherman, 1990). In the general case, while this approximation simplifies the governing equations considerably, the two fields remain coupled through the buoyancy term. This term may also be neglected here, and for all intents and purposes, decoupling may be regarded as complete.

section is to review briefly some recent results from these areas of research which will be of relevance to the material presented in the sections which follow.

7.2.1. Non-condensing Flow in Heat Exchangers.

Rich (1975) investigated the effect of the number of tube rows on the heat transfer performance of a series of heating coils. The coils involved had a comparatively high fin density (14·5 fins/inch), employed staggered tubes, and were similar apart from the depth, which varied from one to six rows, in increments of one row. The reader is referred to the original reference for full circuiting details. The test envelope covered a range of face velocities from 1·2 to 8·1 m/s with an inlet water temperature of 68 °C and a water flow rate of 0·15 kg/s. The test rig was an open-circuit wind tunnel, and hence no control over air entering conditions was possible. When the overall heat transfer coefficient, expressed as a j-factor (equation 7.5.15) was plotted as a function of Reynolds number[60], two features became apparent:

i. At low Reynolds number the overall heat transfer coefficient drops significantly as the number of rows is increased.

ii. At the higher Reynolds rumbers the trend is reversed, although the increase in heat transfer coefficient with increasing number of rows is not nearly as pronounced as the decrease in the former case.

In a second series of experiments, Rich re-circuited his four-row coil in such a manner that it effectively comprised four one-row coils circuited in parallel. It was thus possible to measure the contribution of the individual rows to the heat transfer performance of the coil. The results were generally consistent with those observed in the earlier tests. At low Reynolds number the contribution of each succeeding row to the heat transfer performance diminishes in the direction of the air flow, while at high Reynolds number the trend is reversed.

Rich offered a tentative explanation for his observations, at least insofar as they apply to low Reynolds number performance. He noted that at very low Reynolds number a pair of standing vortices will form behind a cylinder exposed to a cross flow, and that as the Reynolds number is increased above a certain critical value, these vortices will detach from the cylinder and form a vortex street. He further noted that if the ends of the cylinder are constrained between the walls of a channel, as the spacing of the channel walls is decreased, so the critical Reynolds number defined in terms of S_r will increase[61]. In a configuration such as that under test, there is scope for standing vortices of significant size to develop in the wake of the tube. Since there is little movement of air into or out of the resulting recirculation zone, the temperature of the air within the zone will tend to approach that of the fin; the heat transfer potential will be degraded within this zone. The wake

---

[60] Rich used S_r as the characteristic length in defining Reynolds number.

[61] Rich's experimental measurements were in fact confined to one fin density (14·5 fins/inch).

generated by upstream tubes, Rich surmised, has the potential to influence the flow impinging on downstream tubes, thus accounting for the progressive degradation of heat transfer performance as the air moves through the coil.

Rich's analysis concludes with the remark that "additional measurements, preferably on a local basis, are needed to verify these conclusions and to provide a better understanding of the complex flow mechanisms which occur". Such measurements as have been published in the open literature are in accord with the general thrust of Rich's argument. They also add significantly to the detailed understanding of the processes which occur within a plate fin-and-tube heat exchanger.

Gilbert (1987) has reviewed the literature available on the subject. Only three studies need concern us here; those of Gilbert (1987), Saboya and Sparrow (1974; 1976a, b), and to a lesser extent, that of Fukui and Sakamoto (1969). Furthermore, we shall confine our attention to measurements involving staggered tube arrangements. All three studies cited include measurements of local heat (or mass) transfer coefficient. Gilbert and Fukui and Sakamoto back up their measurements with flow visualization studies. All studies have been conducted on scale or full-size models due to the difficulty of instrumenting an actual coil *in situ*.

It is perhaps convenient to consider the measurements of Saboya and Sparrow first, since these comprise the most comprehensive set of measurements of local transfer coefficient available. Their studies made use of the naphthelene sublimation technique, full details of which may be found in Saboya and Sparrow (1974). In brief, the flow between two blocks of naphthelene was used to simulate flow between the fin surfaces of a cooling coil. Delrin spacers simulated the tubes. Thus this study, like the others considered, was concerned solely with heat transfer from the fins. By exposing the surface of the naphthalene to a metered air flow for a certain length of time, and then measuring the topography of the surface, the local mass transfer could be inferred from the depth of material removed. The results are expressed by Saboya and Sparrow in terms of a local Sherwood number,

$$Sh \equiv \frac{KD_h}{D_{AB}} \qquad \textbf{(7.2.1)}$$

where K is the mass transfer coefficient, D_h is the hydraulic diameter (equation 7.5.22) and D_{AB} is the coefficient of diffusion of napthalene in air. In all of the studies cited the results were expressed as parameters of a Reynolds number based on hydraulic diameter (equation 7.5.20). The results can be expressed in terms of an equivalent Nusselt number using the mass transfer analogy (Saboya and Sparrow, 1974; Kays and Crawford, 1993). The configuration studied here is analogous to a heat transfer situation involving fin surfaces of a uniform temperature.

In a series of experiments, Saboya and Sparrow considered flow through a one-row coil (1974), a two-row coil (1976a), and a three-row coil (1976b). The geometry employed corresponded approximately to that of a coil having ⅜ inch diameter tubes and 14 fins per inch. For the one-row coil, Saboya and Sparrow present detailed measurements of the

mass transfer topography at Reynolds numbers of 1271, 648 and 214. At the highest Reynolds number, the following major features are observed:

i. A region in which the transfer coefficient is relatively high is encountered at the leading edge of the fin, where the boundary layer is still developing. Note that this region is not of great importance in a heat exchanger situation, since the driving potential for heat transfer is typically small there. Moving downstream within the boundary layer the transfer coefficient decreases, fairly rapidly at first, and then more rapidly towards the trailing edge of the fin. Directly upstream of a tube the transfer coefficient is depressed due to blockage of the flow.

ii. A horseshoe vortex system is generated at the intersection of the tube and the fin originating from flow along the stagnation line towards the fins at the front of the tube. This continues to develop as the fluid is swept around the tube, reaching maximum strength part way around the tube. At this point the transfer coefficient is significantly higher than at any other point in the flow. A weaker secondary system is observed parallel to the first, but further from the tube. The strength of the vortices then start to decay, the dominant system decaying slowly, while the secondary system has disappeared completely by the 90 degree point around the tube. At this point, the dominant vortex system separates from the tube and moves downstream. Its strength continues to decay downstream, but the transfer coefficient remains noticeably higher than that in the boundary layer.

iii. Within the wake of a tube the transfer coefficient is extremely low. At this Reynolds number the wake extends with approximately the width of the tube, and is sharply delineated on the fin surface by the horseshoe vortex until the trailing edge of the fin is reached.

As the Reynolds number reduces, the transfer coefficient is found to decrease across the surface of the fin. The effect is most pronounced in the case of the horseshoe vortex system. At Re = 648 the secondary vortex system has disappeared completely, while the primary vortex is still sharply defined, reaching a peak only part way around the cylinder from the stagnation line, but it has all but decayed by the time it reaches the separation point. The wake starts to show a tendency to fill in as the trailing edge of the fin is approached. At Re = 214, these effects are even more pronounced. The horseshoe vortex system is weak, and extends for a short distance only from the stagnation line.

For the two-row coil, Saboya and Sparrow have published test results for two values of Reynolds number, 1087 and 211. As might be expected, the topography of the transfer coefficient in the region corresponding to the first row of the coil is generally similar to that for a one-row coil. A number of interesting features are however noticed as the flow moves further downstream. Considering the case of the higher Reynolds number first, it is observed that the wake of the first row extends virtually undiminished to the trailing edge of the fin, downstream from the second row. The flow impinging on the second row is thus accelerated, causing intensification of the horseshoe vortex system. Both the dominant system and the secondary system are stronger than the dominant system of the first row, and both persist to the trailing edge of the fin. A weaker tertiary system is also

observed, which persists beyond the separation point. Interestingly, the wake is not nearly as persistent as for the first row. At the lower Reynolds number the wake from the first row continues to weaken downstream, although it is still readily discernible at the trailing edge of the fin. Again, the horseshoe vortex is intensified, and persists at least to the separation point. A wake with a diffuse boundary is observed downstream from the tube.

For the three row coil the third row lies within the wake of the first. At Re = 1087, the dominant horseshoe vortex is broader than, if not displaying quite as high peak values as the first row. The secondary system is also better developed. The channel between the tubes in this row lie within the wake of the tubes of the second row. In general, the transfer coefficient within this region is higher than within the corresponding region for the second row. Somewhat similar trends are observed at the other Reynolds numbers for which Saboya and Sparrow have published reults (643 and 216). In particular, at Re = 643 well-developed and persistent primary and secondary horseshoe vortex systems are generated around the tubes in the second row, probably due to the augmentation of the Reynolds number at the tubes of the second row due to blockage caused by the wakes of the tubes of the first row.

By integrating the local transfer coefficient over the fin surface areas corresponding to the first, second and third rows, Saboya and Sparrow (1976b) were able to estimate the relative contribution of the various rows to the total transfer capacity of the three row coil. Their results are shown below:

Re	Row 1/Total	Row 2/Total	Row 3/Total
216	0·497	0·286	0·217
643	0·422	0·316	0·262
1087	0·348	0·314	0·338

Caution should be exercised in interpreting these figures, which have been derived on the basis of isothermal fin surfaces, in the context of heat exchanger operation. In particular, the very high values for the local transfer coefficient at leading edge of the fin, a region where the fin temperature will approach the stream temperature, have added undue weight to the relative capacity of the first row. When allowance is made for this, these results are not inconsistent with the trends exhibited by Rich's experiments. A full analysis of the contribution to the total heat transfer of the various mechanisms operating at various points on the fin surface requires knowledge of the accompanying heat transfer potential. Potentially, this should be amenable to modelling using the techniques of computational fluid dynamics. It is clear however that the horseshoe vortex system is a major contributor to the overall heat transfer, and that the relative contributions by the various tube rows will be critically determined by the strength of this system for each.

Additional measurements of the local heat transfer coefficient have been provided by Fukui and Sakamoto (1969) and by Gilbert (1987). Fukui and Sakamoto used the naphthalene

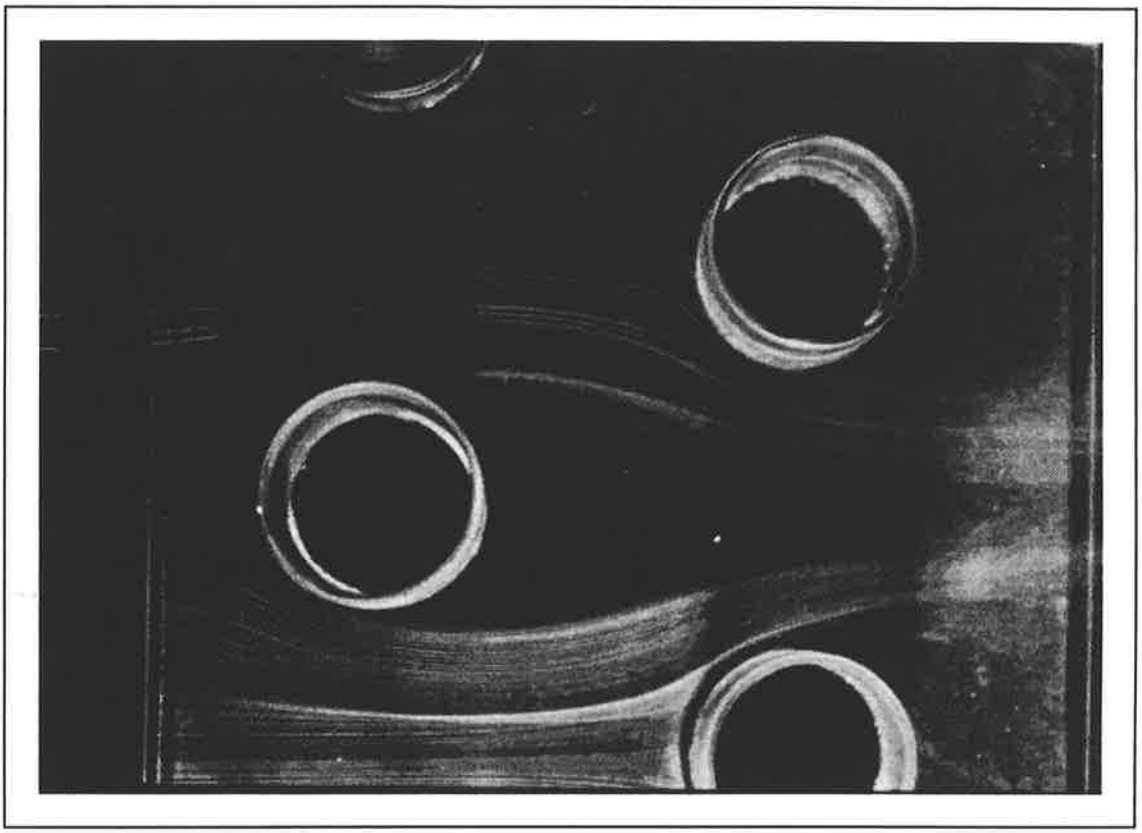


Figure 7.3. Smoke wire photograph showing flow through a staggered two-row heat exchanger model equivalent to a prototype with 4 fins per inch, operating with a face velocity of 2 m/s.

sublimation technique; Gilbert used an array of heat transfer gauges distributed across a scale model of a fin surface. Both sets of results generally confirmed those of Saboya and Sparrow.

Gilbert (1987) also studied the flow mechanisms within a heat exchanger using flow visualisation techniques on scale models simulating one and two row coils having fin spacings of 4, 6 and 8 fins per inch, and at face velocities of 1, 2, 3 and 4 m/s. Smoke wire and china clay visualization was used. Two photographs from Gilbert's smoke wire studies are reproduced here as figures 7.3 and 7.4, showing the case equivalent to flow over a two-row coil at a face velocity of 2 m/s at a fin spacing of 4 fins/inch, and at a face velocity of 4 m/s at a fin spacing of 6 fins/inch. The recirculation zone is well illustrated, as is the horseshoe vortex system. Possibly the most interesting feature is the fact that, even at this face velocity, which significantly exceeds velocities encountered in practice, the flow is at all points *laminar*. In particular, the sharp definition of the horseshoe vortex system indicates that this is a laminar vortex system. These findings run contrary to much common wisdom (e.g. Threlkeld, 1970), which asserts that the flow through heat exchanger coils in air conditioning applications is *turbulent*. By comparing the two photographs a

second feature becomes apparent; as the fin density increases, the width of the recirculation zone also tends to increase. This is consistent with the *reduction* of the Reynolds number based on the spacing between the fins which Gilbert argues convincingly is the dominant length scale in the flow. This reduction in Reynolds number as fin spacing decreases is contrary to Rich's interpretation in which he used the tube spacing as the length scale in the Reynolds number and hence deduced an *increase* in Reynolds number as fin spacing decreased. Discussions of the above interpretations can be found in Gilbert (1987), Luxton and Shaw (1991) and Van Aken (1993).

7.2.2. Condensation in Cooling Coils.

The presence of condensation on a heat transfer surface imposes an additional thermal resistance between the surface and the moist air stream. Any model of a cooling coil exposed to a dehumidifying air stream must account for this resistance in one way or another.

The literature recognizes two basic forms of dehumidification. In *filmwise* condensation the condensate is deposited in a thin film on the heat transfer surface. The underlying theory, which is based on the work of Nusselt (1916), is now well understood. In 1930 Schmidt and his coworkers described a second 'ideal' mode of condensation, in which the condensate is deposited as discrete drops. In this latter mode, known as *dropwise* condensation, the heat transfer typically exceeds that which would be experienced if the condensate was deposited as a film by at least an order of magnitude due to the greatly

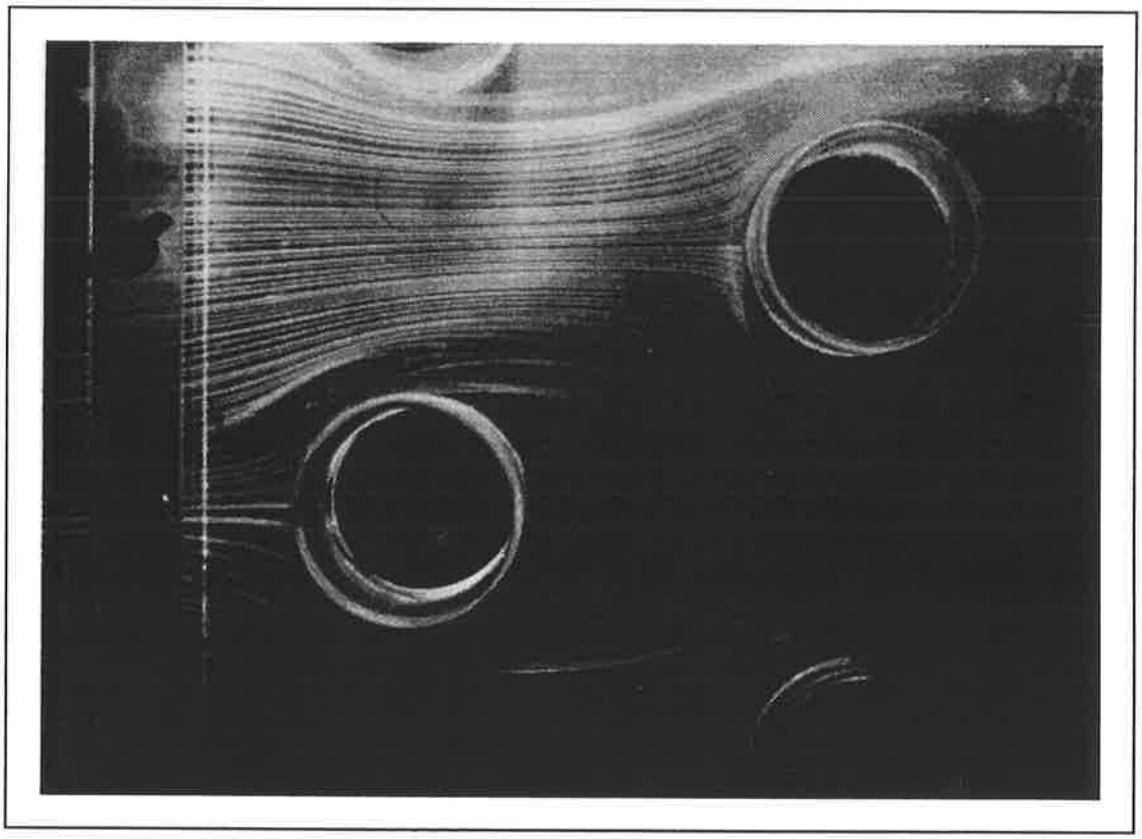


Figure 7.4. Smoke wire photograph showing flow through a staggered two-row heat exchanger model equivalent to a prototype with 6 fins per inch, operating at a face velocity of 4 m/s.

enhanced heat transfer presented by the drops and the exposed surface between the drops. As a result of this highly desirable characteristic, a great deal of effort has been expended in an effort to understand the underlying mechanisms. For the most part this work has been concerned, from both an analytical and a theoretical point of view, with condensation involving pure vapours or ideally clean and smooth surfaces. In 1966, LeFevre and Rose presented and later extended a theory to predict heat transfer in dropwise condensation (LeFevre and Rose, 1966; Rose, 1976, 1988). Although not finding universal acceptance, the LeFevre and Rose theory does provide a plausible means of modelling the various processes involved in dropwise condensation, and forms the basis for much of the work which was subsequently undertaken during the 1960's and 1970's. Due to practical problems of maintaining a stable dropwise condensation regime in industrial environments, interest in this mode of condensation has waned since the early 1980's.

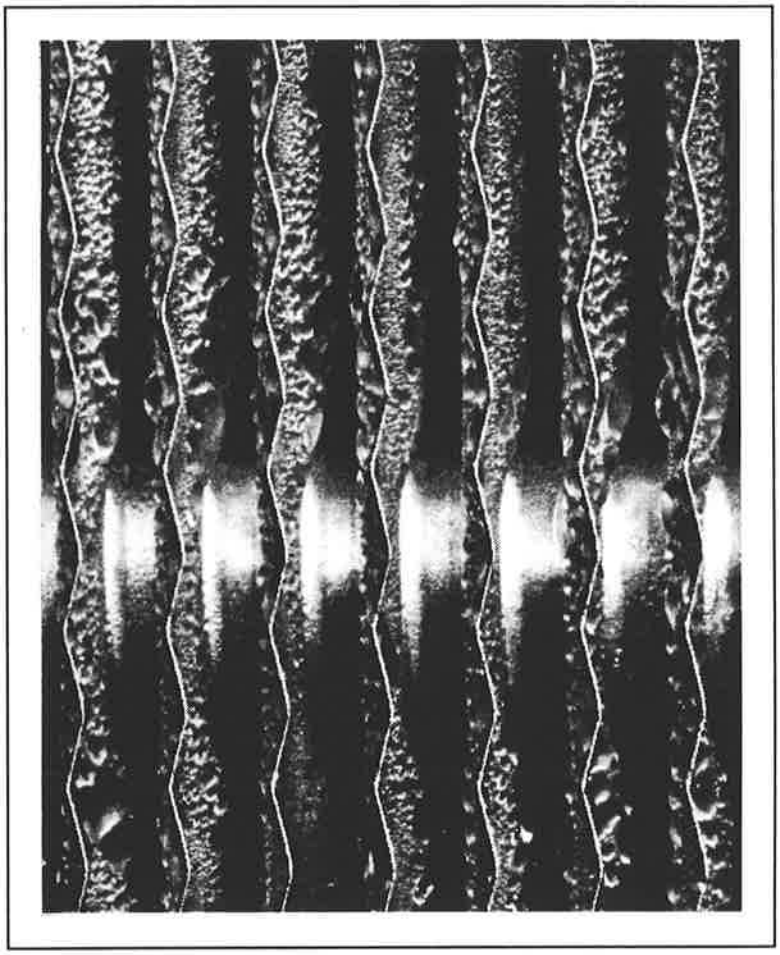


Figure 7.5. Droplike condensation on a dehumidifier coil.

It is commonly assumed that water condensing on the fins and tubes of cooling coils in air conditioning applications form a thin film across the surface (Threlkeld, 1970; ARI, 1981). While this assumption is analytically convenient, experience indicates that filmwise condensation will be the exception, rather than the rule. In a series of experiments McQuiston (1976, 1978a) concluded that it was not possible to produce filmwise condensation on copper or copper-nickel heat exchanger surfaces, and that filmwise condensation would only occur on aluminium surfaces if they were subjected to thorough cleaning beforehand. Our own experience involving tests on industrial grade heat exchangers (Luxton and Shaw, 1990) supports these findings, and indicates that condensation on heat exchanger surfaces in air conditioning practice is almost invariably in the form of discrete drops, rather than films. The point should be made that the coils used in our experiments have not been subjected to any special surface treatment, and that essentially similar condensation behaviour has been observed on a coil reclaimed after 20 years of service.

A more detailed examination of the nature of the condensate formed on vertically-oriented heat exchanger surfaces exposed to moist air (Hallion, 1988; Schroeder-Lanz, 1989; Afnan, 1990; Tai, 1991) indicates that it is incorrect to describe the condensation process observed in air conditioning coils as *dropwise* condensation, and that a third mode of condensation, for which we have coined the term *droplike* condensation, is operative in such cases. An example of this latter mode of condensation is shown in figure 7.5, which is reproduced from Hallion (1988). The basic characteristic which discriminates between the two modes of condensation is the contact angle which the drops make with the underlying surface, as shown in figure 7.6. For dropwise condensation the contact angle $\beta \geq \pi/2$, while for droplike condensation, $0 < \beta < \pi/2$. For filmwise condensation, $\beta = 0$.

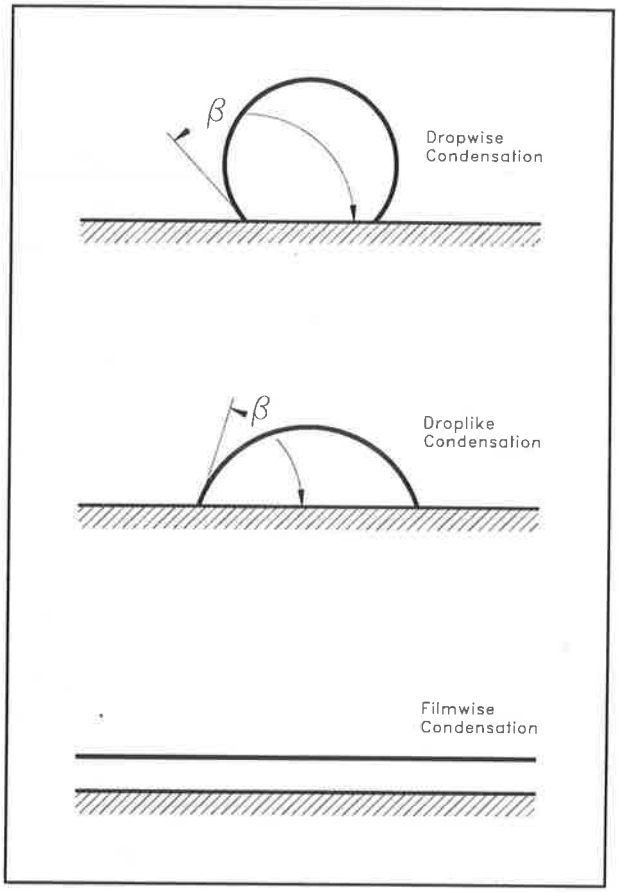


Figure 7.6. Comparison of the three modes of condensation.

The range of contact angles characteristic of dropwise condensation are associated with limited wettability of the surface. For this reason, the mechanism is very sensitive to the treatments applied to the surface on which condensation is occurring. Further factors affect the dynamics of dropwise condensation. In the case of an almost pure vapour, the presence of even minute quantities of non-condensables will significantly affect the stream-side heat transfer coefficient (LeFevre and Rose, 1965). In addition, when the stream velocity exceeds a certain critical value, there is a tendency for the drop formation to become unstable and break down to form a film, with an associated drop in heat transfer coefficient (O'Bara et al., 1967). These factors account for the decrease of interest in the dropwise condensation mode in connection with industrial processes where the high heat transfer coefficients associated with this mode would otherwise be attractive.

In respect of droplike condensation as experienced in air conditioning heat exchangers, the condensing vapour is a very small component within the main mass of the non-condensing gas. In addition, the process would seem to be relatively insensitive to the treatment accorded to the underlying surface, and to be "stable" in at least the lower part of the range of velocities encountered in air conditioning applications.

The theory of LeFevre and Rose applies specifically to the case of dropwise condensation of a pure vapour which is at rest relative to the surface. Both dropwise and droplike condensation on a vertical surface are dynamic processes in which condensation occurs in

the first instance at nucleation sites on the surface. The resulting drops grow by condensation, and by coalescence with neighbouring drops until a certain critical size is attained, at which point gravitational force overcomes the surface tension forces and the drop will run down the surface. The surface in the wake of the falling drop is scoured, but numerous microscopic drops remain which form nucleation sites for a new generation of drops. The condensation layer consequently undergoes a process of dynamic renewal, and it is reasonable to assume that the statistical distribution of the drop sizes will remain invariant with time. The LeFevre and Rose theory is not directly concerned with the dynamics of drop formation and growth. Instead, they seek to model the heat transfer processes occurring in a single drop, and by assuming a drop size distribution, to extend the analysis to calculate the average heat transfer rate for the condensate layer. In the situation modelled by LeFevre and Rose, heat transfer through a single drop is influenced primarily by three factors:

i. Dissipative normal stresses associated with interfacial matter transfer.
ii. The influence of surface curvature on the phase equilibrium temperature.
iii. Conduction through the drop.

The first factors dominate in the case of very small drops, while the last mentioned is of major importance in the case of the largest drops. For the drop size distribution, LeFevre and Rose used an expression of the form

$$\alpha = 1 - \left[\frac{r}{\hat{r}}\right]^m \qquad \textbf{(7.2.2)}$$

where,

α is the fraction of area covered by drops with radius greater than r,
$\hat{r}$ is the effective maximum radius of an adhering drop, and
m is a constant.

Subsequent work by Glicksman and Hunt (1972), Graham and Griffith (1973) and Rose and Glicksman (1973) confirm the suitability of equation (7.2.2).

The LeFevre and Rose theory has been analyzed in detail by Schroeder-Lanz (1989), who extended the theory to cover the case of droplike condensation in a process involving moist air. The major modifications required were:

i. The dissipative normal stress is the only factor contributing to the heat transfer through a drop which will be affected by the presence of non-condensables. By considering the present case as an extreme case of the condensation of a single vapour in the presence of non-condensables, Schroeder-Lanz was able to infer a suitable correction based on the available experimental evidence regarding the influence of non-condensables.

ii. The second modification concerns the shape of the drop. The expressions published by LeFevre and Rose were derived from the assumption that the drops are hemispherical ($\beta = \pi/2$), which is an appropriate assumption for the condensation of steam on a horizontal surface. The theory has subsequently been extended to handle contact angles other than $\pi/2$ (Rose, 1972).

iii. By analyzing the modification of the drop shape caused by the flowing vapour, Schroeder-Lanz inferred that this effect may safely be neglected, at least where the fluid velocity does not exceed 2·1 m/s.

Using the modified LeFevre and Rose theory, Schroeder-Lanz was able to estimate the condensate resistance for a number of experimental cooling coil tests performed by Sekhar (1990). Appropriate values for the contact angle (β) and the maximum effective drop radius ($\hat{r}$) were deduced from measurements of photographs of the droplike condensation process obtained by Hallion (1988) and by Schroeder-Lanz (1989). With the condensate thermal resistance estimated from the experimental measurements using a combination of the AU and ARI methods (see sections 7.4 and 7.5), the theoretical predictions were found to agree with the estimates to within 30%. These results can be regarded as preliminary only. However, they are encouraging. The mechanisms involved in the droplike condensation process are currently being investigated further.

7.3. Parameterization of Cooling Coil Performance.

The fin-and-tube arrangement, in both its plate-fin and helical-fin forms, has long been the preferred configuration for dehumidifying coils within the air-conditioning industry. However, the complex geometry of this configuration, and the resultant complexity of the heat and mass transfer processes occurring across the surface of the coil, render the mechanisms involved particularly intractable to detailed analytical or numerical modelling. Indeed, as has been suggested in the preceding section, our understanding of these processes is quite incomplete. Given the need to predict coil performance in the face of such uncertainties, two general classes of model have evolved.

Differential models are basically concerned with the simulation of processes at the local level, and are thus critically dependent on the ability to predict local heat and mass transfer coefficients. The work of Van Aken (1993) and Hill and Jeter (1991) provide good examples of this approach. Such models offer considerable flexibility in respect of their ability to account for non-uniformities in the air distribution, spatial variations in heat and mass transfer coefficients, and for departures from isobaric conditions through the coil. The COILSIM programme of Van Aken offers the additional benefit of modelling the transient response of the coil. In principle, the philosophy underlying the differential approach to coil simulation suggests that such models should offer a modest capability to handle operating conditions and coil geometries extending beyond the range over which they have been tested. The extent to which this is true will in fact be determined by the range of applicability of the assumptions which have been made regarding the physical processes involved, and experience suggests that, given the current state of the art, such extrapolations should be made with caution.

The flexibility of the differential approach is necessarily purchased at the expense of a certain amount of computational effort, and for the present purposes a simpler model is sought. Parameterization of the heat and mass transfer processes in terms of overall rather than local coefficients leads to an integral approach. In concept, integral models are inherently more dependent on experimental data than are those based on a differential

approach, and should only be used within the range of operating conditions, and for the coil geometries for which their performance can be guaranteed. It should be emphasized that a good integral model is based on a much more sophisticated foundation than a simple curve-fitting procedure. The use of experimental correlations within a framework based on sound physical principles guarantees that the model can be used with confidence over its range of applicability.

To summarize the above, it may be stated that the aim of an integral model is to provide a mechanism whereby experimental data may confidentally be interpolated over the range for which they are available. Differential models in addition seek to provide a capability whereby a limited amount of experimental data will provide the basis for extrapolation beyond the range for which the data is available. Within their common range of applicability, differential and integral models should produce results which closely agree with one another, and with available experimental data. The differential approach is inherently more flexible, and provides a more detailed picture of the processes occurring within the coil. With improvements in our understanding of these processes, and the increasing cost-effectiveness of available computing power, such models may in time provide an attractive alternative to the integral model to be described in the following.

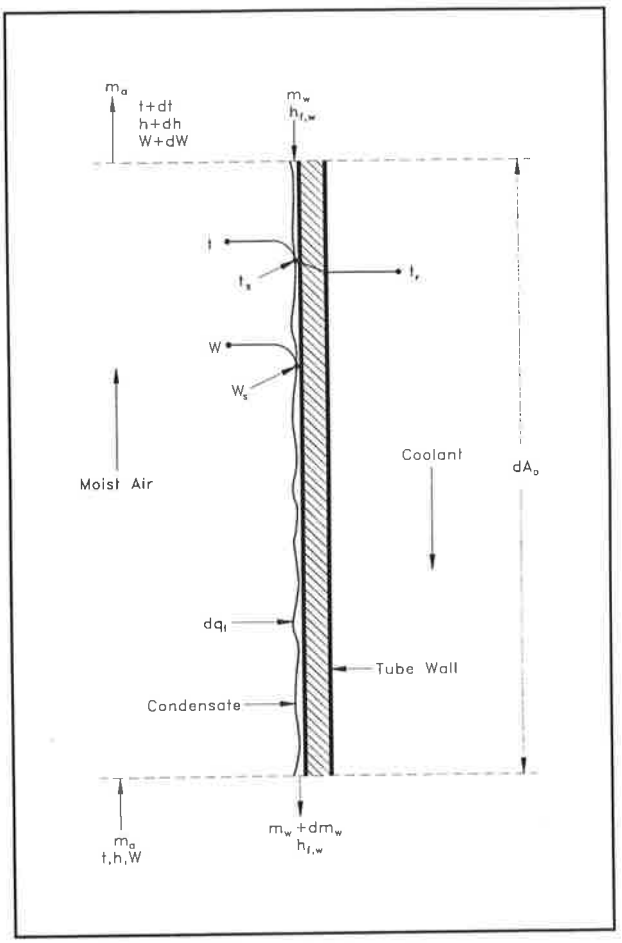


Figure 7.7. Heat and mass transfer processes across an incremental area of coil surface.

7.3.1. The Dual-Potential Model.

As a preliminary step in the derivation of the model, it is necessary to consider the bulk heat and mass transfer processes occurring across an incremental area of the surface of the coil. The driving force for heat and mass transfer between the humid air and the coolant is modelled as two potentials combined in series. The situation is shown schematically in figure 7.7. The driving forces involved are:

i. Heat and mass are transferred between a humid air stream and the wetted surface of the coil. Momentum and thermal boundary layers form at the inner surface of the tube, and in the airstream adjacent to the condensate layer. Under conditions to be defined below, the coil surface will be the site of active condensation of water vapour, which will form a layer of condensate as shown. The air in contact with

the condensate will be saturated at a temperature t_s, the humidity ratio and the enthalpy of the air saturated at this temperature being W_s and h_s respectively. In these circumstances, the total rate of heat transfer from the airstream to the coil surface is:

$$dq_t = -\dot{m}_a dh_a + \dot{m}_a dW_a h_{f,w} \quad . \tag{7.3.1}$$

The operative driving force in this instance is the enthalpy potential between the free stream and the air immediately adjacent to the layer of condensate, and the rate of heat transfer may be represented in terms of the driving force as:

$$dq_t = h_{c,ow} dA_o (t_a - t_s) + K_w dA_o (W_a - W_s)(h_{g,t} - h_{f,w}) \quad . \tag{7.3.2}$$

Introducing the Lewis number, as defined in equation (4.2.11), the above expression becomes

$$dq_t = \frac{h_{co,w} dA_o}{C_{p,a}} \left[C_{p,a}(t_a - t_s) + \frac{(W_a - W_s)(h_{g,t} - h_{f,w})}{Le} \right] \quad . \tag{7.3.3}$$

Invoking equation (4.1.24), for S.I. units the latent component of the enthalpy potential may be approximated as $2501 \cdot 8(W_a - W_s)$ Equation (7.3.3) may then be restated as

$$dq_t = \frac{h_{co,w} dA_o}{C_{p,a}} \left[(h_a - h_s) + \frac{(W_a - W_s)(h_{g,t} - h_{f,w} - 2501 \cdot 8 Le)}{Le} \right] \quad . \tag{7.3.4}$$

Threlkeld (1970) has shown that the last term in the bracket is typically small by comparison with the first, in which case this equation can be restated in the form introduced by McElgin and Wiley (1940):

$$dq_t \simeq \frac{h_{c,ow}}{C_{p,a}} (h_a - h_s) dA_o \tag{7.3.5}$$

where $h_{c,ow}$ should now, strictly speaking, be interpreted as a combined heat and mass transfer coefficient; coil rating techniques based on equation (7.3.5) will necessarily impose this interpretation on $h_{c,ow}$

ii. Heat is transferred through the surface of the condensate layer, through the fins and tube wall, and through the coolant-side boundary layer, where it is removed by convection. The rate of heat removed due to convection is:

$$dq_t = \dot{m}_w C_{p,w} dt_w \quad . \tag{7.3.6}$$

In this case the process involved is one of pure heat transfer, and the driving force is the difference between the temperature of the saturated air immediately adjacent to the condensate layer, and the mean temperature of the coolant. In terms of this driving force, the rate of heat transfer may be represented as:

$$dq_t = h_i \left[\frac{A_o}{A_i} \right] (t_s - t_w) dA_o \tag{7.3.7}$$

where h_i is a combined coefficient for heat transfer through the condensate layer, the tube walls and fins, and the coolant boundary layer, and t_w is the temperature of the coolant.

The potentials described in (i) and (ii) above are connected in series, and refer to the same heat flow. Thus, equations (7.3.5) and (7.3.7) may be equated and rearranged to provide a definition for the coil characteristic:

$$C \equiv \frac{t_s - t_w}{h_a - h_s} = \frac{h_{co,w}}{h_i C_{p,a}} \frac{A_i}{A_o} \quad . \qquad \textbf{(7.3.8)}$$

The coil characteristic provides the basis for a graphical method developed by Kusuda (1957) to determine the surface temperature of a given coil. Note that the analysis underlying the development of equation (7.3.8) is based on a differential element of the coil surface area. The heat transfer coefficients in this definition are thus *local* coefficients. If it can be assumed that these coefficients are essentially independent of the position in the heat exchanger, the coil characteristic may be calculated on the basis of *overall* heat transfer coefficients, and can subsequently used to estimate local coil surface temperatures corresponding to air on to and off from the coil. Experience with an extensive database of test data confirms that this is a valid assumption.

If the heat and mass transfer coefficients defined in the above equations can be predicted for a given heat exchanger configuration and arbitrary flow conditions, the above equations can be used as the basis for a differential coil performance model. Inevitably, the process of reducing experimental test data to obtain coil performance ratings will introduce further assumptions and approximations, which may critically influence the accuracy and validity of the model. The process of reducing experimental test data to obtain coil performance ratings will usually require assumptions and consequently a degree of approximation which is often poorly defined. The most widely used method of rating cooling coils is undoubtedly that prescribed by ARI Standard 410-81, which takes the above analysis as its starting point. The dual potential concept also forms the basis for an alternative scheme for rating cooling coils developed at the University of Adelaide (Sekhar, 1990), hereafter referred to as the AU method. Both of these methods will be discussed in some detail in the following sections. It is however pertinent first to extend the above analysis, and to consider its application to a number of problems involved in the analysis and design of air conditioning systems.

7.3.2. Definition of Coil Performance Problems.

In the selection and analysis of simple dehumidifying coils, two closely-related problems are of major importance. These are defined below.

FA	Given a set of air-on conditions, and a water inlet temperature and flow rate, find the coil surface area, and consequently a coil geometry, which will deliver a desired set of air-off conditions.

FP	Given a known coil configuration, an air-on condition and air flow rate, and a water inlet temperature and flow rate, find the air-off condition.

Conventional design techniques, which have relied on selection of a coil to deliver a specified performance at peak conditions only, have been mainly concerned with problem FA (see for instance, ASHRAE (1988)). The life-cycle design methodology to be developed in subsequent chapters is concerned with obtaining satisfactory operation across the entire operating spectrum of the coil, which may be critically dependent upon the circuiting employed. Thus the need to trial a wide range of candidate solutions for a number of operating conditions will commonly arise. The emphasis in the following is therefore on developing reliable algorithms for the solution of problem FP. Within the context of the current methodology, methods for solving problem FA potentially have some value in forming the basis for techniques for making the preliminary selection of possible candidate coils for a particular application. Derivation of algorithms for the latter problem is a straightforward consequence of the following. It will not however be considered directly.

7.3.3. Determination of Coil Surface Conditions.

The condition of the working fluids entering and leaving a cooling coil are as defined by the notation of figure 7.8. The circuiting shown provides an approximation to a counterflow arrangement. While it is possible to rearrange the circuiting to approximate parallel flow (Van Aken, 1993), this is infrequently done in practice, and the present study will consider the counterflow configuration only. For this case, local conditions through the coil will vary as shown in figure 7.9, which shows the variation in moist air temperature and enthalpy in the body of the air stream, and immediately adjacent to the coil surface, and the variation in temperature of the coolant. In accordance with the dual-potential model developed previously, the surface referred to is the surface of the tube and fins for the dry portion of the coil, and the outer surface of the condensate layer for the wet portion.

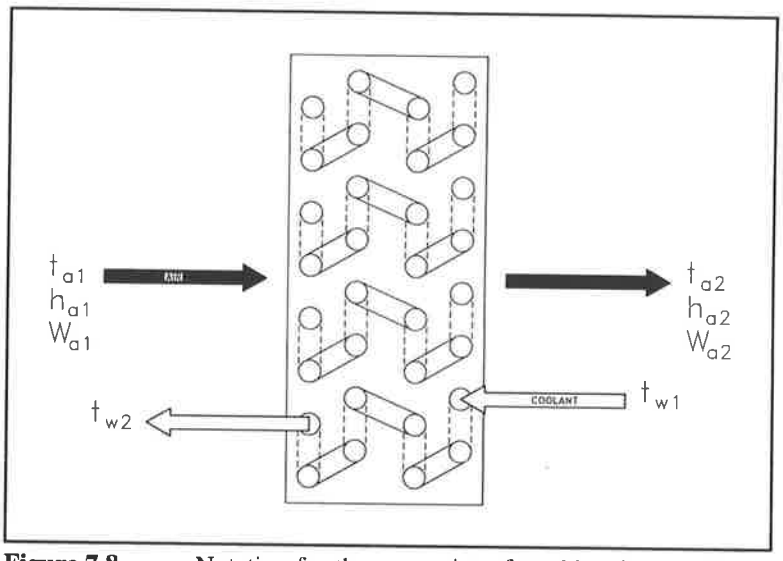


Figure 7.8. Notation for the properties of working fluids entering and leaving a typical cooling coil.

The air entering the coil will in most cases initially be in an unsaturated state. Condensation occurs where the local surface temperature falls below the dew-point temperature of the entering air. The point at which condensation starts to form is referred

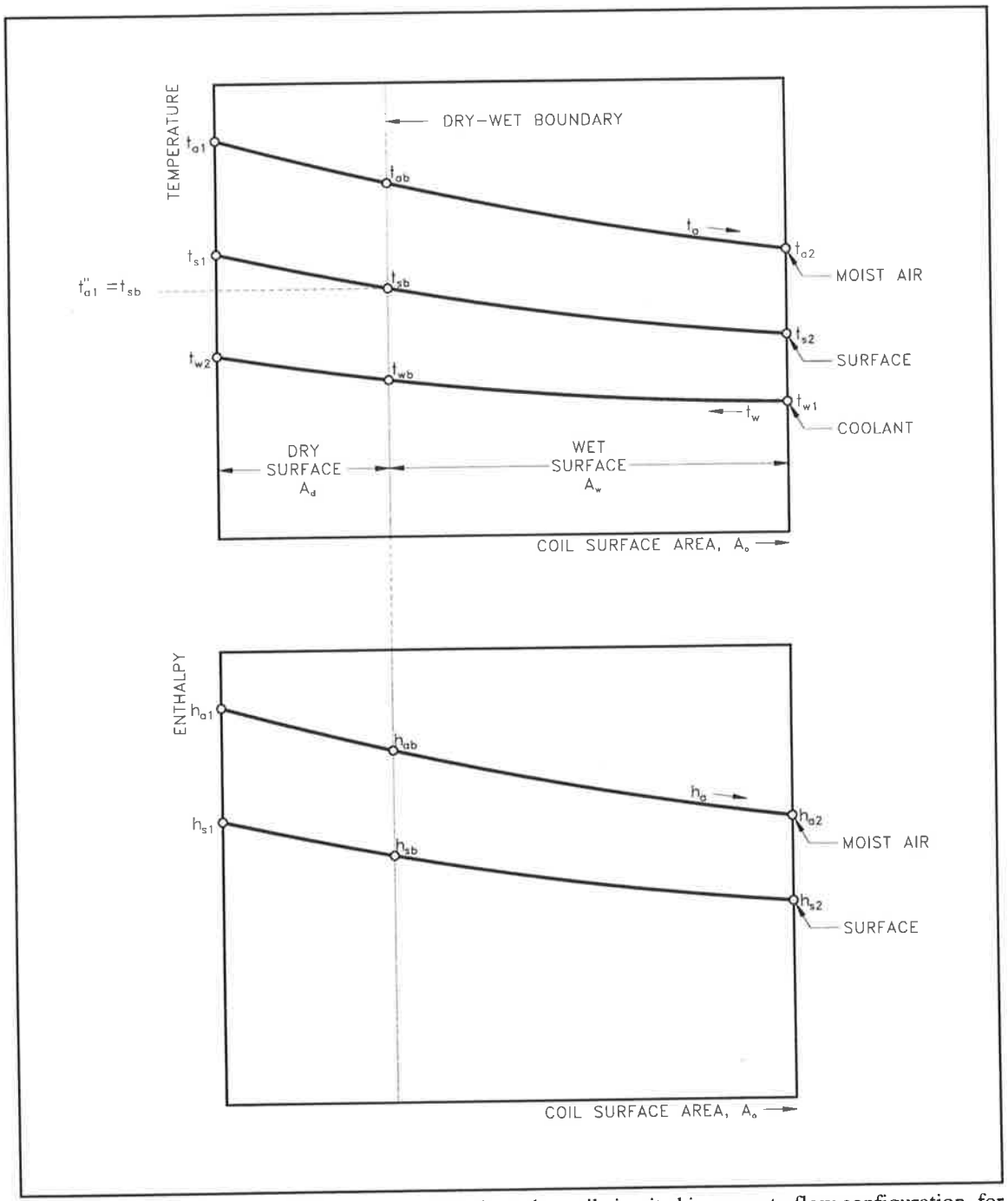


Figure 7.9. Variation of thermal properties through a coil circuited in a counterflow configuration, for the general case of operation with a partially dry surface.

to as the dry-wet boundary in figure 7.9. Those portions of the coil to the left of the boundary (upstream relative to the air flow) will operate dry, while those portions to the right (downstream) will operate wet.

Defining

$$y \equiv \frac{\dot{m}_a}{\dot{m}_w C_{p,w}} = \frac{t_{w2} - t_{w1}}{h_{a1} - h_{a2}} \qquad \textbf{(7.3.9)}$$

and performing an energy balance for the air and coolant flows over the dry portion of the coil leads to the following expression for enthalpy of the air stream corresponding to the dry-wet boundary:

$$h_{ab} = \frac{t_{a1}'' - t_{w2} + y h_{a1} + C h_{a1}''}{C + y} \quad . \qquad \textbf{(7.3.10)}$$

Given the state of the air and coolant entering and leaving the coil, this parameter provides a convenient characterisation of the operating regime for the coil:

a. If $h_{ab} \geq h_{a1}$ condensation occurs across the entire coil surface;
b. If $h_{a1} > h_{ab} > h_{a2}$ the surface is partially dry;
c. If $h_{ab} \leq h_{a2}$ the surface is complely dry.

The following conditions at the dry-wet boundary can also be determined:

$$t_{sb} = t_{a1}'' \qquad \textbf{(7.3.11a)}$$

$$t_{ab} = t_{a1} - \frac{(h_{a1} - h_{ab})}{C_{p,a}} \qquad \textbf{(7.3.11b)}$$

$$t_{wb} = t_{w2} - y C_{p,a}(t_{a1} - t_{ab}) \quad . \qquad \textbf{(7.3.11c)}$$

7.3.4. Determination of Coil-off Conditions.

Given the thermodynamic state and flow rate of moist air and coolant entering the coil, the model algorithm to be described below will predict the appropriate heat transfer coefficients, the state of coolant leaving the coil, and the enthalpy of the air leaving the coil. To complete the solution, one additional property of the air leaving the coil is required. For a fully-dry coil the humidity ration remains constant through the depth of the coil, and the leaving dry bulb temperature is readily found using equation (4.1.25). A means of deriving the leaving air dry bulb temperature for a wet or partially wet coil is derived in the following.

Consider a straight line on a psychrometric chart connecting the entering and leaving air conditions for a coil (figure 7.10). If projected, this line will meet the saturation line at a point which defines an equivalent surface enthalpy ($h_{\bar{s}}$), and an equivalent surface

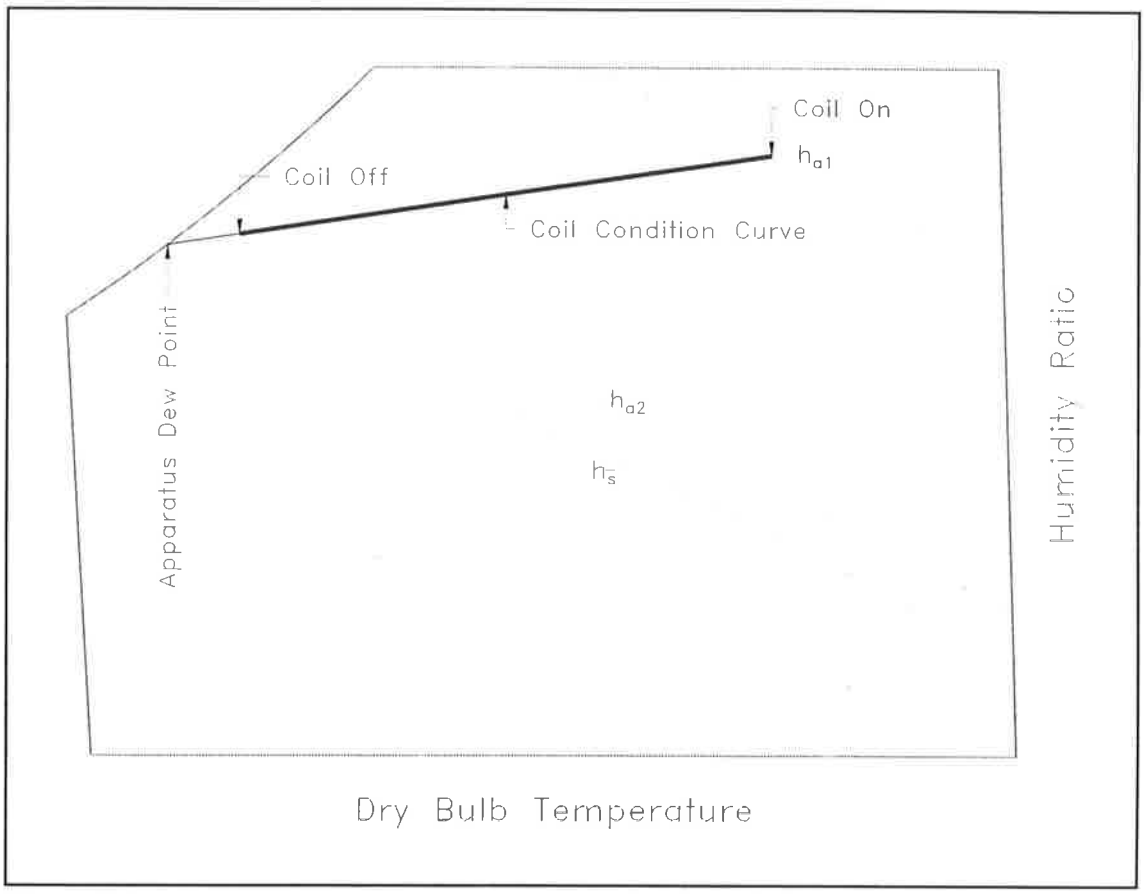


Figure 10.7. Ideal process line defining the equivalent surface enthalpy.

temperature ($t_{\bar{s}}$)[62] Neglecting the second term in equation (7.3.1), which will always be small, the air side heat transfer equation becomes

$$dq_t \approx -\dot{m}_a dh_a \quad . \tag{7.3.12}$$

For the ideal process line of figure 7.10, the surface temperature will remain constant, and equal to the effective surface temperature through the depth of the coil[63], and equation (7.3.5) may be rewritten as

$$dq_t = \frac{h_{c,ow}}{C_{p,a}}(h_a - h_{\bar{s}})\,dA_o \tag{7.3.13}$$

Equating (7.3.12) and (7.3.13), and integrating through the depth of the coil gives

---

62 Commonly known as the *apparatus dew point.*

63 This analysis approaches an exact representation for the case where the coolant temperature remains constant, and the process line approaches a straight line. Experience shows that it will however provide a good working approximation in most cases. Kusuda (1957) further considers the case of varying coolant temperature for a class of parallel-flow heat exchangers. See also Hill and Jeter (1991).

$$\int_1^2 \frac{dh_a}{(h - h_{\bar{s}})} = -\int \frac{h_{c,ow}}{C_{p,a}\dot{m}_a} dA_o \qquad \textbf{(7.3.14)}$$

and thus,

$$\ln\left[\frac{h_{a2} - h_{\bar{s}}}{h_{a1} - h_{\bar{s}}}\right] = -\frac{h_{c,ow}}{C_{p,a}\dot{m}_a} A_o \qquad \textbf{(7.3.15)}$$

or,

$$h_{\bar{s}} = h_{a1} - \frac{h_{a1} - h_{a2}}{1 - e^{-c}} \qquad \textbf{(7.3.16)}$$

where the heat transfer coefficient ***c*** is defined by

$$c \equiv \frac{h_{c,ow} A_o}{C_{p,a}\dot{m}_a} \quad . \qquad \textbf{(7.3.17)}$$

Considering now the sensible heat transfer between the moist airstream and the wetted surface, by analogy with equations (7.3.12) and (7.3.13), we can write:

$$dq_s = -\dot{m}_a C_{p,a} dt_a \qquad \textbf{(7.3.18)}$$

$$dq_s = h_{c,ow}(t_a - t_{\bar{s}}) dA_o \quad . \qquad \textbf{(7.3.19)}$$

Equating (7.3.18) and (7.3.19), and integrating as before gives

$$\ln\left[\frac{t_{a2} - t_{\bar{s}}}{t_{a1} - t_{\bar{s}}}\right] = -\frac{h_{c,ow}}{C_{p,a}\dot{m}_a} A_o \qquad \textbf{(7.3.20)}$$

or

$$t_{a2} = t_{\bar{s}} + (t_{a1} - t_{\bar{s}}) e^{-c} \qquad \textbf{(7.3.21)}$$

where c is as defined in equation (7.3.17). For the situation described at the beginning of this section, h_{a1}, h_{a2}, $h_{c,ow}$, t_{a1} and $\dot{m}_a$ are known and the coil-off dry bulb temperature can be found as follows:

a. Calculate $h_{\bar{s}}$ using equation (7.3.16).

b. Find the dry-bulb temperature $t_{\bar{s}}$ corresponding to the saturation enthalpy $h_{\bar{s}}$. The relationship connecting these two entities is pressure-dependent, and may be most expediently solved by finding the value of t_s for which

$$h_s(\tilde{t}) - h_{\bar{s}} = 0 \qquad \textbf{(7.3.22)}$$

(see section 7.3.6.1), where $h_s(\tilde{t})$ is a function which evaluates the saturation enthalpy for a given trial value of dry bulb temperature by:

i. Finding the corresponding humidity ratio using equation (4.1.10), and thus
ii. Evaluating enthalpy using equation (4.1.25).

The solution may be bracketed to within a fairly broad range using the identity $t_{wl} < t_s < t_{al}$.

c. Hence, solve equation (7.3.21) for the coil-off dry-bulb temperature.

7.3.5. A Model Algorithm for Problem FP.

In the most general case, we may consider the external surface of the coil to be covered by two non-overlapping regions, one fully wet and the other fully dry, these corresponding to the regions lying to the right and left respectively of the dry-wet boundary in figure 7.9. Thus, the total surface area of the coil

$$A_o = A_w + A_d \quad . \tag{7.3.23}$$

Similarly, we may partition the heat capacity for the coil into two components, corresponding to the wet and dry surface areas of the coil:

$$q_t = q_{tw} + q_{td} \quad . \tag{7.3.24}$$

Now we predicate the existence of algorithms to solve the following two problems:

AW	Giving the condition of the working fluids entering a coil, together with the cooling capacity of the fully-wet portion of the coil, find the corresponding fully-wet surface area.

AD	Given the condition of the working fluids entering a coil, together with the cooling capacity of the fully-dry portion of the coil, find the corresponding fully-dry surface area.

Algorithms to solve the above problems will be presented in sections 7.4 and 7.5; for the moment it suffices to assume that they exist.

We are now in a position to propose an algorithm to solve problem FP. Given an estimate of the cooling capacity for the coil and its component parts, $\tilde{q}_t = \tilde{q}_{tw} + \tilde{q}_{td}$, then, using the algorithms for problems AW and AD we can estimate the wet and dry surface areas for the coil, $\tilde{A}_w$ and $\tilde{A}_d$. For a given coil geometry, A_o is known. Thus, $\tilde{q}_t$ may be found by solving the homogeneous equation

$$f(\tilde{q}_t) \equiv A_o - \tilde{A}_d - \tilde{A}_w = 0 \tag{7.3.25}$$

for fixed condition of the working fluids at entry to the coil. In solving this equation, the condition of the working fluids leaving the coil will also be determined.

It is assumed that for $\tilde{q}_t > 0$, equation (7.3.25) has a unique solution[64], which may be solved for using a standard routine for finding the zero of a nonlinear function of one independent variable. It should be noted at the outset that zero-finding techniques such as the Newton-Raphson method, which require a knowledge of the first derivative of the function, are not viable in the present instance. Fortunately, efficient root-finding methods which rely solely on the ability to solve for $f(\tilde{q}_t)$ for some given $\tilde{q}_t$ are available. A procedure to return $f(\tilde{q}_t)$ given $\tilde{q}_t$ as an input parameter is required. The detailed structure of such a procedure will depend on the method selected to calculate the heat and mass transfer coefficients. In outline, the procedure may be implemented as shown in figure 7.11.

7.3.6. Numerical Implementation.

7.3.6.1. Root Finding.

Given an estimate of the coil capacity, a function structured according to the model algorithm of figure 7.11 will return the corresponding value of the objective function $f(\tilde{q}_t)$, which is defined by equation (7.3.25). Such a function may be specified as an argument to a suitable root-finding procedure, which will return an estimate of the solution to equation (7.3.25) to a degree of accuracy determined by a user-supplied tolerance, and the machine precision.

A wide range of such algorithms are available (see for instance, Press et al., 1988). Broadly speaking, these may be classified into two categories, according to whether the iterative technique used requires that the first derivative of the objective function be calculated at the trial points, or otherwise. As mentioned previously, it is not convenient to calculate the derivatives of the objective function in the present case, and an algorithm belonging to the second category must be used. The technique used is in fact an algorithm devised by Brent (1971), commonly referred to as zero. This is a hybrid method, based on the secant method, but using bisection to accelerate convergence where the basic secant method demonstrates poor convergence properties.

This algorithm is used extensively by the Zebra simulation package, and it is worth considering its attributes in a little more detail. Brent's algorithm is a guarded root-finding algorithm in the sense that the user is required to specify the interval (a,b) within which the solution is known to lie[65]. Provided the region (a,b) has been correctly specified,

---

64 No proof of this assumption will be offered; suffice it to say that experience with the algorithm has yet to indicate the existence of a set of conditions which contradict this assumption.

65 In many instances, including the one currently being considered, evaluation of the objective function is computationally expensive. The task of finding the bracketing interval (a, b) (see below) frequently involves the evaluation of the objective function, $f(a)$ and $f(b)$ at the limits of the bracketing interval. However, in the published versions of zero (Brent, 1971; Press et al., 1988),

```
{
    Calculate h_a2, by performing an energy balance on the air stream;        (Eqn. 7.3.12)
    Calculate t_w2, by performing an energy balance on the coolant;           (Eqn. 7.3.6)
    Estimate required thermal resistances and heat transfer coefficients;     (Sections 7.4-7.6)
    Calculate the coil characteristic C;                                      (Eqn. 7.3.7)
    Find the enthalpy of the air h_ab at the dry/wet boundary;                (Eqn. 7.3.10)

    if h_ab ≥ h_a1 then                                                       (Coil is fully wet)
    {
        q̃_tw ⇐ q̃_t;
        q̃_td ⇐ 0;
        Ã_w ⇐ Solution to problem AW;
        Ã_d ⇐ 0;
    }
    else                                                                      (Coil is partially or fully dry)
    {
        if h_ab ≤ h_a2 then                                                   (Coil is fully dry)
        {
            q̃_td ⇐ q̃_t;
            q̃_tw ⇐ 0;
        }
        else                                                                  (Coil is partially wet)
            q̃_td ⇐ ṁ_a(h_a1 - h_ab);
        if t_a1 ≤ t_a2 or t_ab ≤ t_wb then
            return ∞;                                                         (Section 7.3.6)
        Ã_d ⇐ Solution to problem AD;

        if h_ab > h_a2 then                                                   (Coil is partially wet)
        {
            q̃_tw ⇐ q̃_t - q̃_td;
            Ã_w ⇐ Solution to problem AW;
        }
        else
            Ã_w ⇐ 0;
    }

    Calculate t_a2;                                                           (Section 7.3.4)
    return Ã_w + Ã_d - A_o;
}
```

Figure 7.11. Outline of a procedure to calculate the difference between the calculated and actual total coil surface areas.

---

this information is discarded, and *f(a)* and *f(b)* are *re*-evaluated within the body of the function. Using the C++ overloading mechanism (Stroustrup, 1991), it is straightforward to provide an alternative function call permitting values of *f(a)* and *f(b)* to be passed as arguments, when known. This is a trivial modificationbut because of the excellent convergence properties of zero, the computational savings are frequently substantial.

convergence is guaranteed. The details of the iterative mechanism used by zero are particularly important insofar as one aspect of the algorithm of figure 7.11 is concerned. If an excessively large value of $\tilde{q}_t$ is passed as an argument to this procedure, a physically non-realizable situation may arise whereby a leaving coolant temperature in excess of the entering air temperature ($t_{w2} \geq t_{a1}$), or a coolant temperature at the wet/dry boundary in excess of the corresponding air temperature ($t_{wb} \geq t_{ab}$) may be predicted. In such circumstances a value of $f(\tilde{q}_t) = \infty$ is returned[66]. In the search for a bracketing interval, such points will (correctly) be identified as delimiting the interval on the high side. While the presence of such points within the bracketing interval would potentially pose problems for the basic secant method, the more sophisticated iterative mechanism used by zero quarantees that such points will be handled with little loss of computational efficiency.

```
{
        b ⇐ a;
        f_b ⇐ f_a ⇐ f(a);

        while (sign (f_a) = sign (f_b)) and f_b ≠ 0
        {
                b ⇐ b + δ;
                f_b ⇐ f(b);
                if δ < 0 then
                        Swap a and b, f_a and f_b;
                δ ⇐ δ × (-c);

        }

}
```

Figure 7.12. Logic for Algorithm BI1.

7.3.6.2. Finding a Bracketing Interval.

In many cases it will not be immediately obvious from the problem domain as to just how one should proceed to specify a bracketing interval for the solution of a univariate homogeneous function. In most instances however, an initial estimate of the solution can be made. Generally, the simple coils described here will be combined as the components of a more complex structure, as described in detail in chapter 8. Procedures for making an initial estimate of the capacity of a complex coil structure, and its component parts, are also described in that section. The present section considers the problem of systematically selecting a bracketing interval (a,b) for the solution of a homogeneous function, $f(x) = 0$, given some initial estimate $\tilde{x}$ of the solution.

Swift and Lindfield (1978) describe a simple algorithm for selecting a bracketing interval. In the following this is described as Algorithm BI1, and takes the following parameters as input:

a : An initial estimate, $a = \tilde{x}$ of the solution;
δ : An initial positive increment, $\delta(\tilde{x})$ in the search domain;
c : A constant factor, $c > 1$ ($1{\cdot}25 \leq c \leq 2$ would be typical);

---

66 In fact, the value DBL_MAX is returned. This is a constant defined in the standard C header file <limits.h> (Kernighan and Ritchie, 1988), which specifies the largest value of a double precision variable allowed by the architecture of the host machine.

$f(\tilde{x})$: A user-specified function for which a solution is sought.

The logic of Algorithm BI1 is as shown in figure 7.12. Upon completion, a search interval (a,b), such that $f(x) = 0$ for some $a \le x \le b$ is returned, together with the function values $f(a)$ and $f(b)$ at the limits of the search interval.

```
{
        f_a ⇐ f(a);
        if f_a = 0 then
        {
                b ⇐ a;
                f_b ⇐ f_a;
                return;
        }

        b ⇐ δ × a;
        f_b ⇐ f(b);
        if sign(f_a) ≠ sign(f_b) then
                return;

        if |f_b| < |f_a| then
                while (sign(f_a) = sign(f_b)) and f_b ≠ 0
                {
                        a ⇐ b;
                        f_a ⇐ f_b;
                        b ⇐ δ × b;
                        f_b ⇐ f(b);
                }
        else
                while (sign(f_a) = sign(f_b)) and f_a ≠ 0
                {
                        b ⇐ a;
                        f_b ⇐ f_a;
                        a ⇐ a/δ;
                        f_a ⇐ f(a);
                }
}
```

Figure 7.13. Logic for Algorithm BI2.

In using Algorithm BI1, a problem may occur if the search interval is broadened to examine values of $\tilde{x}$ for which feasible values of $f(\tilde{x})$ do not exist. In the context of the present problem, the behaviour of the algorithm illustrated in figure 7.11 and described in the preceding section guarantees that arbitrarily large values of the estimated coil capacity $\tilde{q}_t$, while not physically realizable, will define a usable upper limit to the search interval. Trial values of $\tilde{q}_t \le 0$ cannot be dealt with so easily. A straightforward modification to the algorithm will guarantee however, that such values will not be encountered. The logic for the modified algorithm BI2 is shown in figure 7.13. The input variables a and $f(x)$ are as described above for Algorithm BI1. In this instance, δ is a multiplicative constant with its value constrained such that $\delta > 1$.

The above algorithms are suitable for use in a region of the problem domain in which the objective function exhibits monotonic behaviour. If this is not the case, the search procedure must be modified to incorporate heuristic knowledge derived from the problem domain.

7.4. The ARI Method.

In section 7.3 a model algorithm to predict coil performance was developed. In order to obtain a closure for this model algorithm, it is necessary to provide algorithms to solve problems AW and AD, which are defined in section 7.3.5. These in turn entail the need to predict the various coefficients regulating air-side heat and mass transfer, and coolant-side heat transfer. ARI Standard 410-81, which is based on the dual-potential model described in the preceding section, prescribes a comprehensive methodology for rating cooling coils using chilled water or refrigerant as a coolant, and heating coils using hot water or steam as a heating fluid. The following discussion is restricted to the case of cooling coils using chilled water as a coolant. Extension to other working fluids is straightforward. The purpose of this section is to develop and critically review the foundations upon which this methodology is based, and its suitability for providing the required coefficients. The methodology is developed first for the case of dry coil surfaces, following which its extension to wet coil surfaces is considered.

7.4.1. Dry Coil Surfaces.

For a dry coil, or for the dry portion of a partially-wet coil, the rate of heat transfer is given by the equation

$$q_{td} = U_{o,d} A_d \Delta t_m \qquad \textbf{(7.4.1)}$$

where $U_{o,d}$ is an overall heat transfer coefficient for the dry portion of the surface and Δt_m is a mean temperature difference. For a heat exchanger operating in pure counterflow, the appropriate value for Δt_m is the *log-mean temperature difference* (LMTD), defined as

$$\Delta t_m = \frac{(t_{a1} - t_{w2}) - (t_{a2} - t_{w1})}{\ln\left[\dfrac{t_{a1} - t_{w2}}{t_{a2} - t_{w1}}\right]} \qquad \textbf{(7.4.2)}$$

This expression is only strictly accurate for the case of pure counterflow, smaller values being encountered for other configurations, for which it is customary to relate the true mean temperature difference to the LMTD using a correction factor

$$\Delta t_m = F \times LMTD \qquad \textbf{(7.4.3)}$$

where $0 < F \leq 1$. Values of F have been calculated by Bowman et al. (1940)[67] for a range of heat exchanger geometries. Cooling coils employed in the air conditioning industry are invariably of the cross-flow type, and are usually multipass heat exchangers circuited in a manner which approximates counterflow, in that the feeds are located at the downstream side of the coil with respect to the air stream, the outlet manifolds being located on the upstream side. Comparison with published curves shows that for such a configuration, F may be assumed to be unity with negligible error. Thus, in the remainder of this analysis, we will use the value of Δt_m defined by equation (7.4.2). This conforms with the practice specified by ARI Standard 410-81.

---

[67] A selection of these are reproduced in Threlkeld (1970).

Now, $U_{o,d} = 1/R$, where R is the overall resistance to heat transfer between the air and the coolant. This may be expressed as the sum of the component resistances, combined in series:

$$R = R_{aD} + R_m + R_w \tag{7.4.4}$$

where

R_{aD} is the dry air-side thermal resistance,
R_m is the total metal thermal resistance, and
$R_w = B/h_i$ is the coolant-side thermal resistance.

For the last-mentioned quantity, ARI Standard 410-81 specifies that the following relationship due to McAdams (1954) should be used:

$$h_i = \frac{4209{\cdot}15(1{\cdot}352 + 0{\cdot}0198 t_{wm}) U_w^{0{\cdot}8}}{D_i^{0{\cdot}2}} \tag{7.4.5}$$

where t_{wm} is the mean temperature of the water in the tubes, which may be evaluated as $t_{wm} = 0{\cdot}5(t_{w1} + t_{wb})$ for the wet section of the coil surface, and $t_{wm} = 0{\cdot}5(t_{wb} + t_{w2})$ for the dry section. This expression, together with some alternatives, will be critically examined in section 7.6. The remaining two resistances are considered in the following.

7.4.2. Thermal Resistances for Dry Coil Operation.

The total metal thermal resistance comprises the series sum of the resistances to heat transfer offered by the tube walls and the fins,

$$R_m = R_t + R_f \tag{7.4.6}$$

The former quantity may readily be calculated as

$$R_t = \frac{B D_i}{2 k_t} \ln\left(\frac{D_o}{D_i}\right) \tag{7.4.7}$$

in which k_t is the thermal conductivity of the tube wall. We note that this quantity is frequently negligible, but retain it in the analysis for completeness.

An expression relating the fin thermal resistance to the dry air-side resistance can be derived by defining the fin efficiency ϕ as

$$\phi = \frac{t_{f,m} - t_a}{t_{f,b} - t_a} = \frac{\Delta t_{f,m}}{\Delta t_{f,b}} \tag{7.4.8}$$

where $t_{f,b}$ is the temperature at the base of the fin (which, ignoring contact resistance, is equal to the temperature at the outer wall of the tube), $t_{f,m}$ is a mean temperature for the fin, and t_a is the temperature of the air in contact with the fin. The fin efficiency accounts for the variation in temperature along the fin, permitting the heat transferred through the

fin to be calculated on the basis of a known tube wall temperature. Following Threlkeld (1970) it can be shown that

$$R_f = \left[\frac{1 - \eta}{\eta}\right] R_{aD} \tag{7.4.9}$$

where η is a more relevant fin effectiveness, defined as

$$\eta = \frac{\phi A_s + A_p}{A_o} \tag{7.4.10}$$

For a given fin geometry, the fin efficiency can be calculated analytically, provided certain simplifying assumptions are made. ARI Standard 410-81 stipulates that fin efficiency *shall* be calculated using results developed by Gardner (1945), who performed an elegant series of analyses covering a range of fin geometries. Of most immediate relevance to the current study is Gardner's expression for the efficiency of an annular fin of uniform thickness,

$$\phi = \frac{2}{U_b\left[1 - \left(U_e/U_b\right)^2\right]}\left[\frac{I_1(U_b) - \beta K_1(U_b)}{I_0(U_b) - \beta K_0(U_b)}\right] \tag{7.4.11}$$

where I_0, I_1, K_0 and K_1 are modified Bessel functions[68], and

$$\beta = \frac{I_1(U_e)}{K_1(U_e)} \tag{7.4.12}$$

$$U_b = \frac{w}{\left(x_e/x_b - 1\right)}\sqrt{\frac{2}{R_{aD} k_f Y_f}} \tag{7.4.13}$$

$$U_e = U_b\left(\frac{x_e}{x_b}\right) \quad . \tag{7.4.14}$$

In the above, x_e is the radius of the annular fin, and w is its height, defined as

$$w = x_e - x_b \quad . \tag{7.4.15}$$

Gardner presents no analysis for coils employing rectangular fins, and indeed such geometries are not readily amenable to an analytical treatment. The efficiency of such fins can be calculated using the above expression by considering an annular fin coil having the same secondary area as the plate fin coil. Using the segmentation method of Carrier and Anderson (1944; see figure 7.13), the appropriate equivalent fin radius is found to be

---

[68] Polynomial approximations, such as those presented by Press et al. (1988) are adequate for calculating these functions in the present case.

$$x_e = \sqrt{\frac{S_f S_r}{\pi}} \quad . \tag{7.4.16}$$

The assumptions upon which Gardner's analysis are based are, to quote from Gardner's work:

"1. The heat flow and temperature distribution throughout the fin are independent of time, i.e., the heat flow is steady.
2. The fin material is homogeneous and isotropic.
3. There are no heat sources in the fin itself.
4. The heat flow to or from the fin surface at ant point is directly proportional to the temperature difference between the surface at that point and the surrounding fluid.
5. The thermal conductivity of the fin is constant.
6. The heat-transfer coefficient is the same all over the fin surface.
7. The temperature of the surrounding fluid is uniform.
8. The temperature of the base of the fin is uniform.
9. The fin thickness is so small compared to its height that temperature gradients normal to the surface may be neglected.
10. The heat transferred through the outermost edge of the fin is negligible compared to that passing through the sides."

It should be emphasized that Gardner's analysis is *exact*, within the constraints of the above assumptions. Most of these can be shown to be reasonable. The validity of assumption 7 must be questioned; the very nature of the process means that the air temperature is varying significantly in the direction of flow. Assumption 6 is even more seriously in error, as has been demonstrated by recent investigations of the detailed heat transfer and fluid dynamic processes occurring within a plate-fin heat exchanger (Gilbert, 1987; see section 7.2). These show significant variations in the heat transfer coefficient across the surface of the fin. Herein lies one of the major flaws in the method proposed by ARI Standard 410-81.

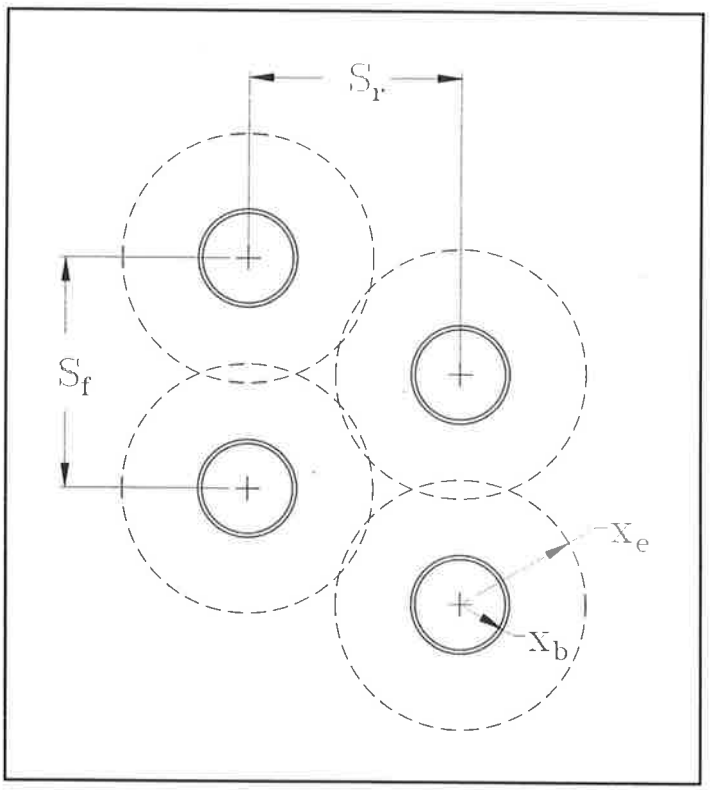


Figure 7.13. Segmentation method for treating a plate fin in terms of an annular fin of equal area.

Substituting for R_f from equation 7.4.9, we can write

$$R_{aD} + R_m = \frac{R_{aD}}{\eta} + R_t \qquad \textbf{(7.4.17)}$$

ARI Standard 410-81 stipulates that plots of R_{aD} versus $V_{a,std}$[69] shall be provided as part of the rating process, thus defining a functional relationship,

$$R_{aD} = f(V_{a,std}) \qquad \textbf{(7.4.18)}$$

We can now propose a procedure to solve problem AD, section 7.3.5, based on the ARI method:

1. In the following order, determine:
   i. R_{aD} using the functional relationship of equation (7.4.18);
   ii. Fin efficiency using Gardner's method (equations 7.4.11-14) in conjunction with an equivalent fin radius (equation 7.4.16);
   iii. Fin effectiveness using equation (7.4.10);
   iv. R_t using equation (7.4.7);
   v. R_w using equation (7.4.5), or equivalent.

   Hence, calculate the overall thermal resistance R, using equations (7.4.4) and (7.4.17).

2. Calculate the log-mean temperature difference, using equation (7.4.2).

3. Hence, estimate the dry surface area (equation 7.4.1).

This procedure is suitable for use within the iterative loop described in section 7.3.5[70].

7.4.3. Wet Coil Surfaces.

For a wet coil, or for the wet portion of a partially-dry coil, the potential driving the heat transfer process is the difference in enthalpy between the air stream and the saturated air in contact with the wet surface, and the rate of heat transfer is given by the equation

$$q_{tw} = \frac{A_w \Delta h_m}{C_{p,a} R_{aW}} \qquad \textbf{(7.4.19)}$$

where R_{aW} is the wet air-side thermal resistance, and Δh_m is a mean enthalpy difference. With the same qualifications stated when condensing the log-mean temperature difference (equation 7.4.2), we may use a *log-mean enthalpy difference* here such that

---

69 $V_{a,std}$ is the coil face velocity, based on ASHRAE standard conditions (see equation 5.1.2).

70 In initializing the iterative loop, an initial estimate of the temperature of the air leaving the coil (t_{a2}) must be provided, so that the LMTD can be calculated the first time through the loop. See section 7.8.3 for a description of the initialization procedure.

$$\Delta h_m = \frac{(h_{ab} - h_{s1}) - (h_{a2} - h_{s2})}{\ln\left[\frac{h_{ab} - h_{s1}}{h_{a2} - h_{s2}}\right]} \qquad \textbf{(7.4.20)}$$

where h_{ab} is as defined by equation (7.3.10)[71]. Now, for an element of wet surface area, equation (7.4.19) may be recast in a form analagous to equation (7.4.5):

$$dq_t = \frac{h_a - h_s}{C_{p,a}R_{aW}}dA_o \qquad \textbf{(7.4.21)}$$

while the corresponding equation for the heat transfer between the wet surface and the coolant may be expressed in a form analagous to equation (7.4.7):

$$dq_t = \frac{t_s - t_w}{R_{mW} + R_w}dA_o \quad . \qquad \textbf{(7.4.22)}$$

Equating (7.4.21) and (7.4.22) and rearranging provides an alternative expression for the coil characteristic, originally defined by equation (7.3.8). Thus,

$$C = \frac{R_{mW} + R_w}{C_{p,a}R_{aW}} \qquad \textbf{(7.4.23)}$$

which may be evaluated using known values for the various thermal resistances (see below). The surface enthalpies required to find the log-mean enthalpy difference may now be calculated. From the definition of the coil characteristic, at the upstream side of the coil,

$$\frac{t_{s1} - t_{w2}}{h_{a1} - h_{s1}} = C \quad . \qquad \textbf{(7.4.24)}$$

A simple functional relationship exists between h_{s1} and t_{s1} (see section 7.3.4). Thus, we seek to find the value of the entering surface temperature for which

$$\tilde{t}_s - t_{w2} - C[h_{a1} - h_s(\tilde{t}_s)] = 0 \qquad \textbf{(7.4.25)}$$

where $t_{w2} < t_{s1} < t_{a1}$. This may readily be solved using the technique of section (7.3.6.1). Similarly, the surface enthalpy at the downstream side of the coil may be found by solving

$$\frac{t_{s2} - t_{w1}}{h_{a2} - h_{s2}} = C \quad . \qquad \textbf{(7.4.26)}$$

---

[71] For fully-wet coils h_{a1} replaces h_{ab} in equation (7.3.20).

7.4.4. Thermal Resistances for Wet Coil Operation.

For wet coil operation, R_t, R_w and ϕ are evaluated as for a dry surface. The ARI Standard 410-81 rating procedure states that plots of R_{aW} as a function of $V_{a,\,std}$ shall be provided for wet coil operation (cf equation 7.4.18). We define an effective air-side heat transfer coefficient f_a, such that for a dry coil $f_{aD} = 1/R_{aD}$. The ARI methodology for wet coils derives from the work of Brown (1954), who showed that with suitable simplifying assumptions, f_a is directly proportional to m'', where

$$m'' \equiv \frac{dh_s}{dt_s} \tag{7.4.27}$$

and its extension to wet coil theory by Ware and Hacha (1960; see Threlkeld, 1970 for a detailed treatment). For a wet coil,

$$f_{aW} = \frac{1}{R_{aW}} \frac{m''}{C_{p,a}} \tag{7.4.28}$$

and equation (7.4.9) can be restated as

$$R_f = \left[\frac{1 - \eta}{\eta}\right] R_{aW} \frac{C_{p,a}}{m''} \quad . \tag{7.4.29}$$

ARI Standard 410-81 provides a chart showing m'' as a function of t_s for a range of air pressures. Within a computational scheme, m'' will be more conveniently found using a numerical method based on the relationships developed in chapter 4[72]. The appropriate surface temperature for determining m'' is obtained by solving

$$\frac{t_{sm} - t_{wm}}{h_{am} - h_{sm}} = C \tag{7.4.30}$$

where $t_{wm} = 0{\cdot}5(t_{wb} + t_{wl})$ as previously, and $h_{am} = 0{\cdot}5(h_{ab} + h_{a2})$.

A procedure to solve problem AW, section (7.3.5), within the context of the iterative solution procedure for problem FP follows:

1. In the following order, determine:
   - i. R_{aD} and R_{aW} using the functional relationship of equation (7.4.18), and its wet-coil equivalent;
   - ii. Fin efficiency using Gardner's method (equations 7.4.11-14) in conjunction with an equivalent fin radius (equation 7.4.16);
   - iii. Fin effectiveness using equation (7.4.10);
   - iv. R_t using equation (7.4.7);
   - v. R_w using equation (7.4.5), or equivalent;
   - vi. m'' using the mean surface temperature of equation (7.4.30);

---

[72] Since the ARI method for *dry* coils only is used in the following work, an alternative procedure being used for wet coils, this technique will not be developed further here.

vii. R_{mW} using R_t already calculated, together with R_f (equation 7.4.29).

2. Calculate the entering and leaving surface enthalpy (7.4.24 and 7.4.26). Hence, find the log-mean enthalpy difference using equation (7.4.20).

3. Hence, estimate the wet surface area (equation 7.4.19).

7.4.5. Coil Rating Procedures.

ARI Standard 410-81 prescribes a procedure for deriving coil rating information from experimental test data. The steps in the rating process follow logically from the theory developed in the preceding sections[73]. For a completely *dry* coil surface, these are as follows:

1. The (sensible) load on the coil is calculated from the measured air flow rate and the difference in enthalpy between the air entering and leaving the coil.

2. The log-mean temperature difference (equation 7.4.2) is calculated using the measured entering and leaving temperatures of the working fluids.

3. The overall heat transfer coefficient is found by substituting into equation (7.4.1); the overall resistance ($R = 1/U_{o,d}$; equation 7.4.4) follows.

4. The coolant-side thermal resistance (R_w) is found using equation (7.4.5), or an equivalent, with measured values of the coolant properties and flow rates. The tube-wall thermal resistance (R_t) is obtained from equation (7.4.7). The dry air-side thermal resistance is found by solving the homogeneous equation

$$R - \frac{R_{aD}}{\eta} - R_t - R_w = 0 \qquad \textbf{(7.4.31)}$$

in which η, defined by equation (7.4.10), is a function of R_{aD}. This equation may be solved by the methods of section 6.3, setting $\tilde{R}_{aD} \approx R - R_w$ as a first approximation.

5. R_{aD} is plotted versus $V_{a,std}$ for a series of tests covering the operational range of the coil, and the functional relationship of equation (7.4.18) derived.

The ARI procedure for a *wet* coil presupposes that a test series has previously been conducted for a dry coil having the same geometry. The steps in the rating process then follow:

---

[73] The following development, which is based on the work of Sekhar (1990), describes an iterative procedure suitable for implementation on a computer. This differs in detail from the procedure prescribed by ARI Standard 410-81, which is intended for hand calculation.

1. The (total) load on the coil is calculated from the measured air flow rate, and the difference in the enthalpy of the air entering and leaving the coil.

2. Using the dry coil rating information, the value of R_{aD} corresponding to the face velocity measured in the wet coil surface test is found. The fin efficiency is then calculated using equations (7.4.11-7.4.15).

3. R_t and R_w are calculated as previously, while R_f is found using equation (7.4.9). The total dry surface metal thermal resistance, R_{mD} follows.

4. The slope of the saturated air temperature versus enthalpy curve (m'') is evaluated at an appropriate mean surface temperature, which is found as follows:

   i. An approximation to the coil characteristic is found, using the dry coil thermal resistances,

$$C \approx \frac{R_{mD} + R_w}{C_{p,a} R_{aD}} \qquad \textbf{(7.4.32)}$$

   ii. The mean enthalpy of the air flowing through the coil is found, using the measured enthalpies of the entering and leaving air.

   iii. Equation (7.4.30) is solved for t_{sm}, using the approximate value of C.

5. The wet air-side thermal resistance is found by rearranging equation (7.4.19), thus

$$f(\tilde{R}_{aW}) = A_o - \frac{q_{tw} C_{p,a} \tilde{R}_{aW}}{\Delta h_m} = 0 \qquad \textbf{(7.4.33)}$$

   and solving using the techniques of section 6.3, with $\tilde{R}_{aW} \approx R_{aD}$ as a first approximation. The steps in evaluating $f(\tilde{R}_{aW})$ are as follows:

   i. R_f is evaluated using equation (7.4.29); R_{mW} follows.
   ii. The coil characteristic is found using equation (7.4.23).
   iii. Equations (7.4.24) and (7.4.26) are solved to find the entering and leaving surface enthalpies. The log-mean enthalpy difference (equation 7.4.30) may thus be evaluated,
   iv. $f(\tilde{R}_{aW})$ is thus evaluated.

5. R_{aW} is plotted versus $V_{a,std}$, and a functional relationship derived, as for a dry coil.

7.4.6. Summary.

The methodology for rating cooling coils and predicting coil performance proposed by ARI Standard 410-81 is based on the work of a number of researchers, some of which dates back to the 1930s. The adequacy of the assumptions underlying this methodology has been questioned by several authors, including most notably McQuiston (1975) and Sekhar et al. (1988).

The concerns expressed cover a number of aspects of the analysis. Those associated with the fin efficiency have already been described. Nevertheless, this rating method does provide adequate predictions for dry coil operation.

More serious reservations exist with respect to its adequacy for wet coil surfaces. A major problem identified by Sekhar et al. (1988) lies in the fact that the analysis partially combines the driving forces with the transfer coefficients, which is based on the explicit assumption that the coil characteristic is constant at different points along a line of constant entering enthalpy for a given face velocity, an assertion which is manifestly not supported by experiment. Furthermore, the use of dry fin efficiency for wet coil prediction implies (see the analysis of Threlkeld, 1970) that the fins are covered with a uniform film of condensate. This assertion is also challenged by recent experimental work conducted at the University of Adelaide (Luxton and Shaw, 1991), and by the earlier work of McQuiston (1976). The validity of this assumption had been questioned in a previous paper by McQuiston (1975)[74], which provided an alternative derivation for fin efficiency which accounts for mass transfer, and reduces to the solution given by Gardner for the case of dry operation. The basic inadequacies of the fin efficiency approach still remain, however. The following section describes a methodology for rating wet coils (the AU method) which does not rely on fin efficiency.

The strategy adopted in this work is to accept the ARI method of rating dry coils, in the absence of an alternative, while using the AU method in preference to the ARI method for predicting wet coil areas. Section 7.8 provides further details of the procedures for deriving rating curves from experimental data for dry coils using the ARI method; for wet coils the reader is referred to the work of Sekhar (1990), who performed a detailed comparison of the ARI method and the AU method for rating wet coils.

7.5. The AU Method.

The AU method for rating cooling coils, like the ARI method, is based on the dual-potential model described in section 7.3. From that point however, the two methods are fundamentally different. The main feature which distinguishes the AU method from the ARI method is that the questionable concept of fin efficiency, which is central to the data reduction methodology of ARI Standard 410-81, is rejected in favour of a more direct method of estimating the various heat transfer coefficients involved. The principles upon which the AU method are based, together with details of the implementation of the method as a means of rating and predicting coil performance, are described in the following. The discussion draws extensively upon the work of Sekhar (1990; Sekhar et al., 1991b).

---

[74] It is stated that "water-film thickness on a fin ... is a very nebulous quantity even if it exists."

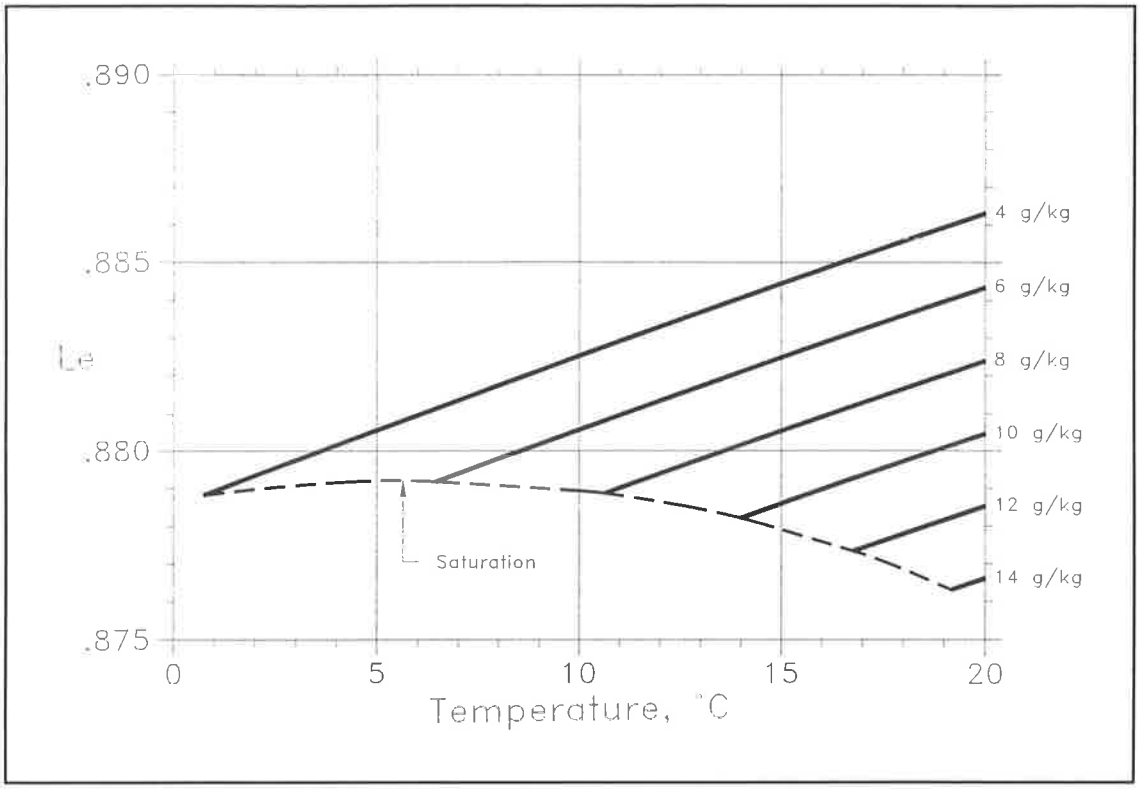


Figure 7.14. Lewis number for moist air as a function of dry-bulb temperature and humidity ratio.

7.5.1. Determination of the Coil Characteristic.[75]

The coil characteristic, defined by equation (7.3.8), establishes a unique relationship between the surface and free-stream air and coolant properties for a particular set of coil operating conditions. To determine the coil characteristic for a given coil test directly from its definition requires measurement of the surface temperature together with the associated free-stream conditions at some point in the coil. Measurement of the surface temperature would be error-prone, if possible. However, the free-stream conditions of the working fluids, together with their mass flow rates, can be accurately measured. The AU method proposes an iterative method for determining the coil characteristic, given an initial estimate.

The iterative technique is based on the observation that the heat and mass transfer coefficients are related by the Lewis number (equation 4.2.11), defined in the terminology appropriate to this problem as

---

[75] Sekhar's discussion refers alternatively to the tie-line slope (*TLS*), defined as

$$TLS = -\frac{1}{C}$$

This term derives from the earlier graphical techniques of Kusuda (1957) and Shaw (1979). The coil characteristic, which is far more commonly used in the literature, is used exclusively in the current work.

$$Le \equiv \frac{h_{c,ow}}{K_w C_{p,a}} \quad . \tag{7.5.1}$$

The Lewis number may be calculated from the fluid properties using the rightmost expression in equation (4.2.11), with $c = ⅓$. The Lewis number for moist air is a weak function of dry-bulb temperature and humidity ratio, and is close to 0·88 over the range of conditions of interest in the present case, as shown in figure 7.14. Now, the heat transfer coefficients which we seek to determine are *overall* values for the coil, which are defined by the relationships

$$q_s = h_{c,ow} A_o \Delta t_m \tag{7.5.2}$$

and

$$q_l = K_w A_o \Delta W_m h_{fg} \tag{7.5.3}$$

in which the sensible and latent coil loads can be determined from the experimental data (see section 7.8), h_{fg} is the latent heat of vapourization of water, and Δt_m and ΔW_m are mean values for the temperature and humidity ratio differential respectively between the cold coil surface and the free-stream conditions. The appropriate values to use for the latter are the *log-mean temperature difference* (LMTD) and the *log-mean humidity ratio difference* (LMWD), which may be found from

$$LMTD = \frac{(t_{a1} - t_{s1}) - (t_{a2} - t_{s2})}{\ln\left[\dfrac{t_{a1} - t_{s1}}{t_{a2} - t_{s2}}\right]} \tag{7.5.4}$$

and

$$LMWD = \frac{(W_{a1} - W_{s1}) - (W_{a2} - W_{s2})}{\ln\left[\dfrac{W_{a1} - W_{s1}}{W_{a2} - W_{s2}}\right]} \tag{7.5.5}$$

respectively. In the above the subscripts *1* and *2* refer to the air entering and leaving the coil respectively. W_s is the saturation humidity ratio corresponding to the coil surface temperature. The surface properties required to evaluate the *LMTD* and the *LMWD* may readily be calculated using equations (7.4.24-7.4.26) if we know the coil characteristic; the discussion accompanying those equations provides further details.

The AU method of determining the coil characteristic reduces then to one of solving the equation

$$f(\tilde{C}) \equiv Le - \tilde{Le}(\tilde{C}) = 0 \tag{7.5.6}$$

where $\tilde{C}$ is a trial value for the coil characteristic, and $\tilde{Le}$ is an estimate for the Lewis number, found by substituting the heat and mass transfer coefficients, calculated on the basis of the resulting surface property estimates, into equation (7.5.1). The actual Lewis number is calculated as

$$Le = \left[\frac{Sc}{Pr}\right]^{1-c} \quad . \tag{7.5.7}$$

This, and the fluid properties required in equations (7.5.1) and (7.5.3) are evaluated at the psychrometric point

$$t_m = \frac{t_{s1} + t_{s2} + LMTD}{2} \tag{7.5.8a}$$

$$W_m = \frac{W_{s1} + W_{s2} + LMWD}{2} \quad . \tag{7.5.8b}$$

Input:
$\tilde{C}$: *Estimated value for the coil characteristic.*

{

t_{s1}, h_{s1}, W_{s1} ⇐ *Values determined by solving equation (7.4.24) for* $\tilde{C}$;	(Use **zero**)
t_{s2}, h_{s2}, W_{s2} ⇐ *Values determined by solving equation (7.4.26) for* $\tilde{C}$;	(Use **zero**)
LMTD ⇐ Log-mean temperature difference;	(Eqn. 7.5.4)
LMWD ⇐ Log-mean humidity ratio difference;	(Eqn. 7.5.5)
Calculate t_m, W_m;	(Eqn. 7.5.8a, b)
Evaluate Le at (t_m, W_m);	(Eqn. 7.5.7)
Calculate $h_{c,ow}$ *appropriate to LMTD;*	(Eqn. 7.5.2)
Calculate K_w *appropriate to LMWD;*	(Eqn. 7.5.3)
$\tilde{Le}$ ⇐ *Lewis number based on estimates of* $h_{c,ow}$, K_w;	(Eqn. 7.5.1)

$f(\tilde{C})$ ⇐ $Le - \tilde{Le}$;

}

Returns: $f(\tilde{C})$

Figure 7.15. Logic for algorithm **LC1**.

We refer to the algorithm to evaluate $f(\tilde{C})$ as algorithm **LC1**. The logic is illustrated in figure 7.15. Equation (7.5.6) can be solved using **zero** (section 7.3.6.1). Several options are available for finding a suitable bracketing interval. In the present work we make an inital estimate of $\tilde{C} = 1$, and use algorithm **BI2** to establish the bracketing interval.

Refering to equations (7.5.2) and (7.5.3), it will be noted that the measured values of the heat and mass transfer coefficients will depend on the external surface area to which they refer. It has long been known (Guillory and McQuiston, 1973; Tree and Helmer, 1976) that the presence of condensate increases the rate of heat transfer over that pertaining to dry surfaces. Careful measurements by Schroeder-Lanz (1989) have shown that *droplike* condensation increases the heat transfer area of a surface by 30 to 60% over its dry area. It might seem reasonable therefore to augment the surface area in equations (7.5.2) and (7.5.3) to account for the presence of condensate. While this procedure would arguably be

strictly correct, assuming the wet surface area could be accurately determined for a given experiment, it should be noted that it is the *ratio* of the heat and mass transfer coefficients which determine the Lewis number (equation 7.5.1). In other words, convergence of the AU method for rating coils is insensitive to the exact external surface area presented for heat transfer. The heat and mass transfer coefficients may thus be unambiguously defined by referencing them to the dry surface area, as will be done in this work.

7.5.2. Tube-side Resistances.

The heat transfer coefficient h_i, defined in a differential sense by equation (7.3.7), is a combined coefficient for heat transfer through the condensate layer, the tube walls and fins, and the coolant boundary layer. An analogous *overall* coefficient is defined by the sensible heat transfer equation

$$q_t = h_i A_i \Delta t_m \qquad \textbf{(7.5.9)}$$

which determines the value of h_i with Δt_m calculated using equation (7.5.4), and q_t extracted from experimental measurements.

The heat transfer coefficient can alternatively be characterized by a thermal resistance

$$R_i = \frac{B}{h_i} \qquad \textbf{(7.5.10)}$$

which combines the various component resistances in series:

$$R_i = R_w + (R_m + R_{ow}) \qquad \textbf{(7.5.11)}$$

where

$R_w = B/f_w$ is the coolant-side thermal resistance,
R_m is the total metal thermal resistance, and
R_{ow} is the resistance of the condensate layer.

Expressions for evaluating the coolant-side thermal resistance are presented in section 7.6. The temperature at which the coolant properties should be evaluated are as described in the discussion following equation (7.4.5). The combined thermal resistance for the metal and condensate may then be determined as

$$(R_m + R_{ow}) = R_i - R_w \quad . \qquad \textbf{(7.5.12)}$$

The procedure described above should be compared with the procedures specified by ARI Standard 410-81. Refer specifically to equation (7.4.4) and its wet-surface equivalent. The most notable feature is that the AU method effectively decouples the air-side heat and mass transfer processes from the transfer of heat through the condensate layer and metal surfaces to the coolant. The air-side coefficients may then be determined independently of any assumptions regarding the heat transfer properties of the condensate and metal components, and the coolant. This is in contrast to the practice advocated by ARI Standard 410-81, whereby the accuracy to which the air-side component of the heat transfer can be

determined is critically dependent upon the accuracy of the models used to determine the metal and coolant-side thermal resistances. The shortcomings of the fin efficiency concept, which is central to the ARI model, have been described above. The suitability of the McAdams equation, which is specified by ARI Standard 410-81 for determination of the tube-side convective heat transfer coefficient, is discussed in section 7.6. The failure of the ARI method to account explicitly for heat transfer through the condensate layer gives rise to further complications when wet coil surfaces are considered.

7.5.3. Data Correlation.

7.5.3.1. Air-side Heat Transfer Coefficient.

Colburn (1933) established an analogy between fluid friction and sensible heat transfer, and was thus able to show that test data for a heat exchanger are well correlated over a wide range of values of the Prandtl number and Reynolds number if the dimensionless parameter

$$j \equiv StPr^{2/3} \tag{7.5.13}$$

is plotted as a function of Reynolds number. St is the Stanton number, defined as

$$St \equiv \frac{h_{c,ow}}{GC_{p,a}} \tag{7.5.14}$$

The factor j defined by equation (7.5.13) is frequently referred to as the sensible heat transfer j-factor. The analogy was carried further by Chilton and Colburn (1934), who suggested that the sensible heat transfer j-factor could also be used to correlate mass transfer data. That is, if we define a mass transfer j-factor,

$$j_m \equiv St_m Sc^{2/3} \tag{7.5.15}$$

where

$$St_m \equiv \frac{K_w}{G} \tag{7.5.16}$$

is a Stanton number for mass transfer, then provided the rate of mass flow is low,

$$j \approx j_m \tag{7.5.17}$$

for a wide range of fluids and flow configurations.

An equation of the form

$$StPr^{2/3} = a \times Re^{b} \tag{7.5.18}$$

where a and b are experimentally determined constants, has been found to correlate sensible heat transfer data particularly well (Kays and London, 1984; McQuiston, 1976; Idem, Jacobi and Goldschmidt, 1990). Kays and Crawford (1993) recommend this form

for both turbulent and laminar flow[76]. Reynolds number and Prandtl number in this instance are calculated at an air state corresponding to

$$t_m = \frac{t_{a1} + t_{a2}}{2} \qquad \textbf{(7.5.19a)}$$

$$W_m = \frac{W_{a1} + W_{a2}}{2} \quad . \qquad \textbf{(7.5.19b)}$$

The question now arises as to the most appropriate length scale to define the Reynolds number. Following Kays and London (1984), it has become common practice to use a Reynolds number based on the hydraulic diameter,

$$Re_d = \frac{GD_h}{\mu} \quad . \qquad \textbf{(7.5.20)}$$

In the above, G is the mass velocity based on the minimum free-flow area. That is,

$$G = \frac{\dot{m}_a}{A_m} \quad . \qquad \textbf{(7.5.21)}$$

The hydraulic diameter,

$$D_h = 4\frac{A_m}{A_o}L_d \quad . \qquad \textbf{(7.5.22)}$$

In a recent paper, Luxton and Shaw (1990) expressed reservations concerning the use of hydraulic diameter as a characteristic length in the analysis of tube and plate fin heat exchangers, for which the aspect ratios of the flow cross-sections are large, arguing that the flow regime within a heat exchanger is largely determined by the fin spacing. In an early investigation of the calculation of friction factor for ducts of high aspect ratio, Cornish (1928) suggested that correlation between the friction factor in such ducts and their circular equivalents could be improved by multiplying the hydraulic diameter by a geometry factor, such that

$$Re^* = \phi^*\left(\frac{F_h}{L_c}\right)Re_d \quad . \qquad \textbf{(7.5.23)}$$

The form of the geometry factor was rigorously derived by Cornish. Jones (1976) presents a simplified formulation which matches the more complex form derived by Cornish to within 2%:

---

[76] The form of equation (7.5.18) is only applicable to laminar flows in the absence or near-absence of longitudinal pressure gradients. It is applicable over a much larger range in the case of turbulent flows (Incropera and DeWitt, 1990).

$$\phi^*\left(\frac{F_h}{L_c}\right) \approx \frac{2}{3} + \frac{11}{24}\frac{L_c}{F_h}\left(2 - \frac{L_c}{F_h}\right) \qquad \textbf{(7.5.24)}$$

The effect of applying the geometry factor is to reduce the Reynolds number by a factor of approximately one third for ducts of high aspect ratio. Jones has shown that the use of this factor indeed results in close correlation between friction factor measurements for circular and rectangular ducts over a wide range of aspect ratios for both laminar and turbulent flows.

If the geometry factor is applied to the test results for an individual coil, the result is simply to scale the abscissa on the correlation plot. In view of the analogy between heat and momentum transfer (Kays and Crawford, 1993), it is tempting to hypothesize that applying the geometry factor may provide a means of correlating heat and mass transfer data between coils which are identical in all respects apart from fin pitch. Unfortunately, we do not yet have the experimental data necessary to test this hypothesis. Accordingly, the Reynolds number based on hydraulic diameter (7.5.20) has been used as the correlating parameter in the present study. This is subject to review when the required experimental data becomes available.

7.5.3.2 Metal and Condensate Resistances.

ARI Standard 410-81 prescribes a procedure for calculating the metal resistance based on the fin efficiency concept. The resistance of the condensate layer is not explicitly modelled in that work. Data reduction using the AU method leads to a determination of the *sum* of the metal and condensate resistances (equation 7.5.12). As in Sekhar's original work (1990), this sum is correlated as a weak function of $h_{c,ow}$, the difference being that in the current work we use a first or second-order polynomial, as opposed to the piecewise linear fit used by Sekhar. There is in fact little physical basis for assuming that this is a satisfactory means of correlating the data, and it remains perhaps the weakest aspect of the AU method. It is unlikely that the functional dependence of these resistances can be more accurately prescribed until the physical processes upon which they depend are better understood. Preliminary studies to provide such understanding have been undertaken at the University of Adelaide (see section 7.2), and these are currently ongoing areas of research.

7.5.4. Solution of Problem AW using the AU Method.

To summarize the above, if a series of tests are performed on a coil and the results reduced using the AU method, the products will be two calibration curves describing

i. The Colburn j-factor as a function of Reynolds number (equation 7.5.18).

ii. The sum of the metal and condensate thermal resistances as a function of the air-side thermal resistance.

These relationships may be inverted in the above sequence to provide an estimate of the air-side heat transfer coefficient $h_{c,ow}$, and the internal heat transfer coefficient h_i. The fluid properties required to invert equation (7.5.18) are evaluated at the air state defined by equation (7.5.19). The heat transfer coefficients are involved in the analysis in two places:

i. Estimation of the coil characteristic using equation (7.3.8).

ii. For the fully-wet section of the coil surface, the log-mean enthalpy difference (*LMHD*) is calculated using equation (7.4.20). For an estimated value of the heat transfer across the wet coil surface ($\tilde{q}_{tw}$), the wet surface area follows as[77]

$$\tilde{A}_w = \tilde{q}_{tw} \frac{C_{p,a}}{h_{c,ow} LMHD} \quad . \tag{7.5.25}$$

7.6. Coolant Specifications.

The preceding sections have presented two alternative approaches to modelling the performance of a finned-tube heat exchanger each based on the dual-potential approach. The dual-potential model in effect decouples the solution for the air-side heat transfer from that for the coolant-side heat transfer, the two solutions being related by the need to satisfy an overall energy balance. This implies that an experimentally-determined parameterization of the air-side heat transfer characteristics for a family of heat exchangers, developed for a given coolant, will in principle be applicable to any other coolant, provided the coolant-side heat transfer coefficient is specified appropriately. This consideration has influenced the selection of the data structures which have been developed to represent the cooling coil in the present study. The cooling coil is represented by an abstract base class *Coil*. This will be described in some detail in chapter 8. The main point to note here is that class *Coil* implements member functions to predict coil performance using the techniques developed in the preceding sections. Coolant-dependent aspects of the solution procedure are implemented by calling appropriate functions, which are declared to be *pure virtual*. In other words, they are declared but not defined within the base class; the appropriate definitions must be supplied by classes derived from class *Coil*. Thus, class *CW_Coil* provides the functions necessary to effect a solution for chilled water coils. A derived class *DX_Coil* for direct-expansion coils is proposed.

Current methods for rating cooling coils require that the coolant-side heat transfer coefficient be calculated according to a *specified* relationship, the air-side parameters being derived from the change in the bulk air properties of the fluids flowing through the heat exchanger, in accordance with the model used. Obviously, the extent to which the desired decoupling will be achieved will depend critically upon the quality of the relationships used to describe the properties of the coolant. This will have implications not only for predictions involving alternative coolants, but also for the extension of the model into

---

[77] Equation (7.5.25) is derived from equation (7.3.5), which in turn is derived by neglecting the final term in equation (7.3.4). Note that neglect of this term is not central to the AU method, and that accuracy may be slightly improved by including it.

regimes for which the coolant heat transfer relationships used to derive the rating curves are no longer valid. The purpose of the following sections is to describe a set of relationships for the thermodynamic and transport properties of water, and for the heat transfer characteristics of chilled water flowing in a tube. The discussion in this section is restricted to chilled water coils; some consideration of direct expansion coils is given in chapter 11.

7.6.1. Thermophysical Properties of Water.

Chapter 11 describes an abstract base class *fluid* which may be used to calculate the thermodynamic and transport properties of a fluid in the liquid, gaseous or wet-vapour states, subject to the equation of state and various other relationships being defined by a derived class. The properties of liquid water required by the coil prediction model are provided by class *water*, which is derived from class *fluid*. The relationships used to calculate these properties are the following.

7.6.1.1. Specific Volume.

Hyland and Wexler (1983a) provide the following relationship for the specific volume of liquid water, valid in the range $0 \le t \le 200°C$:

$$v_f = \frac{\sum_{i=0}^{1} a_i T^i}{\sum_{j=0}^{5} b_j T^j} \qquad (m^3/kg) \qquad \textbf{(7.6.1)}$$

where T is the absolute temperature (K), and

$a_0 = -0{\cdot}3424442728 \times 10,$
$a_1 = 0{\cdot}1619785 \times 10^{-1},$
$b_0 = -0{\cdot}2403360201 \times 10^{4},$
$b_1 = -0{\cdot}140758895 \times 10,$
$b_2 = 0{\cdot}1068287657,$
$b_3 = -0{\cdot}2914492351 \times 10^{-3},$
$b_4 = 0{\cdot}373497936 \times 10^{-6},$
$b_5 = -0{\cdot}21203787 \times 10^{-9}.$

7.6.1.2. Specific Heat.

From ASHRAE (1973), the specific heat of liquid water may be calculated using the relationship

$$C_{p,w} = A + BT + CT^2 + DT^3 + ET^4 \qquad (kJ/kg.K) \quad . \qquad \textbf{(7.6.2)}$$

The appropriate values for the empirical constants are given in table 7.1.

Range, K	A	B	C	D	E
273-450	$1{\cdot}76611\times10^{1}$	$-1{\cdot}47914\times10^{-1}$	$6{\cdot}08619\times10^{-4}$	$-1{\cdot}11867\times10^{-6}$	$7{\cdot}80297\times10^{-10}$
450-603	$-9{\cdot}66159\times10$	$6{\cdot}35694\times10^{-1}$	$-1{\cdot}33872\times10^{-3}$	$9{\cdot}44662\times10^{-7}$	0·0

Table 7.1. Constants for the equation for the specific heat of liquid water (eq. 7.6.2).

7.6.1.3. Dynamic Viscosity.

ASHRAE (1973) provides the following relationship for the dynamic viscosity of liquid water:

$$\mu = \exp\left(A + \frac{B}{T} + \frac{C}{T^2}\right) \qquad (Pa.s) \qquad \textbf{(7.6.3)}$$

The appropriate values for the empirical constants are given in table 7.2.

Range, K	A	B	C
273-350	0·030185	-2191·60	638605·0
350-500	-3·22950	13·18574	265531·0
500-620	-8·77361	5875·87	-1282750·0

Table 7.2. Constants for the equation for the dynamic viscosity of liquid water (eq. 7.6.3).

7.6.1.4. Thermal Conductivity.

ASHRAE (1973) provides the following relationship for the thermal conductivity of liquid water:

$$k = A + BT + CT^2 + DT^3 \qquad (W/m.K) \qquad \textbf{(7.6.4)}$$

The appropriate values for the empirical constants are given in table 7.3.

Range, K	A	B	C	D
273-400	-0·61694	$7{\cdot}17851\times10^{-3}$	$-1{\cdot}16700\times10^{-5}$	$4{\cdot}70358\times10^{-9}$
400-600	-0·14532	$4{\cdot}02217\times10^{-3}$	$-4{\cdot}64993\times10^{-6}$	$-4{\cdot}89257\times10^{-10}$
600-645	190·404	-0·94131	$1{\cdot}55847\times10^{-3}$	$-8{\cdot}61953\times10^{-7}$

Table 7.3. Constants for the thermal conductivity equation for liquid water (eq. 7.6.4).

7.6.2. Water Film Heat Transfer Coefficient and Friction Coefficient.

The importance of using relationships which accurately characterize the coolant-side heat transfer performance when reducing data from cooling coil rating tests has already been discussed. In practice, relationships empirically derived from tests in straight tubes are invariably used for this purpose. No account is taken of the possible influence of return bends. These will have a significant effect on the pressure loss, but experimental evidence indicates that the effect on heat transfer characteristics is far less pronounced (Van Aken, 1993). In the current work, the procedure is accepted as valid, while noting that it is a possible source of discrepancy.

ARI Standard 410-81 decrees that the water film heat transfer coefficient for coil designs having smooth, plain internal tube walls, and operating at a Reynolds number in excess of 3,100 *shall* be calculated using the McAdams equation (equation 7.4.5). The McAdams equation is derived from the well-known Dittus-Boelter (1930) equation:

$$Nu = 0{\cdot}023\,Re^{0{\cdot}8}\,Pr^{n} \qquad \textbf{(7.6.5)}$$

where $n = 0{\cdot}4$ for heating, and $0{\cdot}3$ for cooling. This equation is stated by McAdams to be valid within the range of $0{\cdot}7 \le Pr \le 120$, and $10{,}000 \le Re \le 120{,}000$. As the Reynolds number is reduced below 10,000, the McAdams equation is found to systematically overestimate the water film heat transfer coefficient, when compared with the experimental curves of Sieder and Tate (1936), with the result that the water flow rate required to offset a given load will be correspondingly underestimated, the error reaching 200-300% at the lower end of the transition region ($2{,}000 < Re < 3{,}000$). Clearly, the assertion made by ARI Standard 410-81 that the McAdams equation can be used for Reynolds numbers down to 3,100 must be challenged; this relationship *should* not be used for coil performance prediction, and *must not* be used for rating purposes for Reynolds numbers of less than 10,000. There is in fact reason to query whether the McAdams equation should be used at all, since the underlying Dittus-Boelter equation is now known to overpredict Nusselt number by at least 20% for gases, and to underpredict Nusselt number for the higher Prandtl number fluids by some 7-10% (Kays and Crawford, 1993). The predictions of the Dittus-Boelter equation, when compared to more recent correlations for the Nusselt number (see later) are shown in figure 7.14. In this connection it is worth commenting briefly on the policy adopted by earlier workers in reducing coil test data. McQuiston (1978a) uncritically adopted the Dittus-Boelter equation (7.6.5). Rich (1973) adopted the *form* of the McAdams equation (7.4.5), but experimentally determined the leading coefficient using a Wilson plot. In a series of experiments covering an approximate water velocity range of 0·2 to 2·1 m/s, the value of the coefficient was found to be consistently higher than the value proposed by McAdams, the discrepancy varying between 1 and 13%.

For standard ⅝" internal diameter tubes, a Reynolds number of 10,000 corresponds to a water velocity of approximately 0·75 m/s. Cooling coils are routinely operated at lower velocities, and in conventional practice coils may be operated in the transition region over most of their range. There is a need to specify relationships which will more accurately describe the water-side heat transfer performance within the laminar and transition regions.

In the following development it should be borne in mind that we seek a relationship or set of relationships which will serve two distinct purposes:

a. Coil rating tests seek to characterize the air-side heat transfer performance of the coil accurately, based on a predicted value for the coolant-side heat transfer coefficient. If the two effects are to be accurately decoupled, the relationship used for the water-side coefficient should be accurate and free of ambiguity. In practice, these requirements can most readily be met by performing rating tests in the fully turbulent region ($Re > 10{,}000$). In such circumstances thermal entry length effects and secondary flows induced by the return bends will be minimized. The uncertainty regarding the exact state of the flow, as experienced in connection with transitional flows, will also be eliminated.

b. Having determined the air-side coil characteristics on the basis of a series of tests, we may use that information to predict the performance of the coil over a range of operating conditions. If the water flow rate is specified, the accuracy of the water-side solution will depend on the accuracy of the heat transfer relationship used. This is generally not a problem within the fully turbulent region. However, as the flow velocity falls, considerable uncertainty arises concerning the stability of the flow in the transition region. With a further reduction in flow velocity, the flow becomes fully laminar. In such circumstances it may become necessary to estimate the effect of the thermal entry length (Kays and Crawford, 1993), and of the return bends, depending on the accuracy of solution desired. The problems posed by the algorithms of chapter 6 are however of a different type. In this case we seek to determine the equilibrium air-side conditions for a coil or coils operating within an air conditioning cycle. These may be determined solely on the basis of the known air-side heat and mass transfer characteristics. The solution *specifies* a required water-side heat transfer coefficient; the necessary water flow rate may be calculated *a posteriori*.

In this latter case it is the maximum water velocity, water flow rate and pressure drop which are of fundamental importance in assessing the suitability of a design[78]. These should be determined to a reasonably high degree of accuracy, which is generally attainable, since such conditions are invariably associated with operation in the turbulent or high in the transition region. With reducing velocity, the need for accuracy diminishes, although certain bounds must be placed on the degree of inaccuracy which can be tolerated to avoid prediction of physically unrealizable water temperature rises. The following relationships are considered adequate to meet the requirements of the model:

a. For fully developed turbulent flow in a pipe, Kakak, Shah and Aung (1987) recommend the following correlation developed by Gnielinski (1976):

---

78 Maximum water consumption is always associated with peak load. For a conventional design, this will also coincide with the condition of maximum water velocity and pressure drop. For a design incorporating circuiting changeovers, maximum water velocity and pressure drop may be associated with the top of a stage, on rising load. In most if not all cases, maximum pump power consumption will occur at peak load.

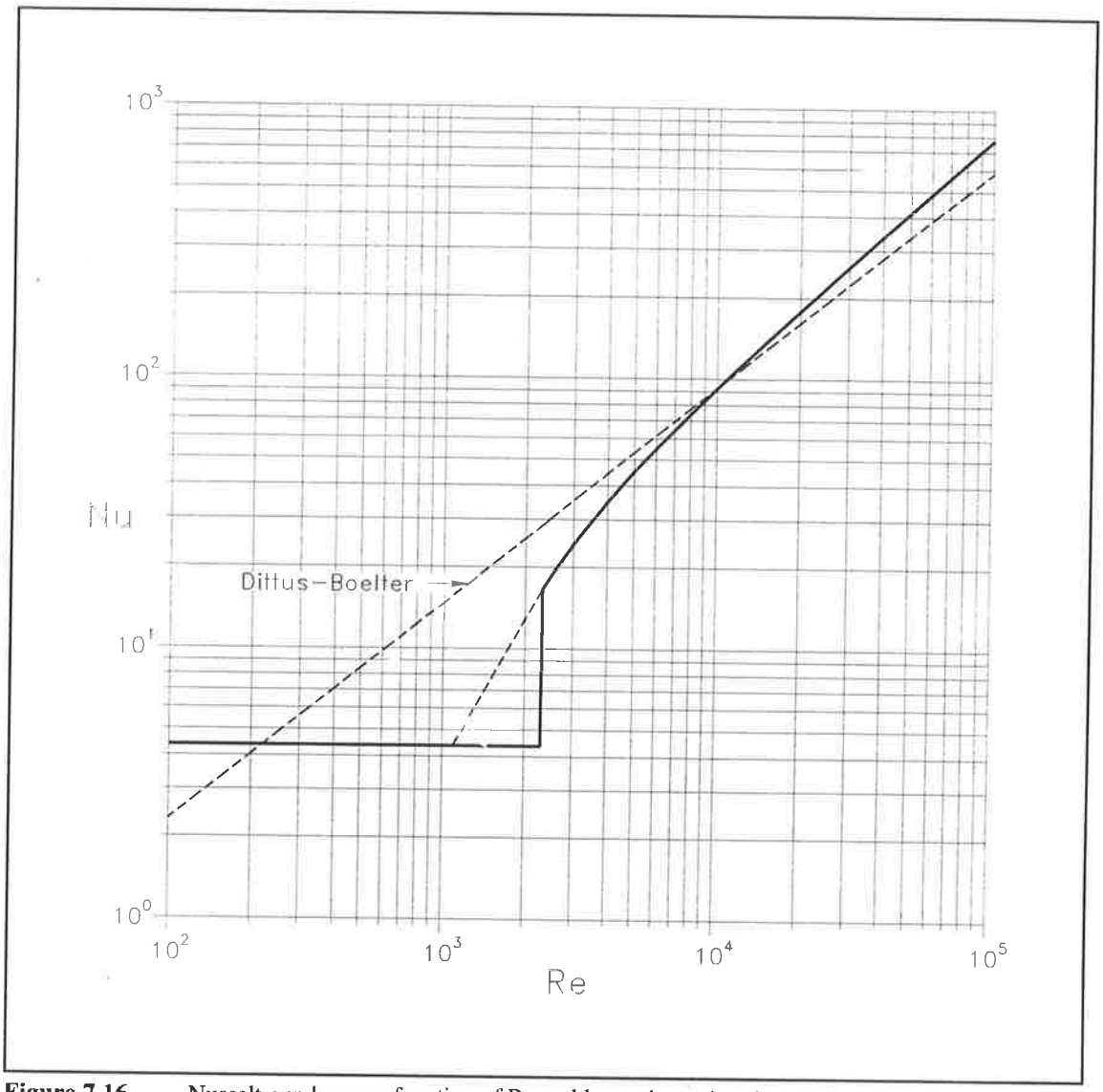


Figure 7.16. Nusselt number as a function of Reynolds number using the relationships described in the text. The Dittus-Boelter relationship is shown for comparison.

$$Nu = \frac{(Re - 1000)Prf/8}{1{\cdot}0 + 12{\cdot}7\sqrt{f/8}(Pr^{\frac{2}{3}} - 1{\cdot}0)} \qquad \textbf{(7.6.6)}$$

This relationship is valid over the range $0{\cdot}5 < Pr < 2{,}000$ and $2{,}300 < Re < 5\times10^6$. The effect of flow disturbances on the transition point for flow in a straight drawn-copper tube is graphically illustrated by the experimental work of Ito (1959). For commercial air conditioning coils it is reasonable to assume that disturbances associated with the return bends will be adequate to bias the transition point towards the lower end of the transition region[79]. Equation (7.6.6) will accordingly

[79] Note however that relaminarization may occur within the return bends themselves (Ito, 1959; Akayima et al., 1988; see also section 7.6.3), particularly in the present case, where the return bends are very tight. This does not however invalidate the argument that the secondary flows induced by the return bends, together with irregularities associated with the joining process are likely to promote

be used in all cases for both rating and simulation, where $Re \geq 2{,}300$. In order to use this equation it is necessary to specify a complementary relationship for the friction factor, f. A widely used expression for the friction factor for turbulent pipe flow is Prandtl's universal friction law for smooth pipes (Ward-Smith, 1980), which is valid over the range $4{,}000 < Re < 3{\cdot}4\times10^6$:

$$\frac{1}{\sqrt{f}} = 0{\cdot}87 \ln[Re\sqrt{f}] - 0{\cdot}80 \quad . \tag{7.6.7}$$

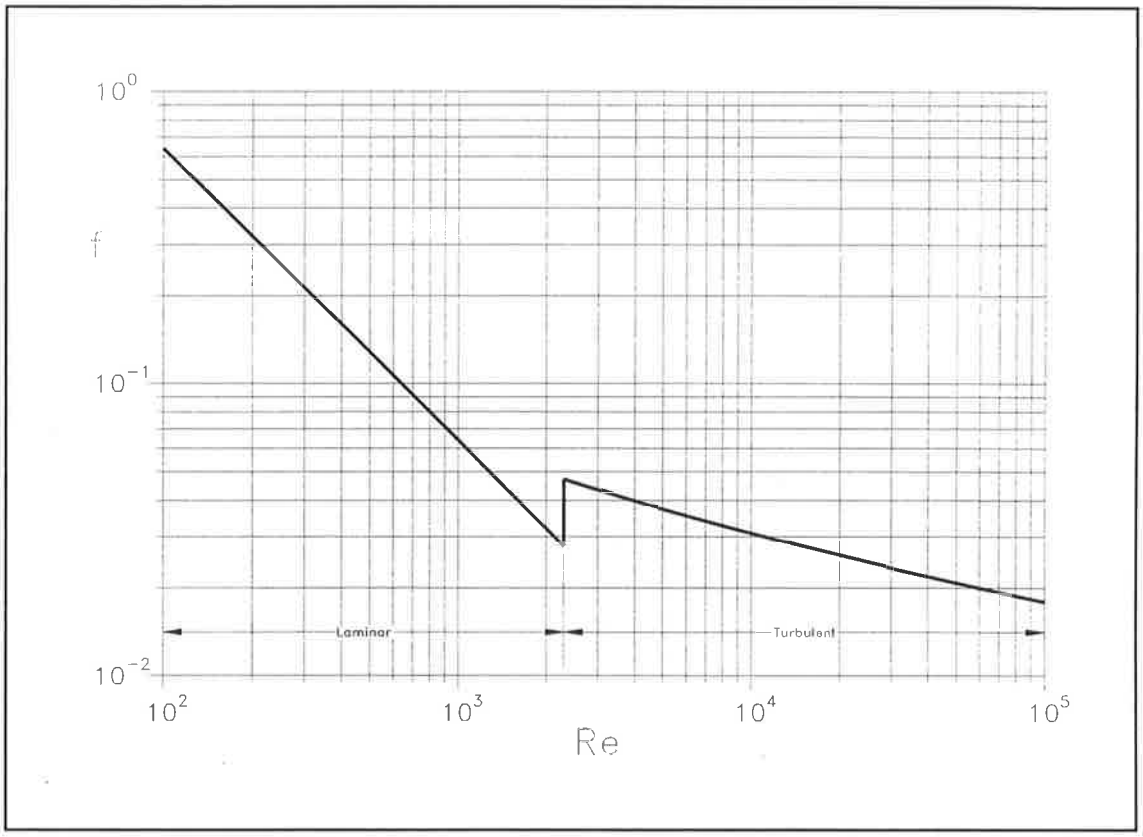


Figure 7.17. Friction factor as a function of Reynolds number using the relationships described in the text.

The experimental results of Ito (1959) show that while equation (7.6.7) provides an excellent correlation for fully developed turbulent flow at $Re > 4{,}000$, the measured values depart systematically from the predicted values at lower Reynolds numbers. For the present purposes however, equation (7.6.7) may be used with acceptable loss of accuracy down to $Re = 2{,}300$. Equation (7.6.7) is an implicit expression which can be solved (see section 7.3.6) given an initial estimate for f. The following expression due to Colebrook (1939) is suitable for this purpose:

---

transition to turbulence within the straight sections of the pipe. It would be worthwhile to perform experimental and numerical studies with the aim of investigating the effect of the return bends on the water-side heat transfer performance of a coil.

$$f = \frac{1}{\left[0{\cdot}78\ln\left(\frac{Re}{7}\right)\right]^2} \quad . \tag{7.6.8}$$

b. For fully-developed laminar flow Kays and Crawford (1993) provide analytical solutions for the heat transfer problem where the heat transfer rate per unit length of tube is constant, and where the surface temperature is constant. The former situation more closely describes that occurring in a chilled water cooling coil. For this case[80],

$$Nu = 4{\cdot}364 \tag{7.6.9}$$

For fully developed laminar flow the friction loss coefficient

$$f = \frac{64}{Re} \quad . \tag{7.6.10}$$

Nusselt number as a function of Reynolds number over the range $100 < Re < 10^5$, as predicted by equations (7.6.9) and (7.6.6) is shown in figure 7.16. The correlation shown is appropriate for chilled water at a mean temperature of 10°C ($Pr = 9.2$). Figure 7.17 shows the corresponding friction factor, as predicted by equations (7.6.10) and (7.6.7). The discontinuity between equations (7.6.9) and (7.6.6) at $Re = 2300$ raises the possibility of control instabilities arising if cooling coils are operated in this range. This is unlikely to occur in practice[81]. However, this point should be borne in mind when devising control strategies for coils which may operate in this region.

---

80 Equation (7.6.9) is appropriate for fully developed laminar flow. The cooling coil paths used in air conditioning practice are almost always sufficiently long that thermal entry length effects can be neglected. Analytical methods to account for the thermal entry length are available (Kays and Crawford, 1993). However, in view of the simplifying assumptions which have been made, and the relative unimportance of aspiring to a particularly high degree of accuracy for flow in the laminar region, equation (7.6.9) should suffice.

81 Other considerations aside, the curves shown in figure 7.16 are based on measurements performed using *straight* tubes and make no allowance for the effect on heat transfer in the tubes of secondary flows induced by the return bends. These are likely to result in some modification of the heat transfer relationships.

The discontinuity shown in figure 7.16 also gives rise to numerical problems when the bracketing interval contains the discontinuity. To overcome this problem, the curve represented by equation (7.6.6) is extended down to the left until it intersects the curve represented by equation (7.6.9), as shown by the dashed line segment in figure 7.16, thus enabling Nu to be expressed as a continuous function of Re.

7.6.3. Water Pressure Drop.

For a given coil, the water will flow through N_{ct} circuits in parallel. Since each circuit is identical, it is only necessary to calculate the pressure loss for one such circuit. Neglecting entry and exit losses, this may be expressed as

$$\Delta p_w = N_{pc}\Delta p_p + (N_{pc} - 1)\Delta p_b \qquad \textbf{(7.6.11)}$$

where Δp_p is the pressure drop per pass of straight pipe, and Δp_b is the pressure drop through a return bend.

For a straight length of pipe, the pressure drop may be found from

$$\Delta p_p = f\frac{L_t}{D_i}\frac{\rho_w U_w^2}{2} \qquad \textbf{(7.6.12)}$$

where f may be found using equation (7.6.10) or (7.6.7), as appropriate.

The pressure drop through a bend can be expressed in terms of a pressure drop coefficient,

$$K = \frac{\Delta p_b}{\frac{1}{2}\rho_w U_w^2} \qquad \textbf{(7.6.13)}$$

For estimating pressure drops, the preferrred form of the coefficient is the gross pressure drop coefficient (Ward-Smith, 1980; Ito, 1960), which is based on measurements sufficiently far removed from the bend. This can be approximated by

$$K_G = \frac{f_c R_c \Theta}{D_i} \qquad \textbf{(7.6.14)}$$

where,

f_c is the friction coefficient for fully developed curved flow,
R_c is the radius of curvature of the bend, and
Θ is the angle subtended by the bend.

Flow through a curved pipe generates secondary flows (Ito, 1959, 1960; Ward-Smith, 1980; Akiyama et al., 1988), which has the effect of suppressing the turbulence of the flow. Thus, even though the flow in the straight sections of the pipe may be fully turbulent, relaminarization can occur in the bends, and some criterion other than the critical Reynolds number is required to determine the state of the flow through the bend. The relevant dimensionless parameter is the Dean number, defined as

$$D = Re\sqrt{\frac{a}{R_c}} \qquad \textbf{(7.6.15)}$$

where a is the internal pipe radius. It is possible to formulate a critical Dean number, which is the highest value of the Dean number for which the flow is always laminar. Ward-Smith (1980) shows that the following relationship correlates the available experimental data well over the range $10 < R_c/a < 200$ (see footnote [82]):

$$D_{crit} = 2300\sqrt{\frac{a}{R_c}}\left(1 + 10\sqrt{\frac{a}{R_c}}\right) \qquad \textbf{(7.6.16a)}$$

or

$$Re_{crit} = 2300\left(1 + 10\sqrt{\frac{a}{R_c}}\right) \qquad \textbf{(7.6.16b)}$$

Note that this equation asympototically approaches the straight pipe relationship as R_c/a becomes infinitely large.

For laminar flow in a smooth pipe, Ito (1959) has shown that the following correlation due to White (1929) fits the available experimental data well over the range $17 < D < 1000$:

$$\frac{Re\,f_c}{64} = \frac{1}{1 - \left[1 - \left(\frac{11{\cdot}6}{D}\right)^{0{\cdot}45}\right]^{2{\cdot}222}} \qquad \textbf{(7.6.17)}$$

while for D > 1000, Ward-Smith (1980) recommends that the following relationship due to Collins and Dennis (1975) be used:

$$\frac{Re\,f_c}{64} = 0{\cdot}1028\sqrt{D}\left[1 + \frac{3{\cdot}70}{\sqrt{D}}\right] \quad . \qquad \textbf{(7.6.18)}$$

For fully turbulent flow in smooth pipes, Ito (1959) has found that the following empirical relationship provides a close fit to the experimental data over the range $0{\cdot}034 < D < 300$:

$$f_c\sqrt{\frac{R_c}{a}} = 0{\cdot}209 + 0{\cdot}304\left[Re\left(\frac{a}{R_c}\right)^2\right]^{-0{\cdot}25} \quad . \qquad \textbf{(7.6.19)}$$

For a particular circuiting configuration it is likely that return bends of several different radii will be used. The appropriate radii can be determined if the circuiting is known. It is more convenient however to assume an approximate but constant radius throughout. For this purpose we use

---

82 In air conditioning practice, values in the range $2 < R_c/a < 5$ are more typical. It is recommended that equation (7.6.16) be used in the absence of more appropriate correlations.

$$R_c \approx \frac{\sqrt{S_f^2 + S_r^2}}{2} \qquad \textbf{(7.6.20)}$$

With the return radius calculated on this basis for a standard ⅝" diameter tube, the curved flow friction coefficient computed according to equations (7.6.16-19) over the range $100 < Re < 10^5$ will vary as shown in figure 7.18. For this case $R_c/a \approx 3{\cdot}5$ and $Re_{crit} \approx 14{,}645$.

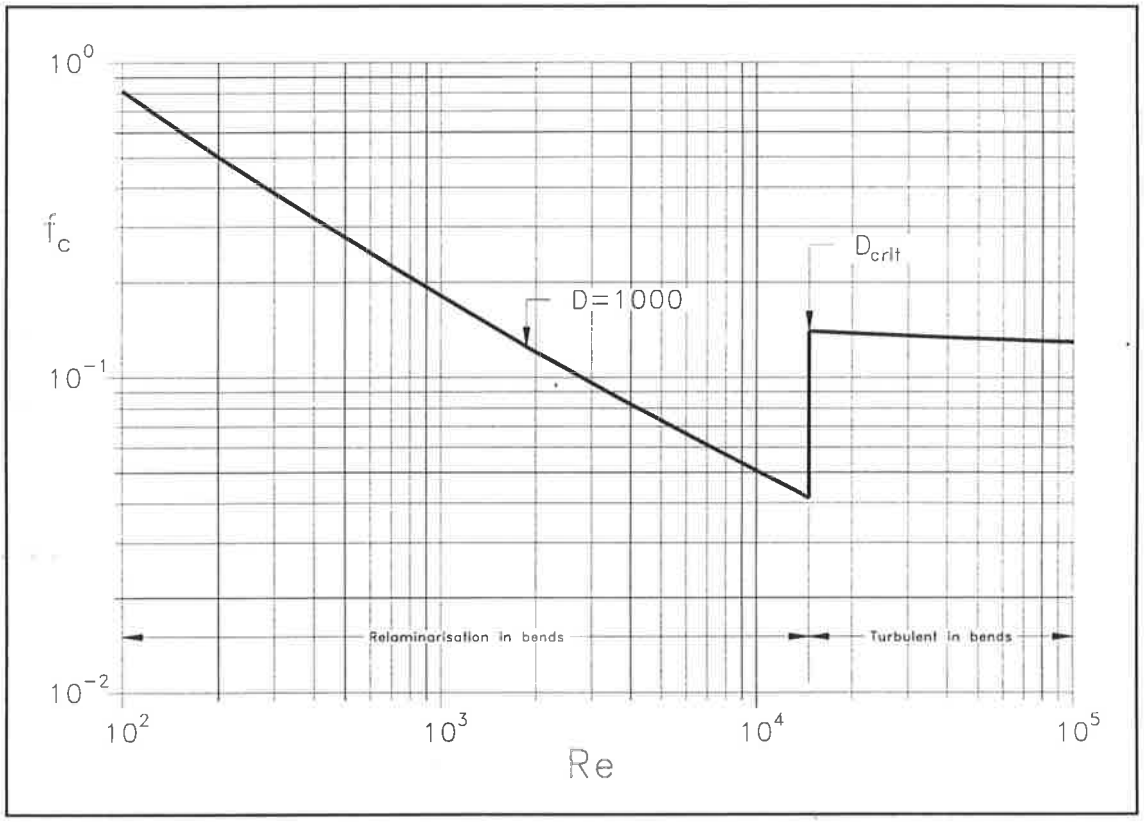


Figure 7.18. Curved flow friction factor as a function of Reynolds number for typical bends in standard ⅝" cooling coils (see text).

7.7. Air Pressure Drop.

The operating point, and consequently the power consumption of a fan in an air conditioning system will be determined by the pressure loss characteristic of the air distribution system. The cooling coil may contribute significantly to the total pressure drop experienced by an air stream. The pressure drop across the coil may become important too in assessing the suitability of a design from the point of view of ease of pressure balancing, particularly where more than one coil is involved. Thus, it is necessary to be able to estimate the pressure drop for a given coil over its range of operating conditions.

Following Kays and London (1984), it is customary to express pressure drop in terms of the Fanning friction factor,

$$f = \Delta p \frac{2}{\rho V_c^2} \frac{A_m}{A_o} \tag{7.7.1}$$

where V_c is the air velocity based on the minimum free-flow area. Chapter 9 describes a methodology for calculating the total pressure drop for an air distribution system. For compatibility with the other fittings in the system, the pressure loss will need to be expressed in terms of local loss coefficient, C. This may be calculated from the Fanning friction factor as

$$C = f \frac{V_c^2}{V_f^2} \frac{A_o}{A_m} = f \frac{A_f^2 A_o}{A_m^3} \tag{7.7.2}$$

where V_f is the face velocity.

Measurements of the friction factor have been made by a number of workers. Following Kays and London (1984) again, it is common practice to present the friction factor for a particular coil configuration as a function of the Reynolds number based on the hydraulic diameter (equation 7.5.22).

Idem et al. (1990) suggest, on the basis of their work with integral-finned heat exchangers, that the following relationship exists between the friction factor and the Reynolds number for a dry coil,

$$f = a \times Re_d^{-b} \tag{7.7.3}$$

where a and b are empirical constants. A similar expression is used for fully wet coil data. The data of Idem et al. however exhibit considerable scatter. Rich (1973) presents a set of experimental measurements which would appear to have been obtained under conditions of better quality control than those of Idem et al.[83] These latter data indicate that the simple logarithmic relationship of equation (7.7.3) is definitely not adequate to correlate the data.

The presence of condensation further complicates the matter. Those authors who have published friction factor data for fully-wet coils (McQuiston, 1978a; Eckels and Rabas, 1987; Idem et al., 1990) also present corresponding data for the same coils in a fully-dry state. These studies indicate unequivocally that the presence of condensation will result in an increase in the friction factor. This result is hardly surprising. An experimentally derived friction factor will contain contributions from two separate mechanisms, shear stress and form drag. Both of these components will typically be enhanced by the presence of condensate. Eckels and Rabas (1987) have enjoyed partial success in accounting for the augmentation of shear stress during the condensation process by drawing an analogy between the vertical velocity component associated with the condensing vapour and boundary layer suction. They used the relationships of Hartnett and Eckert (1957) and Kinney and Sparrow (1970), derived for boundary layer suction, to estimate the shear stress

---

[83] The fact that Rich performed his tests on heating coils, thereby guaranteeing absence of condensation, may have been a contributing factor here.

enhancement to be expected due to the normal component of the vapour velocity. It is not clear whether the failure of this approach to account fully for the increase in friction factor is due more to shortcomings in the theory, or to the augmentation of form drag caused by the presence of condensate. It should be noted that the test coils used by Idem et al. were thoroughly cleaned before each run, with the result that most of their measurements were made under conditions of *filmwise* condensation. The form drag associated with *droplike* condensation, which is apparently the norm under operational conditions, can be expected to exceed that experienced in the case of filmwise condensation by a considerable margin. We cannot even start to quantify this effect until the surface geometry arising from droplike condensation has been properly characterized.

It is clear from the above that any prediction of the pressure drop to be expected from a given cooling coil operating under a specified set of operating conditions, must rely heavily on careful experimentation. Unfortunately, pressure drop is not one of the parameters measured in the experimental programme to be described in the next section. The procedures for predicting the coil friction factor which have been used in the present work have therefore been based on experimental data for comparable coils which have been published in the open literature. Probably the most comprehensive set of test data of this type is that of McQuiston (1978a), which covers dry coils and coils subject to filmwise and to droplike condensation. In a companion paper, McQuiston (1978b) published a set of correlations based on dimensional analysis, which attempt to provide a universal correlation of friction data for plate fin and tube cooling coils. The correlations are based both on McQuiston's own test results, and on relevant dry-coil test results published by Rich (1973) and by Kays and London (1984). The resulting curves are accurate to within ±35%. The relatively large error band is accounted for in part by minor differences in the geometrical configuration and constructional methods of the coils tested by the various authors. The major factors however would seem to be the relatively large scatter in McQuiston's data, together possibly with certain inadequacies in the correlating functions chosen. McQuiston's (1978b) correlations are presented uncritically below in the context of the present study[84]. In the longer term these should not be seen as a substitute for a systematic experimental study using coils of the exact type for which we wish to predict the pressure drop.

---

[84] Comparison of pressure drops estimated using McQuiston's correlation with predictions for the same coil using a number of proprietary coil selection programmes indicates a high degree of mutual incompatibility between the various estimation methods, the discrepancy between the highest and lowest estimates in some cases approaching 100%. Pressure drops estimated using McQuiston's correlation lie near the upper end of spectrum, and limited testing performed in our laboratory suggests that these higher figures more closely approximate the pressure drops to be expected in practice. Note however that in some situations the multiplicative factor $F(\dot{N}_f)$ of equations (7.7.12) and (7.7.13) may be less than one, indicating that the pressure drop through a coil subject to dropwise condensation is *smaller* than that through an identical dry coil subject to the same flow conditions. Obviously, the last word on this subject remains to be written. We are currently initiating a systematic series of laboratory tests in an attempt to a produce a set of pressure drop correlations which may be regarded as definitive.

See the discussion accompanying equation (7.5.22) for a possible alternative means of correlating frictional data from coils of different fin densities.

McQuiston's correlations are based on a parameter *FP*. A number of more basic quantities need to be defined before we can offer a definition for *FP*. The ratio of total surface area to primary surface area may be calculated as the ratio A_o/A_p using the quantities derived in section 7.1. For consistency with McQuiston's work however, we use the following expression

$$R_a \equiv \frac{A_o}{A_p} \approx \frac{4}{\pi}\frac{S_r}{D_h}\frac{S_f}{2x_b}\frac{A_m}{A_f} \quad . \tag{7.7.4}$$

Now, R^* is a kind of hydraulic radius defined by the equation

$$\frac{R^*}{x_b} = \frac{R_a}{(S_f - 2x_b)\dot{N}_f - 1} \quad . \tag{7.7.5}$$

Two further dimensionless groups follow:

$$F_1 = \frac{(S_f - 2x_b)\dot{N}f}{4(1 - \dot{N}_f Y_f)} \tag{7.7.6}$$

and

$$F_2 = \frac{S_f}{2R^*} - 1 \quad . \tag{7.7.7}$$

Thus, *FP* is defined as

$$FP = Re_d^{-0{\cdot}25}\left(\frac{x_b}{R^*}\right)^{0{\cdot}25} F_1^{-0{\cdot}4} F_2^{-0{\cdot}5} \quad . \tag{7.7.8}$$

For a dry coil surface McQuiston derived the correlation[85]

$$f_d = 4{\cdot}904 \times 10^{-3} + 1{\cdot}382(FP)^2 \quad . \tag{7.7.9}$$

Two further parameters must be defined in connection with wet coil surfaces. The first is a Reynolds number based on the fin spacing,

$$Re_s \equiv \frac{G}{\dot{N}_f \mu} \tag{7.7.10}$$

the other being a factor to account for the fin thickness,

---

[85] The first term in the following equation is variously given as $4{\cdot}904 \times 10^{-3}$ and $4{\cdot}094 \times 10^{-3}$ in McQuiston's paper.

$$F_s \equiv \frac{1}{1 - \dot{N}_f Y_f} \qquad \textbf{(7.7.11)}$$

whence McQuiston defines a function $F(\dot{N}_f)$ which essentially provides a correction to the dry coil curve to account for the presence of condensate; FP in (7.7.9) will be replaced by $FP \times F(\dot{N}_f)$. For droplike condensation[86], McQuiston presents the correlation

$$F(\dot{N}_f) = [0{\cdot}325 + Re_s^{-0{\cdot}05}]F_s^{-3} \quad . \qquad \textbf{(7.7.12)}$$

Thus, the friction factor for a fully-wet coil surface subject to droplike condensation follows:

$$f_w = 4{\cdot}904 \times 10^{-3} + 1{\cdot}382[FP \times F(\dot{N}_f)]^2 \quad . \qquad \textbf{(7.7.13)}$$

McQuiston states that the above correlations are valid over the range:

Tube diameter:		9·5 to 15·9 mm
Tube spacing	:	25·4 to 50·8 mm
Fin pitch	:	4 to 14 fins/inch
Fin thickness	:	0·15 to 0·25 mm
Face velocity	:	1 to 4 m/s

The correlations are appropriate to coils having four tube rows. In general, we are dealing with coils having an arbitrary number of rows, and for which the coil surface is neither fully wet nor fully dry. To predict the pressure drop for a coil in the general case, the following procedure is recommended:

i. The wet and dry friction factors for a four row deep coil subject to the same air flow regime as the test coil are calculated using equations (7.7.9) and (7.7.13). The wet and dry pressure drops for the four-row coil follow from equation (7.7.1).

ii. On the assumption that pressure drop is proportional to the number of tube rows, the pressure drops calculated above are converted to pressure drops per row. Thus, Δp_d is the pressure drop per row for a fully dry coil, and Δp_w is the pressure drop per row for a fully wet coil.

iii. The wet and dry surface areas are known from the heat transfer calculations. Thus, the pressure drop for the coil,

$$\Delta p = N_r \left[\frac{\Delta p_d A_d + \Delta p_w A_w}{A_o} \right] \qquad \textbf{(7.7.14)}$$

---

86 The corresponding correlation for filmwise condensation is considered to be of little relevance in the operational case, and is not reproduced here.

iv. The corresponding local loss coefficient may thus be calculated (equation 7.7.2).

The above procedure is based on a number of assumptions which are subject to experimental verification.

7.8. Experimental Measurement of Coil Characteristics.

An extensive series of coil rating tests have been performed by Sekhar (1990) using a unique closed-cycle thermal environment wind tunnel, which provides the ability to test coils in a closely controlled environment over a wide range of entering air conditions. The experimental facility and associated experimental procedures have been described in detail in earlier works (Sekhar, 1990; Sekhar et al., 1991a, b), and will not be described further here. It will be noted however that the test facility has recently been rebuilt at another site, and the opportunity was taken to provide even closer control of the operating parameters by implementing a number of direct digital control loops. Coil testing is continuing using the rebuilt rig.

The coil tests analyzed in the current work have been drawn entirely from the databank assembled by Sekhar, and cover coils having a fin density of 6 fins per inch[87], and 1, 2 and 4 rows in depth. A new software package has been developed to analyze the coil test data. The software implements analysis according to ARI standard 410-81 for fully dry coils, and according to the AU method for fully wet coils. The appropriate analysis procedure is selected using criteria developed by Sekhar (1990):

i. The coil is *fully dry* if the ratio of sensible to total heat, $q_s/q_t > 0{\cdot}95$.

ii. The coil is *fully wet* provided:
   a. The ratio of sensible to total heat, $q_s/q_t \leq 0{\cdot}75$.
   b. The entering water temperature (t_{wl}) lies within the range of 1·7 to 12·8°C.
   c. The entering wet-bulb depression, $t_{al} - t_{al}' \geq 3{\cdot}3°C$.
   d. The differential between the leaving wet-bulb temperature and the entering water temperature, $t_{a2}' - t_{wl} \geq 2{\cdot}8°C$.

iii. If neither of the above sets of criteria is satisfied, the run is classified as *partially dry*, and rejected.

---

[87] The modelling study described in chapter 13 presents the results of simulations performed using coils of 9 and 12 fins per inch. The coil rating curves for these fin densities have been derived by performing a series of "simulated experiments" using a proprietary coil simulation programme, and analyzing the results using the same procedures as have been used for the 6 fpi test results. The resulting relationships are of an interim nature only, and will not be reported herein; they will be replaced in due course with correlations based on tests performed in our laboratory. This will result in an improvement in the accuracy to which the performance of coils having these higher fin densities can be simulated, but is not likely to invalidate the conclusions of chapter 13.

The above criteria determine the default behaviour of the programme when presented with a set of test data. Processing according to the above criteria may be overridden by specifying the analysis method to be used by means of an optional command-line switch.

The analysis methods used are as described in sections 7.4 and 7.5, and generally follow the logic of the processing methods described by Sekhar (1990). The methods described herein differ from those of Sekhar on a number of points of detail, the most important of which are:

i. The water-side film heat transfer coefficient is calculated using equations (7.6.6) and (7.6.7) in preference to the McAdams equation (7.4.5), for both the AU method and the ARI method. In this respect the procedures described herein depart from strict adherence to ARI Standard 410-81.

ii. The specific heat and fluid transport properties are calculated at an appropriate mean temperature for the working fluids, using the relationships of Chapter 4 for moist air, and section 7.6.1 for water. In the earlier work by Sekhar, constant values were used for these properties.

iii. A similar procedure is adopted in calculating the Lewis number; Sekhar assumed a constant value of $Le = 0{\cdot}9$.

iv. The procedure used to extract the sensible and latent heat capacity of a test coil from the state of the air entering and leaving the coil is as shown in figure 7.19. Let subscript *1* denote the entering air state, subscript *2* denote the leaving air state, and subscript *i* denote a state determined by the dry-bulb temperature at state *1*, and the humidity ratio at state *2*. Then,

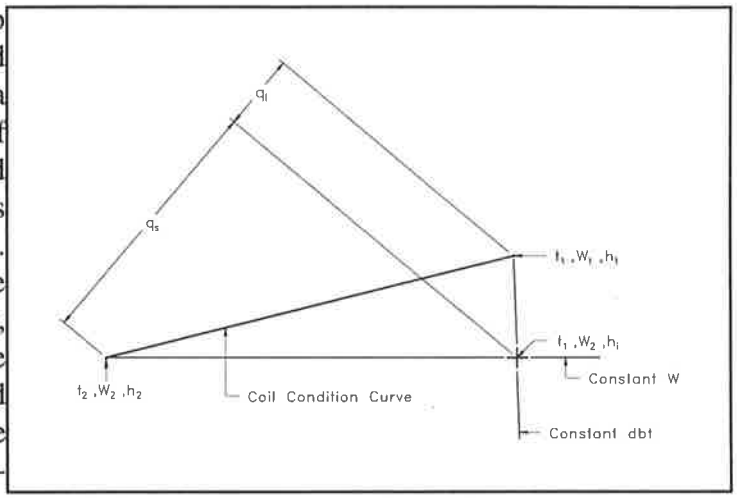


Figure 7.19. Extraction of sensible and latent heat capacity from coil test results.

$$q_l = \dot{m}_a(h_1 - h_i) \qquad \textbf{(7.8.1a)}$$

$$q_s = \dot{m}_a(h_i - h_2) \quad . \qquad \textbf{(7.8.1b)}$$

Sekhar et al. (1991b) used the relationships

$$q_l = \dot{m}_a h_{fg}(W_1 - W_2) \qquad \textbf{(7.8.2a)}$$

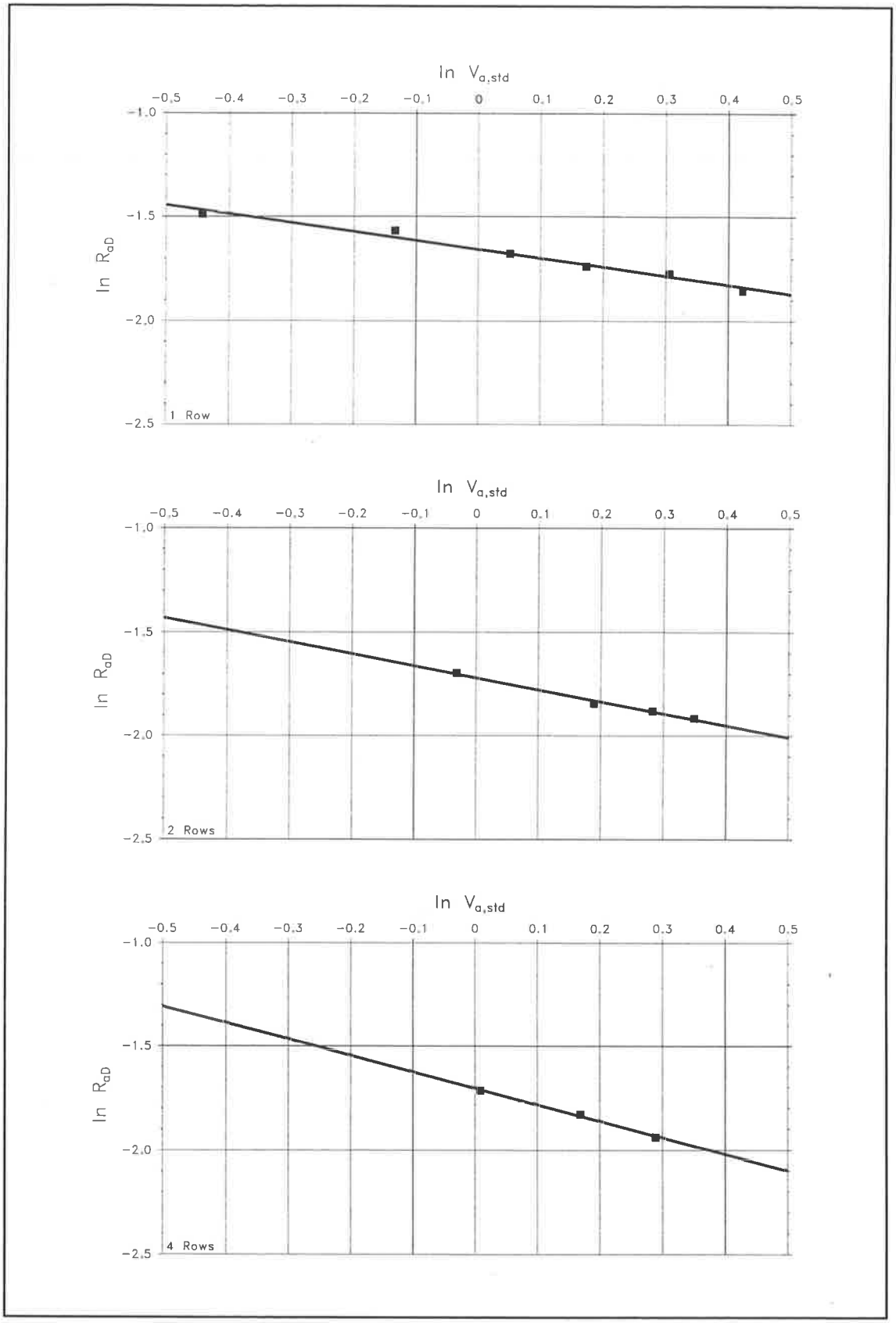


Figure 7.20. Calibration plots for the dry air-side thermal resistance for a family of coils, as prescribed by ARI Standard 410-81.

$$q_s = m_a C_{p,a}(t_1 - t_2) \quad . \tag{7.8.2b}$$

The latent heat can be calculated unambiguously using equation (7.8.2a). It is not clear however at what set of conditions the specific heat in equation (7.8.2b) should be evaluated. Van Aken (1993) provides relationships for a mean specific heat which may be used for this purpose. It is preferable however to use equation (7.8.1b), which is entirely unambiguous and more straightforward.

For the above reasons, the relationships presented in the following section differ slightly from those originally presented by Sekhar (1990).

7.8.1. Experimental Correlations.

The coils tested belong to a family of plate fin and tube coils manufactured by F. Muller and Sons Pty. Ltd. The geometry of these coils is specified by the following set of dimensions (refer to section 7.1):

Tube external diameter (D_o)	:	15·875 mm.
Tube internal diameter (D_i)	:	14·915 mm.
Fin thickness (Y_f)	:	0·19 mm.
Vertical tube spacing (S_f)	:	38·1 mm.
Spacing between coil rows (S_r)	:	35·05 mm.
Fin material	:	Aluminium.
Tube material	:	Copper.

All coils used in deriving the experimental correlations were half-circuited.

7.8.1.1. Dry Coil Tests.

The analysis method prescribed by ARI Standard 410-81 aims to deliver a set of plots which may be used as graphical engineering aids for predicting coil performance. For the family of coils tested, the sensible cooling heat transfer performance is described by the following set of plots:

i. Dry air-side thermal resistance (R_{aD}) as a function of standard air velocity ($V_{a,\,std}$), both plotted on logarithmic axes. The appropriate plots for the 1, 2 and 4 row coils are shown in figure 7.20. These may be correlated using a linear relationship:

$$\ln R_{aD} = a + b \ln V_{a,std} \tag{7.8.3}$$

where, for a one-row coil,
a = -1·658142, b = -0·427785;
for a two-row coil,
a = -1·719165, b = -0·579415;
for a four-row coil,
a = -1·702484, b = -0·791284.

ii. The sum of the dry air-side and metal thermal resistances (R_{aD} + R_{mD}) as a function of dry air-side resistance (R_{aD}), shown in figure 7.21 as a composite plot for the family of coils tested. This relationship is redundant in the computational methodology developed herein.

iii. Water-side film thermal resistance as a function of water velocity. The form prescribed by ARI Standard 410-81 is based on the McAdams equation; the equivalent plot for the present analysis is shown in figure 7.16.

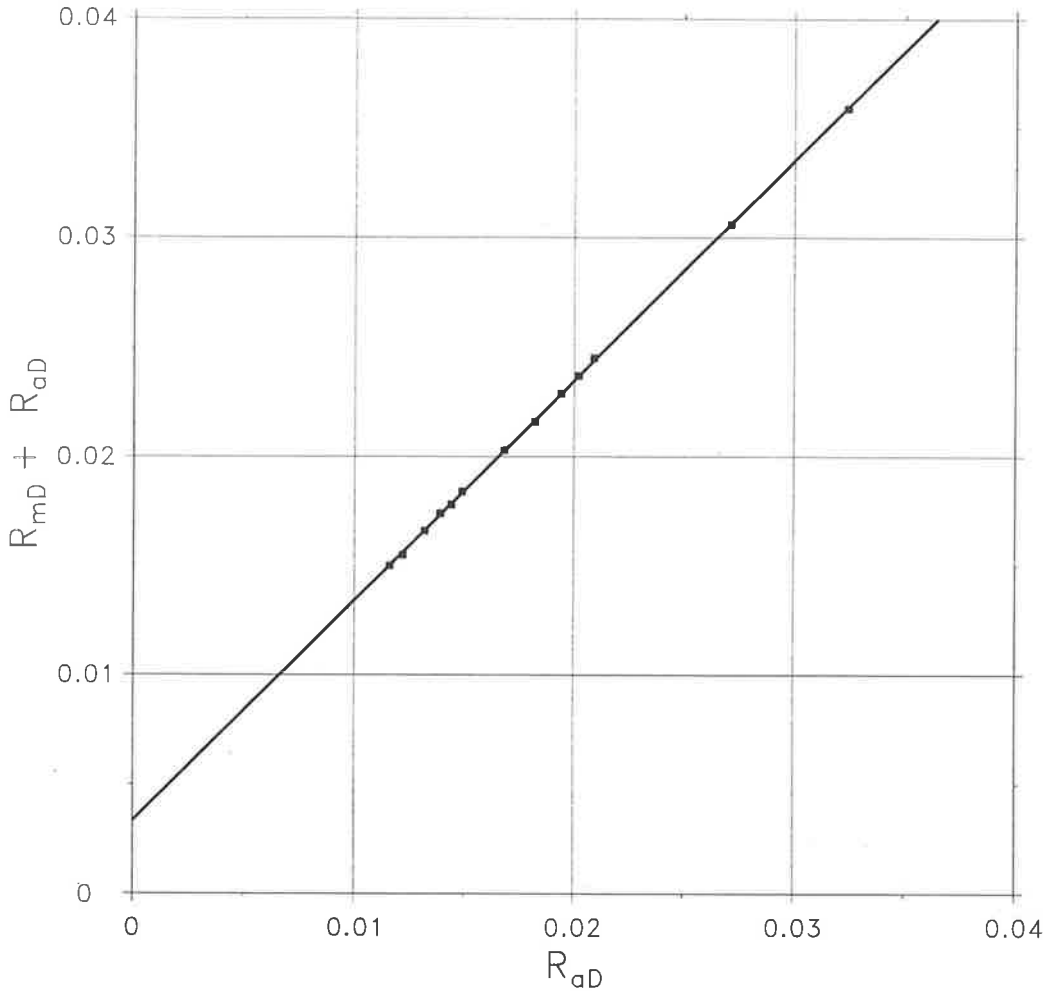


Figure 7.21. Sum of the dry air-side and metal thermal resistances plotted as a function of dry air-side resistance. Data reduction according to ARI Standard 410-81.

7.8.1.2. Wet Coil Surfaces.

In the AU method, the air-side heat transfer coefficient ($h_{c,ow}$) is correlated in the form of the dimensionless grouping $St.Pr^{2/3}$ as a function of Reynolds number (Re_d), using equation (7.5.19). Plots for 1, 2 and 4 row 6 fpi coils are shown in figure 7.22. The appropriate constants for use in equation (7.5.19) are,

for a one-row coil,

a = -0·673770, b = -0·376074;
for a two-row coil,
a = -1·009592, b = -0·244951;
for a four-row coil,
a = -1·194125, b = -0·181296.

For the same coils the sum of the metal and condensate resistances ($R_m + R_{ow}$) are shown plotted as a function of the wet air-side thermal resistance ($R_{c,ow} = 1/h_{c,ow}$) in figure 7.23. The experimental results have been fitted with a second-order polynomial of the form

$$R_m + R_{ow} = a + bR_{c,ow} + cR_{c,ow}^2 \qquad \textbf{(7.8.2)}$$

where,
for a one-row coil,
$a = 4{\cdot}125333 \times 10^{-3}$, $b = 1{\cdot}488645 \times 10^{-1}$, c = -2·870635;
for a two-row coil,
$a = 4{\cdot}076190 \times 10^{-3}$, $b = 9{\cdot}146160 \times 10^{-2}$, c = -3·657706;
for a four-row coil,
a = 0·003581, b = 0·0129560, c = 0·0.

Caution is required in using these functions to extrapolate beyond the range of the experimental data; the same applies to the piecewise linear correlations of Sekhar (1990). As has been mentioned previously, a more precise formulation of the functional relationship between the various thermal resistances awaits a better understanding of the physics involved.

7.9. Summary.

Although largely neglected in much conventional air conditioning practice, the performance of the cooling coil or coils may correctly be regarded as the major factor determining whether an air conditioning system will be capable of maintaining design comfort conditions, while minimizing energy consumption. An understanding of the behaviour of a cooling coil under varying load conditions is thus an essential component of any modelling strategy. The current chapter commenced with a review of recent investigations into the dynamics of the air flow and condensation processes occurring within a cooling coil. Subsequently a model algorithm to predict the state of the working fluids leaving a cooling coil was developed, and this was used as a framework to implement and utilize two coil rating methods; the ARI method for dry coil surfaces, and the AU method for wet coil surfaces. Procedures for calculating the air and water-side pressure drops in cooling coils were also described, and a critique of current understanding of the functional dependencies of air-side pressure drop was presented. The chapter concluded by describing the application of the previously developed coil rating procedures to the analysis of test results for a series of coils.

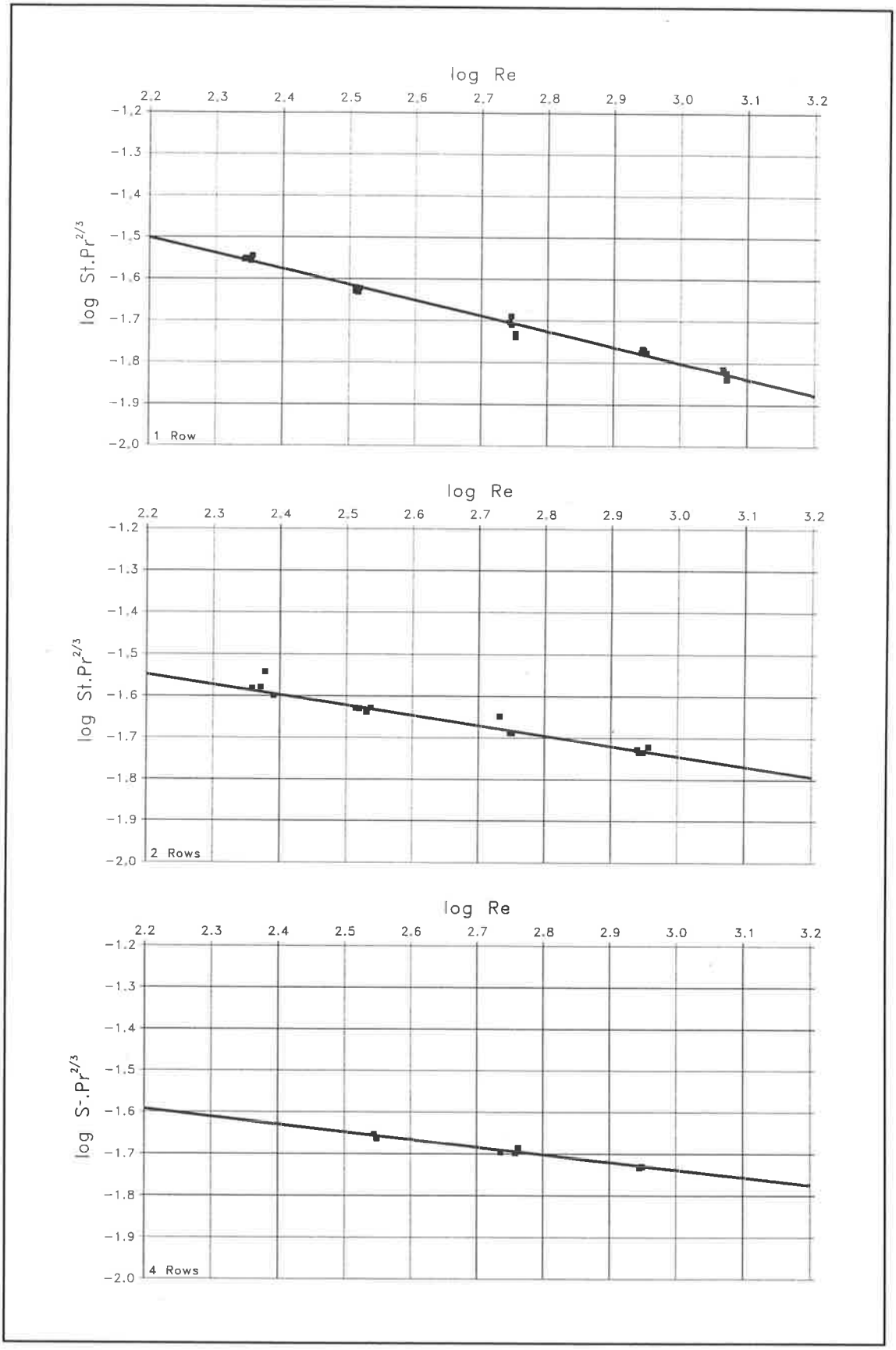


Figure 7.22. Colburn j-factor as a function of Reynolds number for a family of 6 fpi coils. Fully wet test results processed using AU method.

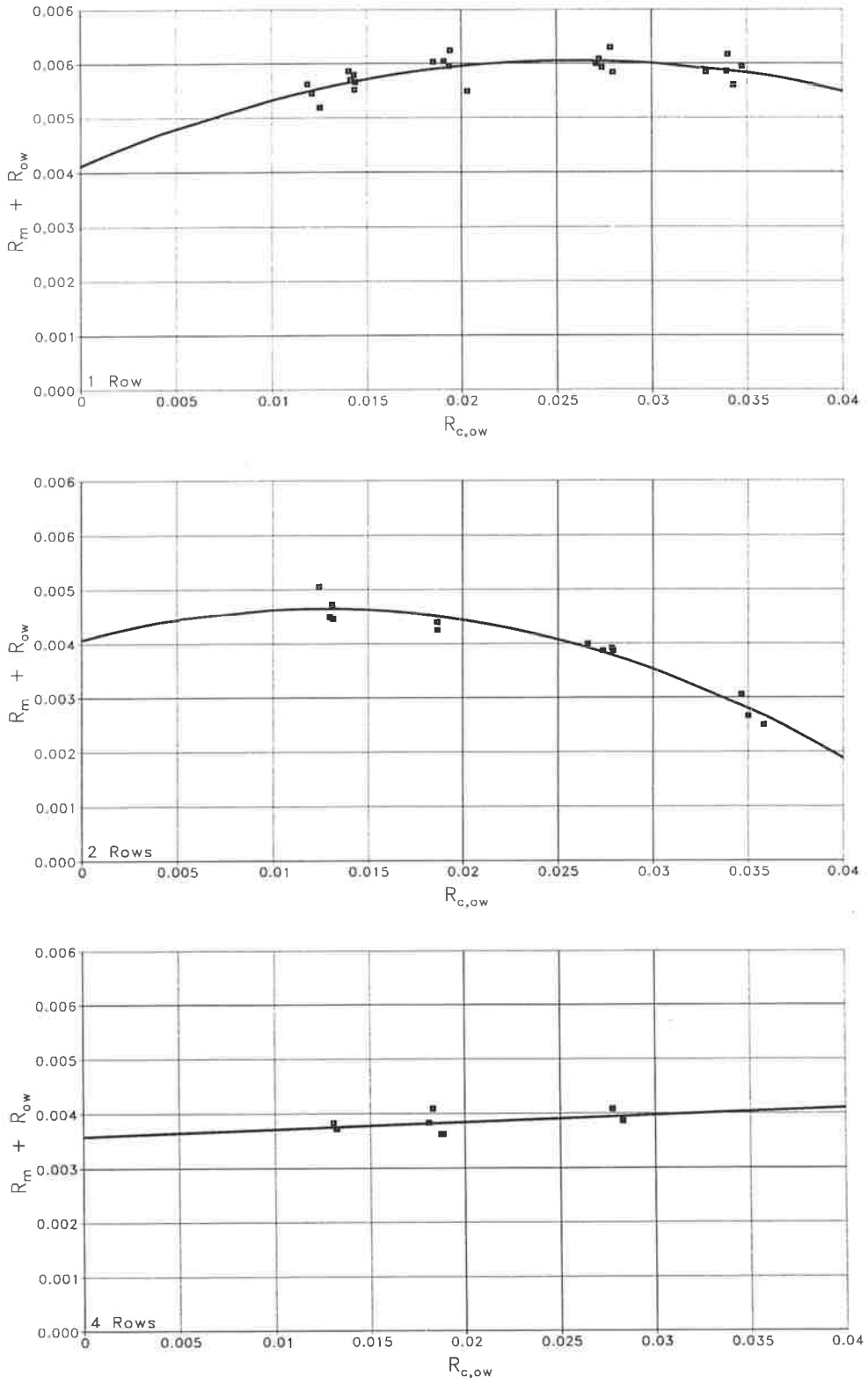


Figure 7.23. Sum of the metal and condensate thermal resistances plotted as a function of wet air-side thermal resistance for a family of 6 fpi coils. Results derived using the AU method.

Chapter 8. Simulation of Composite Coil Banks.

The preceding chapter has described a set of algorithms which, when used in conjunction with suitable heat and mass transfer correlations, enable the user to compute the state of the coolant and moist air leaving a given coil, given the condition and flow rates of the same fluids at entry to the coil. Subsequently, an empirical set of heat and mass transfer correlations were presented for half-circuited fin-and-tube coils having depths of 1, 2 and 4 rows. The designer's options would of course be severely restricted if these simple coils defined the limits of his repertoire. Nevertheless, using the simple half-circuited coil as a basis, it is possible to extend greatly the range of circuiting options available to the designer. In the following, two extensions to the basic method are described. The first provides a means of relaxing the limitation to half-circuited coils to permit a coil having an arbitrary number of passes per circuit to be modelled. Secondly, the simple coil is used as an element within a more general structure which may be used to model and solve for composite coils of some complexity. The properties and applications of such a structure are considered both in terms of their physical implementation and by reference to the conceptually equivalent computer data structures.

8.1. Generalized Coil Circuiting.

The experimental tests used to derive the heat and mass transfer correlations employed half-circuited coils in all cases. In this circuiting arrangement, chilled water is fed to every second tube in the rearmost row of the coil. If the data are to be used to calculate the performance of a coil, the circuiting of the coil under consideration must be converted to that of an equivalent half-circuited coil. Sekhar (1990) has shown that this can readily be achieved by altering the number of passes per circuit to that appropriate to half-circuiting, while keeping the number of rows, the total number of circuits, and the active tube length per circuit (and hence the face area) constant. The resulting coil, which is used in the heat and mass transfer calculations, is referred to as a 'virtual' coil, and is derived from the original coil by the following operations:

$$L_t' = L_t \frac{N_{tr}}{2N_{ct}} \tag{8.1.1}$$

$$N_{pc}' = \frac{2N_{ct}N_{pc}}{N_{tr}} \tag{8.1.2}$$

$$N_{tr}' = 2N_{ct} \tag{8.1.3}$$

where

L_t is the tube length;
N_{ct} is the total number of circuits;
N_{pc} is the number of passes per circuit;
N_{tr} is the number of tubes per row.

The primed variables refer to the 'virtual' coil, and the unprimed variables refer to the original coil. It should be borne in mind that the 'virtual' coil is used only for the heat and mass transfer calculations; pressure drop for air and coolant through the coil must be calculated on the basis of the actual coil specifications.

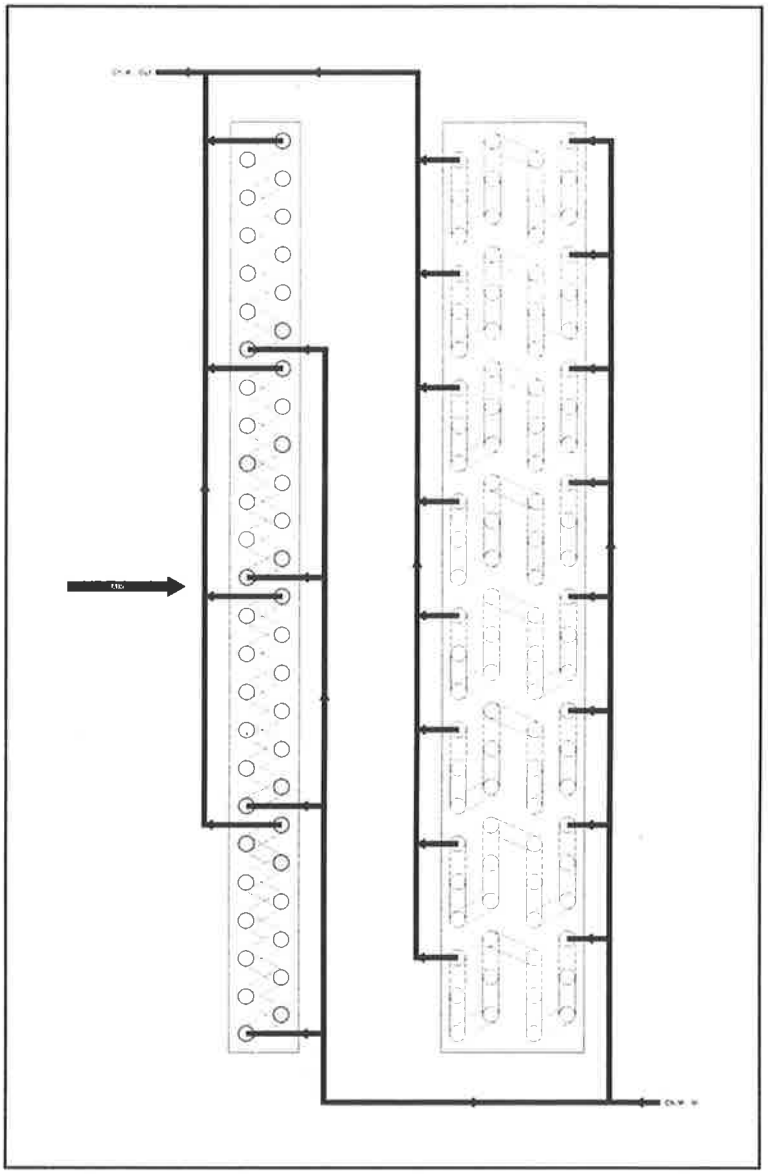


Figure 8.1. A six-rowed coil simulated using two coils with chilled water circuiting in parallel.

8.2. Data structures for Composite Coils.

Even with the constraint of half-circuiting removed, the range of coil configurations which may be simulated using the simple 1, 2 and 4 row coils described previously is severely limited. Such simple coils may however be used as elementary building blocks to construct composite coils of some complexity. There are a number of reasons why we may wish to do this. Possibly the most obvious use for such a facility is to permit coils of arbitrary depth to be simulated. Three-row coils are frequently used in our design methodology[88]. Where constraints on space are tight, it may be necessary to specify coils having six rows of depth in order to achieve the required sensible cooling. Even deeper coils are not uncommonly specified in conventional practice, often in the mistaken belief that such coils promote deep dehumidification. A further use for such a facility, and one which is specific to the LFV/HCV design methodology, arises in connection with the *staging* process. Staging refers to the process whereby coolant velocity through a coil is maintained above a lower threshold by deactivating circuits, and thus reducing the active surface area as the load

---

88 Simulating coils as composite structures of course increases the computational effort involved in solving for a set of operating conditions. Experimental work is currently being undertaken to add the 3-row coil to our database as an elementary building block.

decreases. The underlying principles have been described in chapter 5. Chapter 12 considers control strategies to effect the activitation and deactivation of circuits under dynamic loading conditions; applications of the staging process are considered in chapter 13. The current section addresses the problem of simulating the staging process within a computational model. The requirements of a data structure to model the range of configurations with which we will be involved may best be understood by considering some examples.

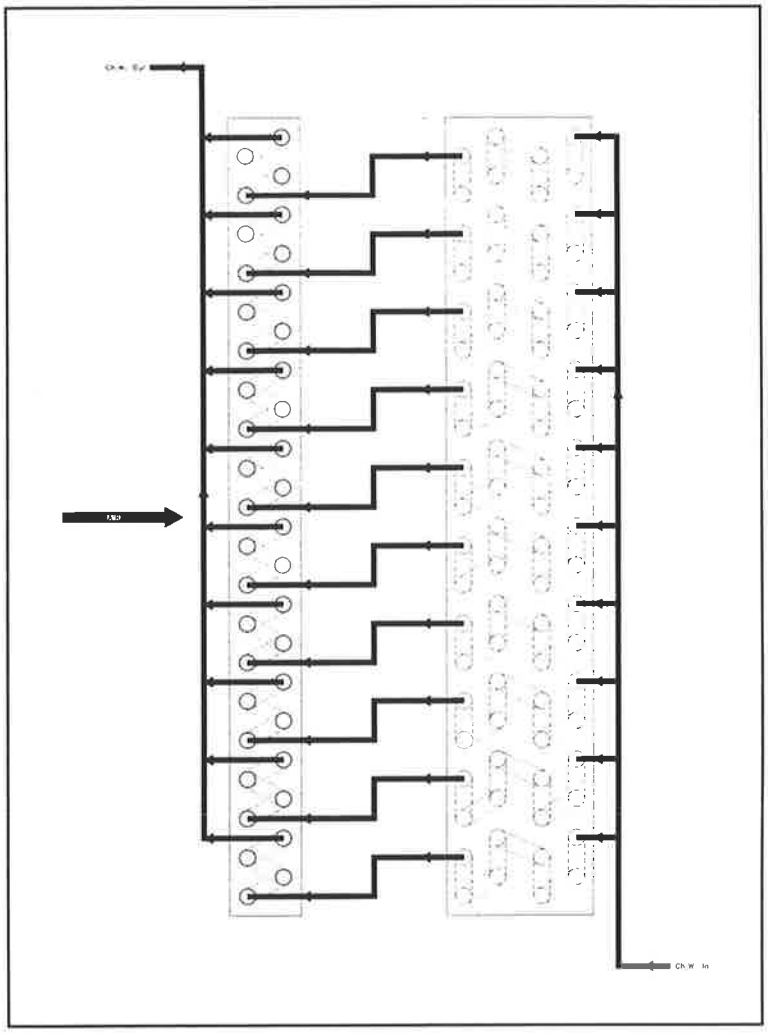


Figure 8.2. A six_rowed coil simulated using two coils with chilled water circuiting in series.

Consider first the case of simulating a 6-row coil having 12 passes per circuit. The only constraint imposed on the coil by this circuiting arrangement is that the face of the coil must be an even number of tubes in height. If the desired coil is 24 tubes high, maximum computational efficiency may be achieved by structuring the coil model as a composite of a 2-row and a 4-row coil, each of 12 passes per circuit, and arranged *in series* in terms of the air flow, and *in parallel* in terms of the coolant flow, as shown in figure 8.1[89]. It is of course axiomatic that both of the elementary coils must present the same face dimensions to the air stream. A solution for the composite coil is readily obtained by solving for the elementary coil upstream with respect to the air stream, setting the coil-on condition for the downstream coil equal to the coil-off condition of the upstream coil, and thus solving for the downstream coil. Such a strategy is clearly not feasible if the desired coil is 20 tubes in height. This case may be simulated using the circuiting configuration of figure 8.2, in which we combine a 4-row coil having 8 passes/circuit with a 2-row coil having 4 passes/circuit. As before, the elementary coils are arranged *in series* with respect to the air stream. However, they must also be circuited *in*

---

89 The circuiting arrangements shown for the four-row coil components in figures 8.1 and 8.2 are far from optimal, and should be regarded as conceptual only.

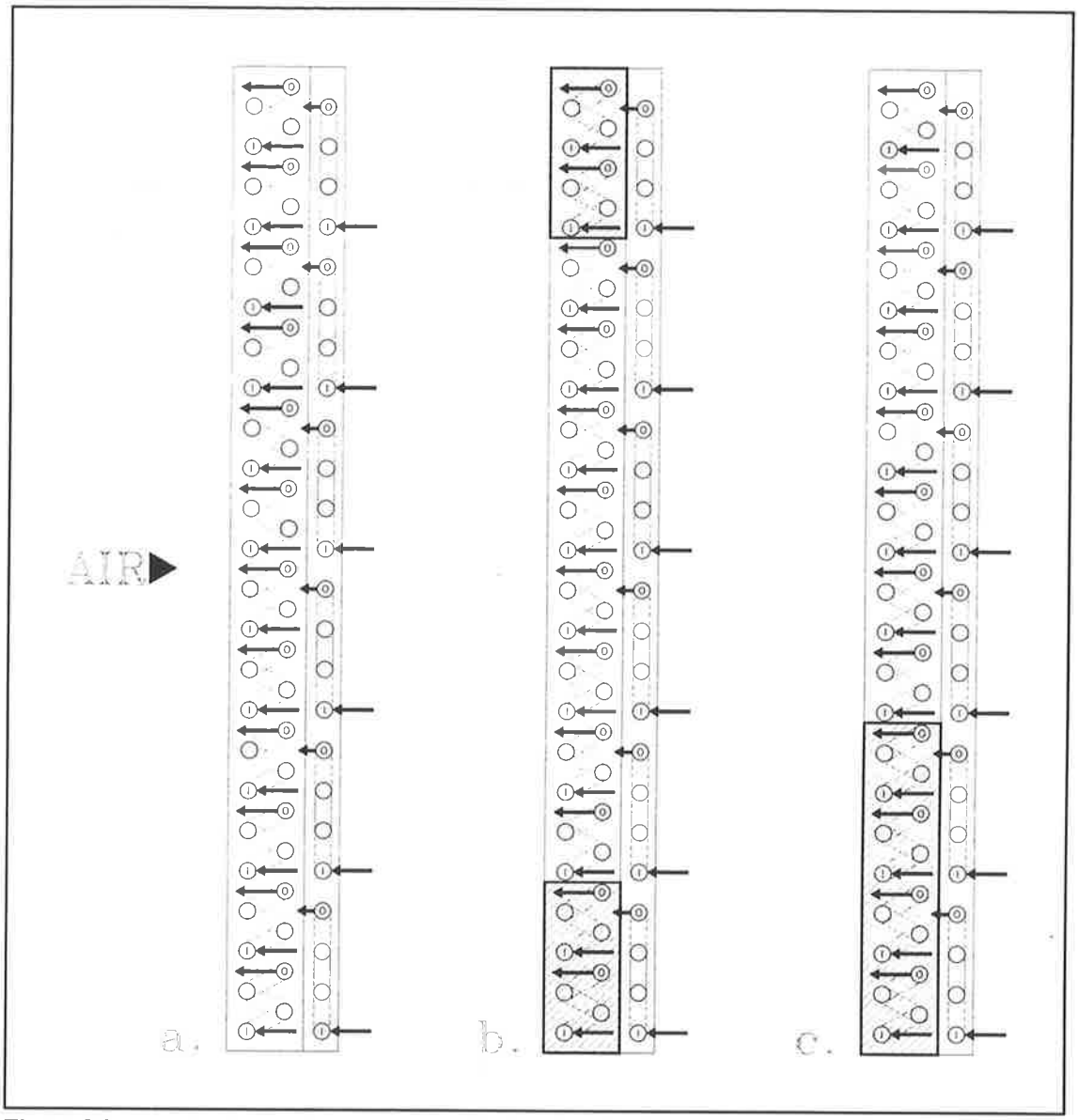


Figure 8.3. Staging of a 3-row coil. Deactivated circuits are shown hatched. Captions refer to the text.

series with respect to the coolant flow. In view of the fact that we are considering a *counterflow* situation, the solution procedure for this latter case is considerably more complex than that for the first case considered. In order to specify the coil-on condition to the downstream coil, it is necessary to solve for the upstream coil. However, the temperature of the chilled water supplied to the upstream coil will not be known until a solution has been obtained for the downstream coil. In other words, an iterative solution procedure is called for. In order to model coils of arbitrary depth and coil configuration, it is necessary then to provide the ability to combine elementary coils either in series or in parallel with respect to the coolant flow[90].

---

90 Strictly speaking, it is only necessary to provide the ability to combine elementary coils in series with respect to the coolant flow. However, since the case of parallel circuiting can be solved much more efficiently, such an arrangement should be used *where possible*.

The second example concerns a case of staging. Consider a 3-row coil, 24 tubes high. This coil may be simulated as a composite of a 2-row and a 1-row coil circuited in parallel, as shown in view (a) in figure 8.3. Suppose now that the staging strategy requires that four circuits be deactivated from the upstream[91] two rows at part load, as shown in view (b). It is in fact likely that the circuits will be deactivated symmetrically as shown, in order to minimize inhomogeneities in the air flow downstream of the coil. From a thermodynamic viewpoint the arrangement of view (c) is equivalent. It is now clear how the staging process may be handled within a computational model of a composite coil structure. The computational equivalent of view (c) is shown in figure 8.4. The upstream elementary coil is now represented by not one, but two elementary coils, one of which contains the circuits which remain active after the stage change, while the other contains the circuits which have been deactivated. The downstream coil has likewise been split into two coils, the respective face heights of which match those of the active and deactivated upstream coils[92]. A solution is obtained for the two upstream coil components, the air passing through the deactivated portion without

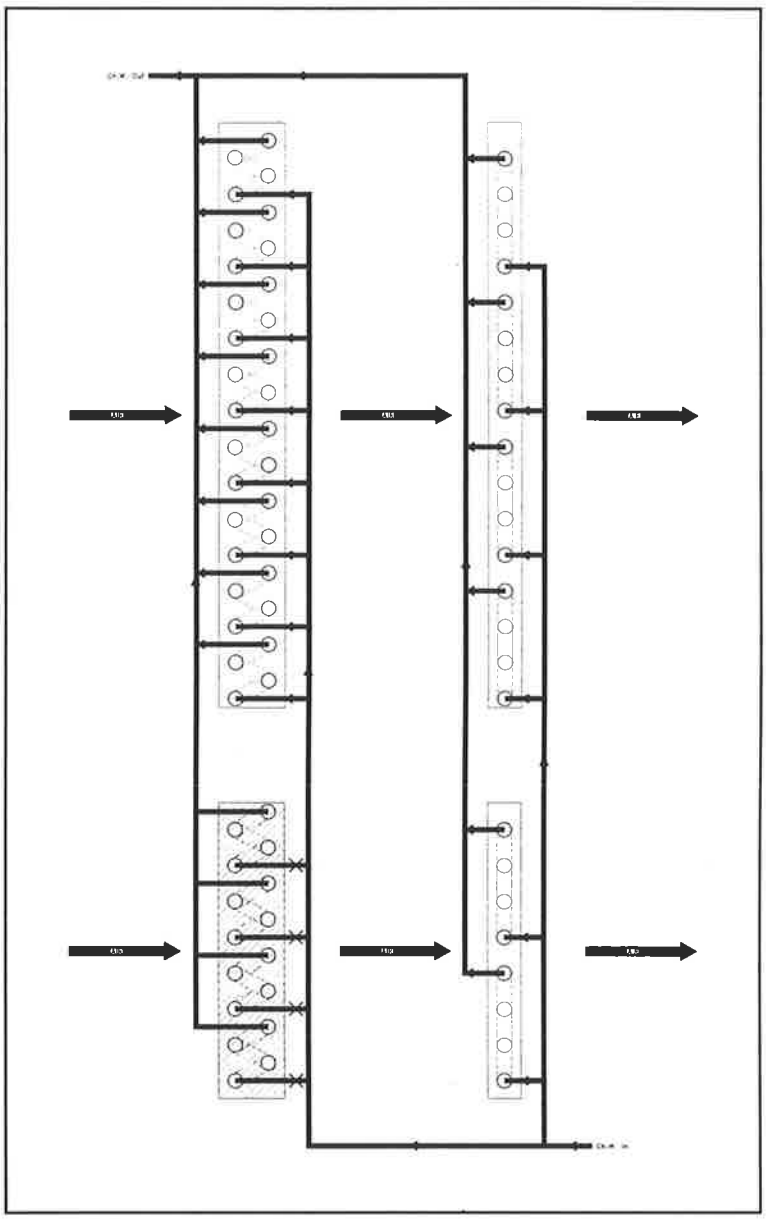


Figure 8.4. Computational representation of a 3-row staged coil with four circuits deactivated.

---

91 The staging algorithm described in section 8.3.3 requires that any deactivated circuits be located within the first elementary coil encountered by the airstream.

92 The circuiting arrangement specified must be such as will make it possible to match the face heights of the coil components.

change. This defines the coil-on condition for the components of the downstream coil, for which the coil-off conditions can now be calculated, and the two streams psychrometrically mixed downstream. The model must therefore provide the ability for each elementary coil to be represented by two smaller coils having the same circuiting, one of which coils may be deactivated or, in the case where staging is not in effect, *null*. It is important to realise that at least one row remains active throughout the entire height of the coil following a changeover. In other words, no air is bypassed. The benefits conferred by bypassing air are minimal, at best.

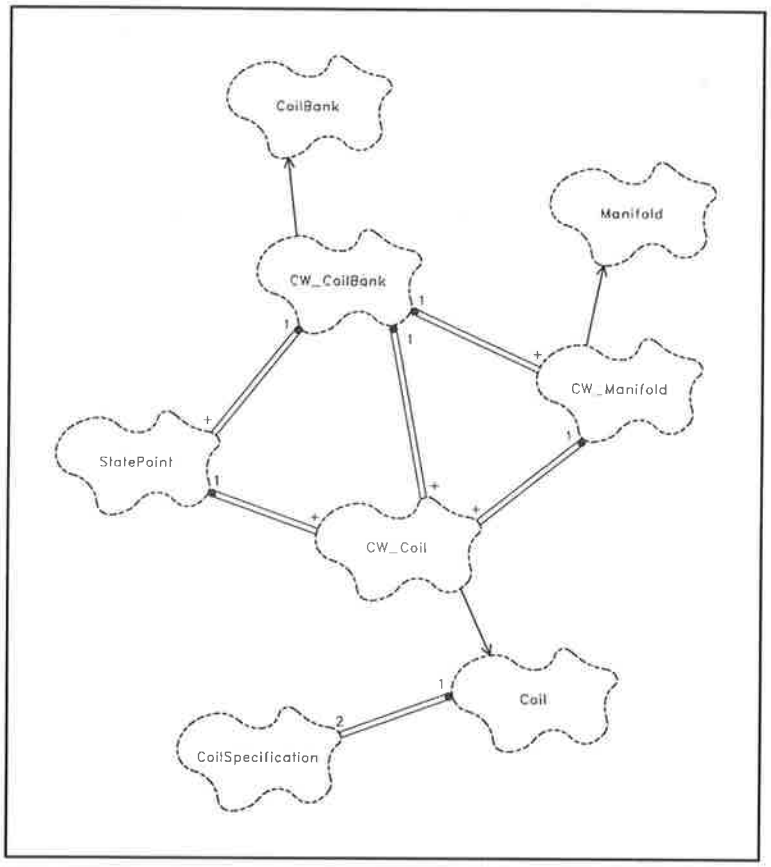


Figure 8.5. Class diagram showing relationships between the various classes from which a coil bank is constructed.

The generalized data structure which we use to simulate composite coils is referred to as a *coil bank*, and implemented by defining a C++ class *CoilBank*. Objects of this class operate on data structures composed of entities belonging to three more basic classes:

i. Class *Coil* provides a set of variables and operations to simulate the simple coils which are the elementary building blocks for the coil bank.

ii. Class *StatePoint* provides variables and operations referring to the state of the air at a point in the structure.

iii. Class *Manifold* conceptually refers to the hardware item of the same name; the purpose of this class is to provide variables and operations to access and manipulate the state of the coolant at a point in the coil bank.

Classes *CoilBank*, *Coil* and *Manifold* are *abstract* base classes. To complete the implementation, information must be supplied concerning the properties and thermal behaviour of the coolant used. In the case of chilled water the additional information and operations are supplied by the derived classes *CW_CoilBank*, *CW_Coil* and *CW_Manifold*. The following discussion will be restricted to chilled water systems, and

a distinction will not always be drawn between the operations supplied by the base classes, and those specific to the derived classes. It should be borne in mind however that most of the operations described are provided by the base class, and are accessible for use in simulating DX coils, with the provision of a suitable superstructure. The relationships between these classes are as shown in figure 8.5. The properties of the various classes will be described below.

8.2.1. Class *Coil*.

Class *Coil* makes use of a more fundamental class, *CoilSpecification*, which stores primary and derived dimensional data for the coil, together with the tube and fin thermal conductivities. The internal structure of class *CoilSpecification* need concern us little here. Basically, a set of primary dimensions are supplied as arguments to the constructor for the class[93], and the derived dimensional data calculated using the relationships of section 7.1. An additional member variable which will be of importance when we come to discuss the manner in which coil banks are assembled is the boolean variable `CoilSpecification::cm`[94]. The constructor for class *CoilBank* will assemble the coil components in sequence from the front (the side facing the air stream) of the coil. Specifying a value of *true* for `CoilSpecification::cm` indicates that this coil is to be circuited in parallel with its predecessor (figure 8.1); if *false* is specified, it will be circuited in series (figure 8.2). If *false* is specified, a further boolean variable `CoilSpecification::am` is set to true if a real manifold is intended, or *false* if the manifold is conceptual, as in figure 8.2. In the latter case, an extra return bend must be included in coolant pressure drop calculations, if this parameter is not to be underestimated.

The constructor for class *Coil* accepts a reference to an object of class *CoilSpecification* as an argument. A second copy of this object is created internally by the constructor. The internal copy differs from the original in that it contains the dimensional information relevant to an equivalent half-circuited coil, derived from the original as described in section 8.1. This 'virtual' coil is used internally for all heat and mass transfer calculations. Performing a changeover (section 8.3.3), which involves deactivating circuits, will further

---

93 By default, the tube and fin dimensions used are those pertinent to the family of plate fin and tube coils described in section 7.8.1. Member functions are provided to change these parameters, although it should be borne in mind that the heat and mass transfer relationships embodied in class *Coil* are based on this family of coils. It is clearly desirable that a more flexible arrangement be implemented in future, and that full support for helically-wound coils (these are partially supported at the moment) be provided. The constructor accepts the following arguments, some or all of which may be omitted, inwhich case the defaults are used:

Number of rows (N_r)	:	Default = 4.
Passes per circuit (N_{pc})	:	Default = 4.
Tubes per row (N_{tr})	:	Default = 24.
Length of tube (L_t)	:	Default = 1000 mm.
Fin density ($\dot{N}_f$)	:	Default = 6·18 fpi.

94 Common Manifold.

modify this internal copy. However, the original object remains unchanged, and describes the actual physical coil from which the internal copy has been derived.

The constructor for this class takes a second argument which determines whether the object as created is to have all its circuits fed with chilled water (*true*), or none fed (*false*). The constructor is thus declared as

```
Coil::Coil (CoilSpecification coil,
            boolean f_flag);
```

An object of class *Coil* created with the second argument set to *false* is a *null* coil. Internally a *null* coil is identified by setting the total number of circuits (N_{ct}) equal to zero. *Null* coils pass no airflow and are bypassed by the coolant flow; they therefore have no cooling capacity. *Null* coils must be distinguished from *deactivated* coils, which have a non-zero number of circuits, none of which is fed with coolant. When the data structures comprising a *CoilBank* are created (section 8.3.1), a *null* coil is created corresponding to every non-*null* coil. These in effect serve as placeholders for use during the changeover process (section 8.3.3).

The entering air state is accessed by means of a pointer to an object of class *AirState* (this pointer may be set after the object has been created); the leaving air state is stored in an internal variable. The operations which may be performed on an object of this class are:

i. Activating or deactivating the coil; a deactivated coil has no circuits active.

ii. Changing the air flow and coolant flow.

iii. Calculating air pressure drop.

iv. Calculating coil thermal performance for the current set of operating conditions (problem FP). A member function `Coil::Solve ()` is provided for this purpose. If the coil is deactivated, the leaving air state is simply set equal to the entering air state.

v. Accessing and setting coil operating parameters, as appropriate.

The algorithms used to perform operations (iii) and (iv) have been described in detail in the preceding chapter. In order to use this base class, derived classes must provide **protected** member functions which will compute and return:

i. The tube-side coolant film heat transfer coefficient.
ii. Temperature of the coolant entering and leaving the coil.

In addition to the above, the derived class *CW_Coil* adds operations to:

i. Set the entering water temperature.
ii. Calculate and return the water pressure drop using the algorithms of section 7.6.

8.2.2. Class *Statepoint.*

Classes *StatePoint* and *Manifold* provide facilities to store and update the properties of the air and coolant respectively at various points in the coil bank[95]. More importantly perhaps, they serve as links, channelling the working fluids from one point in the structure to another. Internally, class *StatePoint* is represented by the following member variables:

`a` The psychrometric condition of the air at the state point, represented by an object of class *AirState.*

`ce` A singly-linked list of pointers to a set of coils, the air off of which is psychrometrically mixed to produce the condition at the point.

`cl` A singly-linked list of pointers to a set of coils which are fed with air from this state point.

`ma` The mass flow of air ($\dot{m}_a$) at the state point.

`p` The pressure at the state point. This is gauge pressure referenced to an arbitrary datum.

Schematically, an object of class *StatePoint* may be linked into a structure as shown in figure 8.6. Class *StatePoint* supports the following operations:

i. Reading and setting the values of member variables, as appropriate.

ii. Attaching a coil feeding into the state point. The coil is appended to list `StatePoint::ce`.

iii. Attaching a coil fed from the state point. The coil is appended to list `StatePoint::cl`. Since objects of class *Coil* do not store the entering air state internally, but store a pointer to an external object of class *AirState*, this pointer is set to address member variable `StatePoint::a` of the appropriate object of class *StatePoint.*

iv. A member function `StatePoint::Solve ()` is provided which performs the following operations in sequence:

   a. The condition of the air at the state point is found by psychrometrically mixing the coil-off condition of the air leaving the coils feeding into the state point (referenced by list `StatePoint::ce`), weighted according to the mass flow through each.

---

[95] The *physical* or *conceptual* point in a coil bank described by an object of class *StatePoint* is referred to in the following as a *state point.*

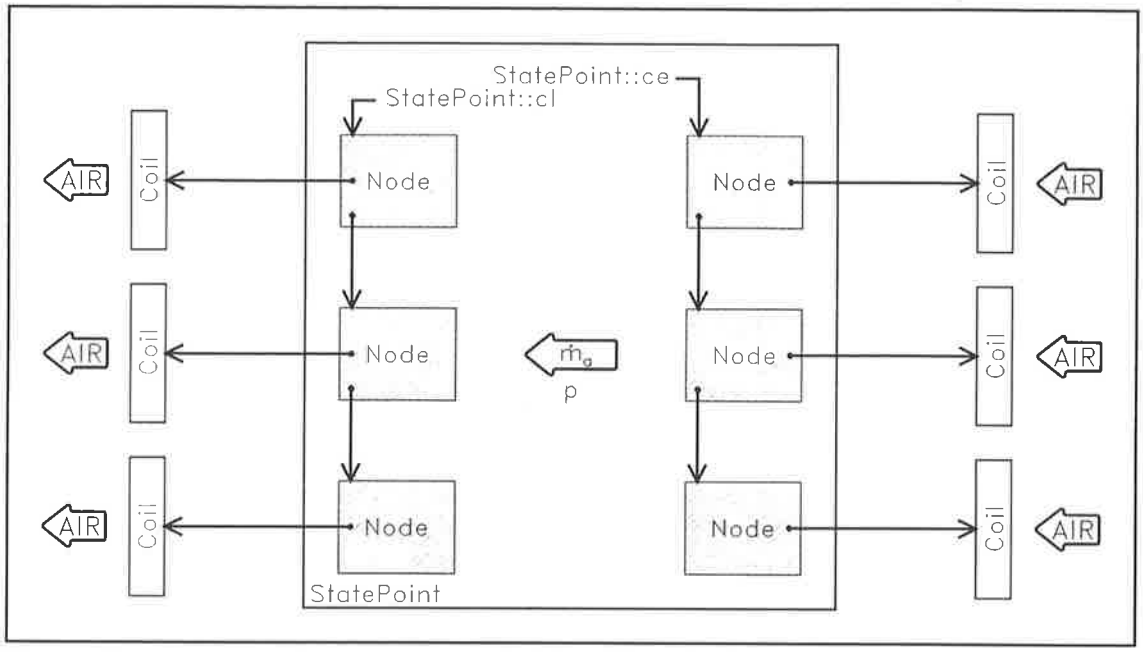


Figure 8.6. Internal structure of an object of class *StatePoint*.

b. Member variable `Coil::Solve ()` is called for each *Coil* on list `StatePoint::cl`.

v. An additional member function `StatePoint::Update ()` psychrometrically mixes the air-off conditions for the coils referenced by list `StatePoint::ce` to find the condition of the air at the state point (`StatePoint::a`), without initiating any further action on the part of the coils downstream.

vi. A member function

```
void StatePoint::ChangeAirFlow (const double m = 0.0);
```

is provided to update the mass air flow through an object of class *StatePoint*. This function takes an optional argument `m`. If list `StatePoint::ce` is empty, variable `StatePoint::m` is set equal to the value specified by argument `m`; otherwise it is evaluated by summing the mass flows through the coils referenced by list `StatePoint::ce`. The mass air flow rates through the coils referenced by list `StatePoint::cl` (if any) are subsequently set in proportion to their face areas.

8.2.3. Class *Manifold*.

Class *Manifold* essentially performs the same function in respect of the coolant stream as class *StatePoint* does in respect of the air stream[96]. However, whereas the external

---

[96] A specific point in the coolant circuit may be described by an object of class *Manifold*. Frequently such a point will correspond to an actual manifold in the physical implementation of the coil bank. In the situation represented in figure 8.2 an object of class *Manifold* would be inserted between the four-row and two-row coils used to simulate a six-row coil, although clearly no physical manifold

working fluid is always air in the examples we are considering, the data structures provided to handle the coolant streams must be sufficiently flexible to accommodate chilled water or refrigerant[97]. For this reason, class *Manifold* is intended to be used as a base class. The internal structure of the class closely resembles that described above for class *StatePoint*. The following member variables are defined:

`ce` A singly-linked list of pointers to coils which feed into the manifold.

`cl` A singly-linked list of pointers to coils which are fed from the manifold.

`m` The mass flow of coolant through the manifold.

Member functions are provided to perform the following operations:

i. Read the mass coolant flow.

ii. Append pointers to objects of class *Coil* to list `Manifold::ce`, or list `Manifold::cl`, as required.

iii. A member function `Manifold::UpdateMassFlow (m)` is provided which is strictly analogous in respect of its operation to function `StatePoint::ChangeAirFlow (m)` described above. The mass flow rates through the coils fed from the manifold are set in proportion to their internal flow areas.

Note that none of the variables or functions described above define or make reference to the properties of the working fluid. Such functions are provided as required by a derived class. Class *CW_Manifold* adds facilities specific to *chilled water*. Member variables `CW_manifold::t` and `CW_manifold::p` specify the temperature and pressure respectively at the manifold. Functions are provided to set and access these variables. Additional member functions are provided to:

i. Update the water temperature at the manifold:

```
void CW_Manifold::UpdateTemperature (const double t = 0.0);
```

Three conditions are possible:

a. Some or all of the coils referenced by list `Manifold::ce` are active. In this case the temperature at the manifold is set equal to the average value of the

---

is present. In the discussion which follows, the term manifold will be used to refer to a point in the coil bank which is represented in the model by an object of class *Manifold*, regardless of whether a *physical* manifold is present at the point, or whether the use of the term is simply a *conceptual* convenience.

97 Steam is also frequently used as a tube-side working fluid in heating applications. Such applications could be handled with elementary modifications to the member functions of class *Coil*.

temperature of the water leaving the *active* coils, weighted according to the number of active circuits in each.

b. None of the coils referenced by list `Manifold::ce` are *active*; the water temperature is set equal to that at entry to the coils which feed into the manifold.

c. List `Manifold::ce` is *empty*. In this case the water temperature is specified by an optional argument (`t`) to the function.

Having found the water temperature at the manifold, the entering water temperature for each coil referenced by list `Manifold::cl` is set accordingly.

ii. Update the water pressure at the manifold:

```
void CW_Manifold::UpdatePressure (const double p = 0.0);
```

As above, three conditions are possible:

a. Some or all of the coils referenced by list `Manifold::ce` are *active*. In this case the water pressure drops for the *active* coils are calculated (these will be identical) by invoking the appropriate member function for class *Coil*, and the water pressure at the manifold set equal to that of the water leaving the coils.

b. None of the coils referenced by list `Manifold::ce` are *active*; the water pressure is set equal to that at entry to the coils which feed into this manifold.

c. List `Manifold::ce` is *empty*. In this case the water pressure is specified by an optional argument (`p`) to the function.

Having found the water pressure at the manifold, the entering water pressure for each coil referenced by list `Manifold::cl` is set accordingly.

8.3. Operations on Composite Coils.

The structure of and uses to which class *CoilBank* may be put are perhaps best understood by considering the member variables and operations which this class provide. The attributes of the class are embodied in the following member variables:

i. A set of doubly-linked lists of pointers:

`cl` Each node on the list points to a singly-linked list of pointers to objects of class *Coil*.

`sl` Each node on the list points to a singly-linked list of pointers to objects of class *Statepoint*.

`mf` Each node in the list points to an object of class *Manifold*.

The structure of these lists may best be understood by reference to figure 8.7, while the manner in which they are created is described in section 8.3.1. These structures permit composite coils of considerable complexity and generality to be constructed and operated on.

ii. A set of singly-linked lists of pointers. A pointer to each object of class *Coil*, *Manifold* or *Statepoint* is inserted onto one of three lists as it is created, as appropriate:

`C` Objects of class *Coil*.
`S` Objects of class *StatePoint*.
`M` Objects of class *Manifold*.

These lists perform a housekeeping rôle. They are used by the destructor for class *CoilBank* for deleting the objects which they address.

iii. Objects of class *StatePoint* referring to the air state upstream (`CoilBank::S_in`) and downstream (`CoilBank::S_out`) of the coil.

iv. Pointers to objects of class *Manifold* describing the state of the coolant entering (`CoilBank::M_in`) and leaving (`CoilBank::M_out`) the *CoilBank*. The objects themselves are referenced by list `CoilBank::mf`.

v. The following additional member variables are provided, each of type **double**:

`A_face`	Face area (m²).
`ma`	Mass flow of air (kg/s).
`Qa_act`	Actual air volume flow (L/s).
`Qa_std`	Air volume flow (L/s) under ASHRAE conditions.
`Va_act`	Actual air face velocity (m/s).
`Va_std`	Standard air face velocity (m/s).
`apd`	Air pressure drop (Pa).
`qt`	Total refrigeration capacity (kW).
`qs`	Sensible refrigeration capacity (kW).
`ql`	Latent refrigeration capacity (kW).

Class *CoilBank* is an abstract class in that it declares two pure **virtual** functions which must be defined by a derived class. These perform the following functions:

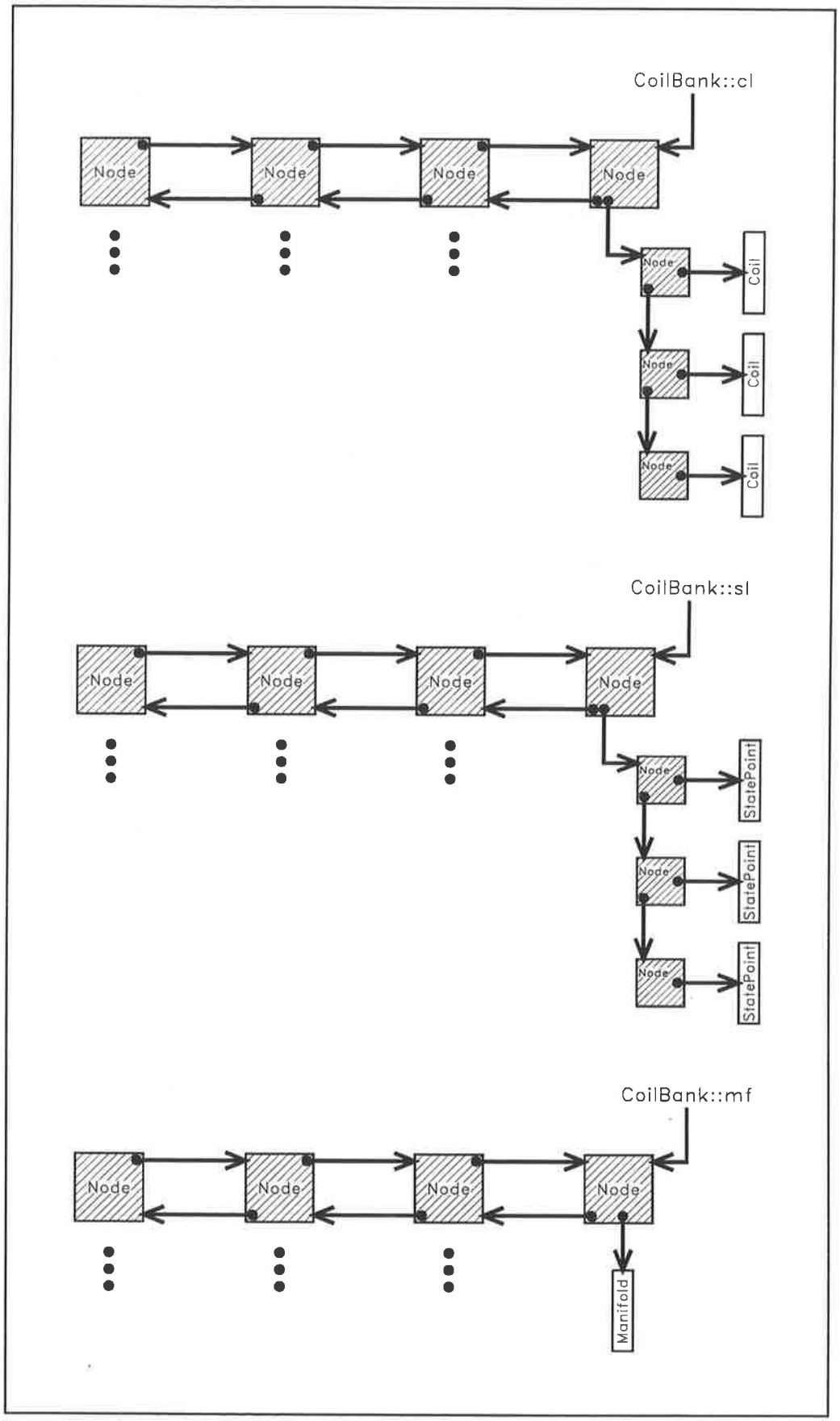


Figure 8.7. Elementary component structures for an object of class *CoilBank*.

i. Function

```
virtual Coil* CoilBank::CreateCoil (CoilSpecification& c,
                                    boolean f_flag) = 0;
```

creates and returns a pointer to an object of the appropriate subclass of *Coil*.

ii. Function

```
virtual Manifold* CoilBank::CreateManifold () = 0;
```

creates and returns a pointer to an object of the appropriate subclass of *Manifold*.

The derived class must also provide a public member function which provides a solution for the current set of operating conditions, and which overrides the pure **virtual** function

```
virtual void CoilBank::Solve () = 0;
```

The major member functions of class *CoilBank* are described in the following subsections. The following section contains a consideration of the facilities which must be added to implement a derived class using chilled water as a working fluid.

8.3.1. Constructor.

Class *CoilBank* provides three constructors which are distinguished by their calling sequences. The first takes no arguments, and merely creates an empty *CoilBank*. This is of little interest. The other two constructors create a *CoilBank* given an array of type *CoilSpecification*, the elements of which specify the elementary component coils from which *CoilBank* is to be constructed, in sequence from the front of the coil. These constructors differ only in the source from which the coil specifications are drawn. In the first instance the array is specified directly through the parameter list for the constructor. In the second case a reference to an object of class *TextSource*[98] is passed as a parameter to the constructor, and the array constructed from the information extracted from the associated input stream. Both constructors call an internal routine to actually assemble the *CoilBank* from the specified array, and it is the procedures embodied within this routine with which we shall be concerned here.

The task of creating the coil bank, which is essentially that of constructing the data structures of figure 8.7, is a three-stage process:

1. Stage 1: Creating and linking in the coils. In this stage the doubly-linked list `CoilBank::cl` is created.

2. Stage 2: Creating and linking in the state points. In this stage the doubly-linked list `CoilBank::sl` is created.

---

[98] See section 2.4.

3. Stage 3: Creating and linking in the manifolds. In this stage the doubly-linked list `CoilBank::mf` is created.

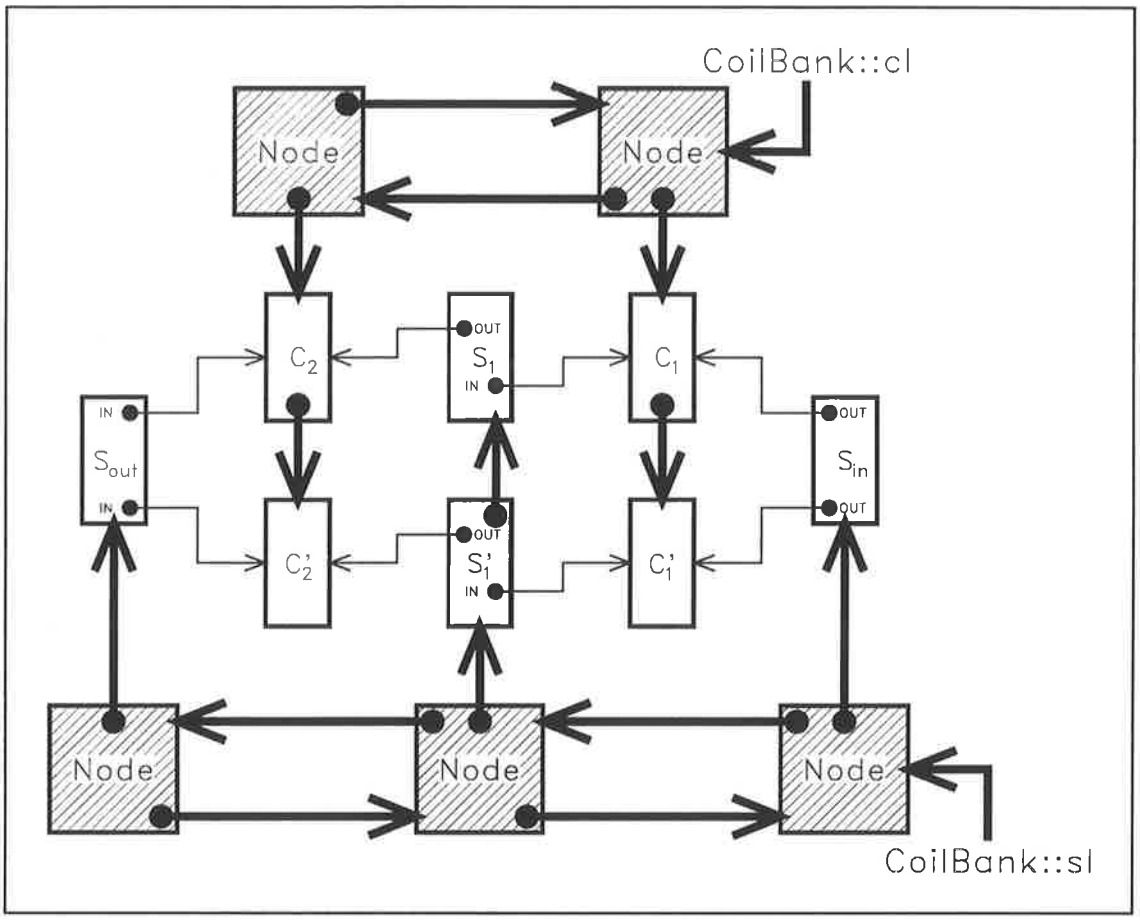


Figure 8.8. *Coil* and *StatePoint* structures used to represent the coil configurations of figures 8.1 and 8.2.

It would be superfluous to provide a detailed description of the procedures used in creating these data structures. However, an understanding of the data structures themselves, and the manner in which they relate to the physical coil, provides necessary background to the discussion which follows. Consider again our example of figures 8.1 and 8.2. The *Coil* and *StatePoint* structures required to represent both configurations are as shown in figure 8.8. Objects of class *StatePoint* are labelled S; objects of class *Coil* are labelled C. The air condition at state point S_{in} is the condition of the air at entry to the coil bank. The condition at state point S_{out} is the condition of the air leaving the coil bank, produced by psychrometrically mixing the air streams entering the state point. The arrowed lines which originate at the state points and terminate at the coils are to be interpreted as pointers from an object of class *StatePoint* to its associated objects of class *Coil*. The annotation IN or OUT indicates the direction of air flow *with respect to the state point*. Thus, those pointers which are stored on list `StatePoint::ce` are labelled IN; those which are stored on list `StatePoint::cl` are labelled OUT. In this and the diagrams which follow the intermediate nodes in the singly-linked lists, which are shown in figure 8.7, have been omitted for clarity. Note in particular that each elementary coil segment is represented in the model by two objects of class *Coil*. Thus, object C_1 is a computer representation of the 2-row

coil in figure 8.1. Object C_1' is a copy of C_1 in which the number of circuits, and the number of circuits fed, has been set equal to zero; a *null* coil in other words. The significance of this second copy of object C_1 will become clear when we consider the process of performing a changeover (section 8.3.3).

The data structures shown in figure 8.8 simulate the *air* side of the coil bank. The real difference between the models used to simulate the two physical configurations lies in the manner in which the list of manifolds is linked into the structure. Thus, the parallel-circuited configuration of figure 8.1 can be adequately handled using manifolds to represent chilled water state at the inlet and outlet points of the composite structure, as shown in figure 8.9. The diagrams of figures 8.9 and 8.10 are to be interpreted in an analogous manner to figure 8.8 (See the discussion above). Manifolds are labelled ***M***. The temperature and pressure of the water at manifold M_{in} are those of the water entering the coil bank, while the properties of the water at M_{out} are those of the water leaving the coil bank. The objects of class *Manifold* shown in figure 8.9 will represent to *physical* manifolds. To model the series-circuited configuration of figure 8.2 it is necessary to insert an additional *conceptual* manifold between the two elementary coils from which the model is constructed. This has no physical counterpart. The arrangement is shown in figure 8.10.

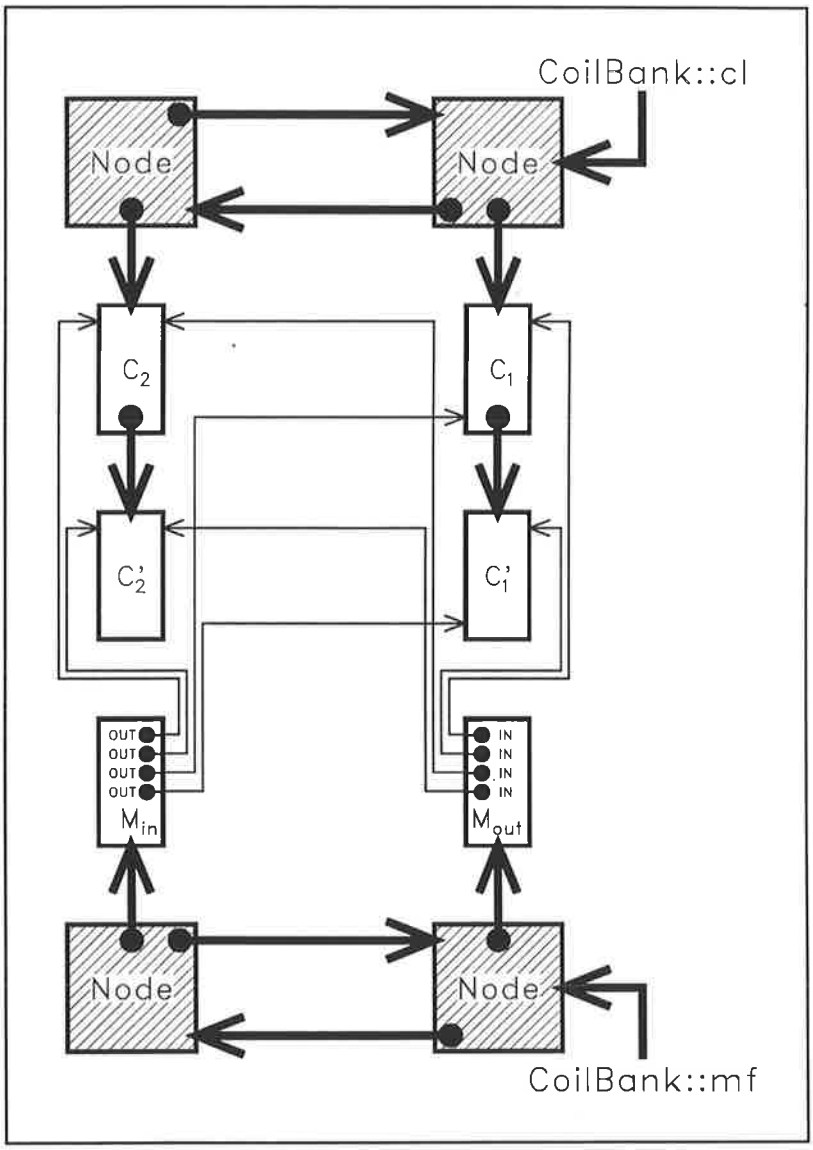


Figure 8.9. *Manifold* data structures used to model the chilled water circuiting configuration shown in figure 8.1.

8.3.2. Destructor.

The destructor for class *CoilBank* is responsible for ensuring that the structures from which objects of this class are constructed will be deleted in an orderly manner when the object is no longer needed. Deleting the data structures of figure 8.7 will cause the nodes of which they are composed to be deleted. The objects of classes *Coil*, *StatePoint* and *Manifold* which they reference are left intact. The singly-linked lists `CoilBank::C`, `CoilBank::S` and `CoilBank::M` are provided specifically for the purpose of deleting these objects. A pointer to each of the objects referenced by these lists must be extracted in sequence by invoking for instance, member function `Splist<Coil>::get ()` (section 2.3.2), and deleting the referenced entity.

8.3.3. Performing a Changeover.

The concept of *staging*, introduced earlier, implies the need to selectively deactivate circuits within the coil as the load is reduced. This action is referred to as performing a *changeover*. We will make frequent reference in the following to the example presented in figure 8.3. For computational purposes. the physical coil of this example was decomposed into a set of elementary coils arranged as shown in figure 8.4. This configuration may be represented using the data structures of figures 8.8 and 8.9.

A changeover is performed by calling a member function declared as

```
virtual void CoilBank::ChangeOver (int Nc_deact = 0,
                                   int Nr_rem = 0);
```

In the following it must be borne in mind at all times that in the staging strategy presented here, circuits can only be deactivated from the frontmost elementary coil with respect to the air stream. Thus, in the configuration shown in figure 8.4, circuits can only be deactivated in the 2-row segment of the structure, represented by the two 2-row elementary coils in the diagram. In the computational model of figure 8.8, these correspond to coils C_1 and C_1'. With this restriction, the circuiting may be specified using the parameters in the argument list to the function:

`Nc_deact` The number of circuits to be deactivated (N_d).
`Nr_rem` The number of active rows remaining (N_{rm}) in that portion of the coil in which circuits have been deactivated.

Thus, the configuration shown in figure 8.4 can be specified by setting $N_d = 4$ and $N_{rm} = 0$. The meaning of this second argument may be clarified by considering a variation on this circuiting arrangement. Suppose that we had specified $N_d = 4$ as before, but with $N_{rm} = 1$ in this instance. In other words, we are specifying that the four circuits to be deactivated be contained within *one* ($N_r - N_{rm}$) row of the 2-row elementary coil. In the notation of figure 8.8, this latter can be represented by the elementary coils:

C_1 2 rows, 4 circuits.
C_1' 1 row, 4 circuits; now active.

C_2 1 row, 2 circuits.
C_2' 1 row, 4 circuits.

All circuits are of course of 4 passes, as before.

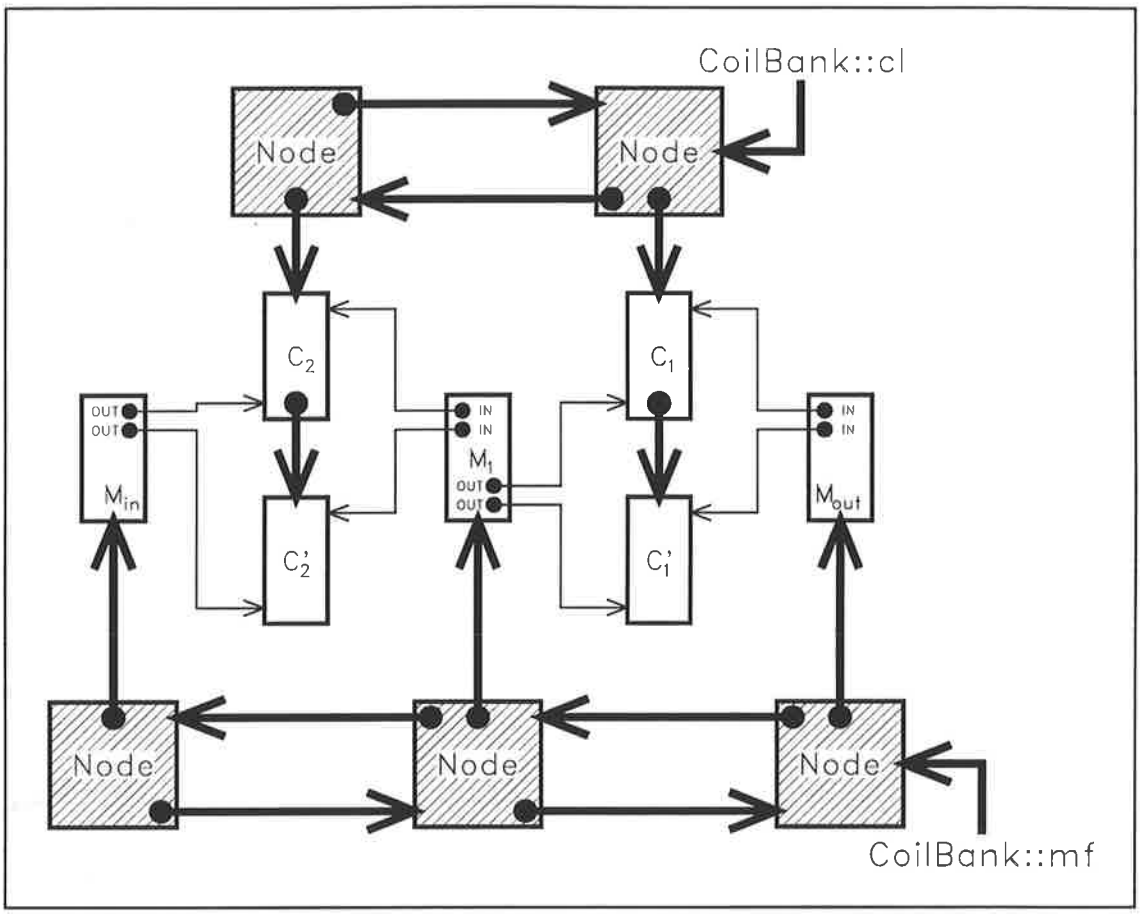


Figure 8.10. *Manifold* data structures used to model the chilled water circuiting configuration shown in figure 8.2.

The above arguments may not be specified arbitrarily. Neither are they independent. The following rules govern the combinations which are admissible:

1. Both arguments must be non-negative. Setting $N_d = 0$ (default value; N_{rm} arbitrary) specifies that all circuits are to be active.

2. The number of rows remaining in that portion of the coil in which circuits have been deactivated must be less than the number of rows in the coil:

$$N_{rm} < N_r \quad . \tag{8.3.1}$$

3. Let N_e be the total number of circuits, active and inactive, in that portion of the coil in which circuits have been deactivated. Suppose for instance that we have a 2-row coil in which two circuits contained within one row have been deactivated. Then, $N_e = 4$, comprising the two deactivated circuits, together with the two matching

circuits in the adjacent row which remain active. This parameter must satisfy the condition

$$N_e \leq N_{ct} \tag{8.3.2}$$

where, if $N_e = N_{ct}$, one or more entire rows will have been deactivated. For a given coil N_e can be calculated from

$$N_1 \equiv N_r - N_{rm} \tag{8.3.3a}$$

$$N_2 \equiv N_{rm} \tag{8.3.3b}$$

$$N_e = N_d\left(1 + \frac{N_2}{N_1}\right) \tag{8.3.3c}$$

where *integer* division is implied in this expression. In other words, the result of the division is truncated to the largest integer value which is less than or equal to the result.

4. The circuits deactivated must be equal in height to an integral number of matching circuits in the adjacent rows which remain active. That is, we require that

$$(N_d \times N_2) \ modulo \ N_1 = 0 \tag{8.3.4}$$

All elementary coils downstream of the first must satisfy a similar condition. Note that this is a restriction imposed by the model, rather than by the possibility of violating physical laws.

Providing N_d and N_{rm} are an admissible pair, the changeover can proceed. One of two situations may arise:

1. $N_e = N_{ct}$. An integral number of rows have been deactivated entirely. In this case coil $\boldsymbol{C_1}$ specifies the half-circuited equivalent to a coil identical to the original coil in all respects except the number of rows, which is reduced to N_{rm}. $\boldsymbol{C_1}'$ points to a *null* coil.

2. $N_e \neq N_{ct}$. In this case we may consider the original elementary coil to comprise two portions, each of which will be represented by a new elementary coil[99]:

   i. Coil $\boldsymbol{C_1}$ contains the same number of rows as the original coil. However, the number of circuits is reduced to $N_{ct} - N_e$.

   ii. Coil $\boldsymbol{C_1}'$ contains N_{rm} rows and $N_e - N_d$ circuits. A special case arises if $N_e = N_d$. In this case the coil is *inactive* but not *null*; the coil passes airflow in proportion to its height, but the air is passed unchanged in respect of its thermal properties.

---

[99] Derived in fact by modifying existing objects.

After setting up, the elementary coils are converted to their half-circuited equivalents.

Staging, as implemented in the current model, splits the air flow into two parallel streams. It is assumed that these will propagate without mixing through the remainder of the coil bank; only then will the separate streams undergo psychrometric mixing. The manner in which this is implemented in the model is shown in figures 8.4 and 8.8. The air leaving coil C_1 subsequently enters coil C_2 by way of state point S_1. In parallel with this, the remaining portion follows the path C_1' - S_1' - C_2'. The process may be continued indefinitely downstream, provided condition (8.3.4) is satisfied by each succeeding elementary coil.

The final step in the changeover procedure is to call member function `CoilBank::ChangeAirFlow (Qa)` to effect a redistribution of air flow through the structure.

8.3.4. Changing the Airflow.

Member function

```
void CoilBank::ChangeAirFlow (double Qa);
```

accepts as its sole argument the air flow rate, expressed as an ASHRAE standard volume flow rate (L/s). Member variables `CoilBank::ma`, `CoilBank::Qa_act`, `CoilBank::Qa_std`, `CoilBank::Va_act` and `CoilBank::Va_std` are calculated accordingly. The actual velocity and flow rate are evaluated using the air state at entry to the coil bank. The air distribution through the coil bank is then altered using the following sequence of operations:

i. In an outer loop, the doubly-linked list `CoilBank::sl` is traversed from head to tail (from the front of the list to the rear).

ii. As each node in `CoilBank::sl` is visited, the singly-linked list to which it points is traversed. Each node on this latter list points to an object of class *StatePoint*, the mass flow rate of air through which may be updated by calling member function `StatePoint::ChangeAirFlow (m)`, where the argument specifies the mass flow rate of air (kg/s) entering the coil bank. The manner in which this function operates has already been described in section 8.2.2. If a state point has no coils feeding into it, the air flow is set equal to the value specified by the argument, otherwise the argument is ignored, and the air flow evaluated by summing the flow through the coils feeding into the state point. In the present context, the mass flow rate through state point S_{in} is set according to the argument, the flow rate for downstream state points being determined by the internal structure of the coil bank.

8.3.5. Calculating Air Pressure Drop.

The air pressure drop for the structure may be found by traversing the doubly-linked list `CoilBank::cl` in the *forward* direction. Each node visited points to a singly-linked list of pointers to objects of class *Coil*, which represent the elementary coils from which the coil bank is constructed. The total air pressure drop may be found by summing the pressure drops through the *first Coil* on each of these latter lists; these coils will never be *null*. The pressure drop for each is found by invoking member function `Coil::AirPressureDrop ()`.

8.3.6. Calculating Cooling Capacity.

The various components of the cooling capacity (`CoilBank::qt`, `CoilBank::qs` and `CoilBank::ql`) may be found simply by visiting each coil addressed by the structure, and accumulating the sums of the various load components.

8.3.7. Initialization.

The solution procedure for problem **FP**, described in chapter 7, requires an initial estimate of the total refrigerating capacity of a given coil component. While the procedure is sufficiently robust to iterate to a solution given any positive value as an initial estimate, computational efficiency will be enhanced if the initial estimate is reasonably close to the actual value. The system initialization procedures described in chapter 6 enable initial estimates of required capacity to be assigned to the various coil banks in a system. This estimated capacity must be aportioned between the various elementary coils from which the coil banks are constructed. In return, the system initialization procedures require an estimate of the air-off condition for the coil bank.

The process of initializing a coil bank is performed by member function

```
void CoilBank::Initialise (double qtd);
```

which takes an estimate of the required total capacity as its sole argument. The initialization is performed in three stages:

1. The *active* surface area for the coil bank is found by summing over all *active* coils in the structure.

2. The total refrigeration capacity for the coil bank (as specified by argument `qtd`) is aportioned among the active components in proportion to their respective surface areas.

3. The coil-off conditions for each component, and hence of the structure as a whole, are found using the following sequence of operations:

    i. The current pointer for list `CoilBank::cl` is set to point to the first node on the list; that for `CoilBank::sl` to point to the second node on the list.

Thus, for the structure shown in figure 8.8, the current pointer for `Coilbank::cl` will address the list containing pointers to coils C_1 and C_1'; that for `Coilbank::sl` will address the list containing pointers to state points S_1 and S_1'. The psychrometric condition at state point S_{in} (`Coilbank::S_in.a`), which specifies the coil-on conditions for coils C_1 and C_1', will have been set beforehand.

ii. The singly-linked list addressed by the current pointer for list `Coilbank::cl` is traversed, and the air-off condition is estimated for each coil on the list. The leaving enthalpy for each may be found as

$$h_{a2} = h_{a1} - \frac{q_t}{\dot{m}_a} \qquad \textbf{(8.3.5)}$$

whence the leaving air dry-bulb temperature may be found using the procedure described in section 7.3.4.

iii. The singly-linked list addressed by the current pointer for list `CoilBank::sl` is traversed, and member function `StatePoint::Update ()` invoked for each element addressed by the list. This establishes the coil-on conditions for the following list of coils.

iv. Provided the end of the list has not yet been reached, the current pointer for lists `CoilBank::sl` and `CoilBank::cl` are moved to the next node on their respective lists, and steps (ii) to (iv) are repeated. On the last loop the coil-off condition for the coil bank (`CoilBank::S_out.a`) will be evaluated.

8.4. Additional Operations for Chilled Water Coils.

The operations described in the preceding section are essentially independent of the coolant-side working fluid. To be fully functional, a second set of operations must be prescribed which account for the coolant-side performance. These must be provided by a derived class. Note that certain of the member functions of class *CoilBank*, such as `CoilBank::Initialise (qtd)` and `Coilbank::ChangeOver (Nc_deact, Nc_rem)`, while fully defined, have been declared **virtual**. This allows them to be redeclared and redefined by the derived class. Others, such as `CoilBank::Solve ()` are *pure* **virtual** functions, and must be provided by the derived class.

The operations described below are member functions of class *CW_CoilBank*, and are suitable for coil banks using chilled water as a coolant.

8.4.1. Solving for Heat Transfer Performance.

With the coil-on condition and entering water temperature, together with the flow rates of the working fluids specified, and the coil bank initialized, we can proceed to solve for its

thermal performance; to solve problem **FP** for the composite structure in other words. For the circuiting situation shown in figures 8.8 and 8.9 the procedure is straightforward:

i. Set the current pointer for the doubly-linked list `CoilBank::sl` to address the first node on the list.

ii. Traverse the singly-linked list addressed by the current node of list `CoilBank::sl`. For each element visited, invoke member function `StatePoint::Solve ()`. This function is described in detail in section 8.2.4. In brief, it causes the internal air state for the state point to be updated by psychrometric mixing of the air streams leaving any coils which feed *into* the state point. This effectively updates the coil-on condition for all coils fed *from* the state point. Member function `Coil::Solve ()` is then called for each coil fed *from* the state point.

iii. If the current pointer is not yet at the end of list `StatePoint::sl`, it is moved to the next node on the list, and steps (ii) and (iii) repeated.

The reader can easily verify that the above procedure will indeed result in a solution to the stated problem. Consider now the circuiting configuration of figure 8.10. The presence of the *intermediate* manifold ($\boldsymbol{M_1}$) introduces a major complication in that the temperature of the chilled water, which is the entering water temperature for coils $\boldsymbol{C_1}$ and $\boldsymbol{C_1}'$ is unknown, and awaits a solution for coils $\boldsymbol{C_2}$ and $\boldsymbol{C_2}'$. However, the air-on condition for these coils is initially unknown. The problem can be recast as a minimization problem, for which efficient solution procedures are available.

Consider a generalization of the configuration shown in figure 8.10, in which we now have n *intermediate* manifolds labelled in sequence [1,...,n] from the tail end of list `CoilBank::mf` (we are concerned with the counterflow situation only). Let $\tilde{t} = [\tilde{t}_1,...,\tilde{t}_n]$ be a vector of estimates for the chilled water temperatures at the manifolds, and follow the sequence of steps (i) to (iii) above. This procedure results in an estimate for the water temperature rise through each coil. Now the temperature at the inlet manifold ($\boldsymbol{M_{in}}$) is known. We may therefore traverse list `CoilBank::mf` in the *reverse* direction to produce an updated estimate of the temperatures at the *intermediate* manifolds, $t' = [t_1',...,t_n']$. Clearly the problem is solved if we can find some $\tilde{t}$ for which

$$f_1(\tilde{t}) \equiv \tilde{t} - t' = 0 \qquad \textbf{(8.4.1)}$$

for each and every intermediate manifold. Equivalently, we seek to minimize the objective function

$$f_2(\tilde{t}) \equiv \sum_{i=1}^{n} (\tilde{t}_i - t_i')^2 \quad . \qquad \textbf{(8.4.2)}$$

As in previous cases, the policy we adopt is one of solving equation (8.4.1) using the root-finding routine zero for the special, but frequently encountered case where $n = 1$. For $n > 1$, the objective function f_2 is minimized using a standard minimization routine (smsno).

The problem of providing initial estimates for $\tilde{t}$ will be dealt with in section 8.4.4. Suffice it to say here that we can provide an initial estimate for the chilled water temperature distribution which will be reasonably close to the actual distribution. It is therefore desirable that a constraint be placed on the search strategy such that the value of $\tilde{t}_i$ for any $1 \leq i \leq n$ cannot be altered by more than an arbitrary amount of (say) 1°C between successive evaluations of f_2. The recommended minimization routine provides a mechanism which makes this possible (Gay, 1983).

8.4.2. Changing the Water Flow.

A member function

```
void CW_CoilBank::ChangeWaterFlow ();
```

is provided to perform the important function of setting the chilled water flow rate through the coil bank. This takes as its argument the desired volume flow rate (Q_w), for which the corresponding mass flow rate ($\dot{m}_w$) is calculated. The doubly-linked list `CoilBank::mf` is then traversed in reverse order (from $\boldsymbol{M_{in}}$ to $\boldsymbol{M_{out}}$). For each manifold visited, the chilled water volume flow rate is set by invoking member function `CW_Manifold::UpdateMassFlow (Qw)` (section 8.2.3).

8.4.3. Calculating Water Pressure Drop.

Water pressure drop through the coil bank is calculated and returned by a member function `CW_CoilBank::WaterPressureDrop ()`. In this routine list `CoilBank::mf` is traversed in *reverse* order, and member function `CW_Manifold::UpdatePressure ()` (section 8.2.3) invoked for each manifold visited. The water pressure drop through the coil bank is then simply estimated as

$$\Delta p_w = p_{in} - p_{out} \quad . \tag{8.4.3}$$

8.4.4. Initialization.

The initialization routine provided by the base class performs initialization functions relating to the refrigeration capacity and air side of the coil bank only. In order to initiate the solution procedure described in section 8.4.1 above it is necessary to provide an estimate of the chilled water at each manifold. Class *CW_CoilBank* provides a member function

```
CW_CoilBank::Initialise (double qtd);
```

which overrides the **virtual** function of the same name provided by the base class. This new function explicitly calls the base class member function `CoilBank::Initialise (qtd)` to set up initial estimates for the refrigeration capacity and air-off condition for the various components of the coil bank, according to the estimated refrigeration capacity for the coil bank. The extra function performed within the derived class is that of making an initial estimate for the chilled water temperature distribution. It

is assumed that the distribution of chilled water flow through the structure will have been initialized by invoking member function `CW_CoilBank::ChangeWaterFlow (Qw)`, where the argument specifies in this instance an estimated water flow (section 8.4.2; see also section 6.3.6). We note that, given the capacity and mass flow rate of the working fluids for the coil, the corresponding water temperature rise can be found from

$$\Delta t_w = \frac{q_t}{C_{p,w}\dot{m}_w} \quad . \tag{8.4.4}$$

The steps required to initialize the water temperature for a coil bank are then as shown in figure 8.11.

```
{
        t ⇐ 0;
        Reset current pointer for list mf;
        for each preceding manifold do                    (Traverse mf in reverse direction)
        {
                if inlet manifold then
                        t ⇐ Manifold temperature;
                else
                        Manifold temperature ⇐ t;
                                          (Call Manifold::SetTemperature (t))
                N_c ⇐ 0;
                q_t ⇐ 0;
                for each coil fed from manifold do
                {
                        Coil inlet water temperature ⇐ t;
                        N_c ⇐ N_c + number of circuits fed;
                        q_t ⇐ q_t + total refrigeration capacity;
                }
                if circuits fed then                                        (N_c ≠ 0)
                        Δt ⇐ q_t/(ṁ_w C_p);          (WTR from equation (8.4.4))
                else                                                        (N_c = 0)
                        Δt ⇐ 0;
                t ⇐ t + Δt;
                for each coil fed from manifold do
                        Coil water temperature rise ⇐ Δt;
        }
}
```

Figure 8.11. Logic to initialize the water side characteristics of a coil bank.

8.4.5. Estimating Coil Capacity.

Member function

```
void CW_CoilBank::EstimateCapacity (double epsilon = 0.3);
```

is a utility function used to provide a preliminary estimate of the capacity of a coil bank for use as a starting value in iterative schemes. When a coil is considered within the context of the air conditioning system of which it is a component, it is always possible to make an initial estimate of the required coil capacity (section 6.3.6). From time to time it is necessary to estimate the performance of a coil solely on the basis of the flow rates and entering conditions of the working fluids. The function has been implemented to address the initiialization requirements of this situation. The estimate returned by the function is approximate in the extreme, and should be used *for no other purpose than that for which it is intended.* Thermal capacity is estimated using the concept of heat exchanger effectiveness as

$$\varepsilon = \frac{q_{actual}}{q_{max}} \qquad \textbf{(8.4.5)}$$

where,

q_{actual} is the actual rate of heat transfer, and
$q_{\max}$ is the maximum possible rate of heat transfer given the same working fluids, the same inlet temperatures, and the same flow rates.

In general, ϵ is a function of the heat exchanger geometry and operating conditions (Stoecker, 1989; Kays and Crawford, 1993), and must take a value in the range $0 < \epsilon < 1$. In the present case the user *prescribes* a value of ϵ as an argument to the function. This takes a default value of $\epsilon = 0{\cdot}3$.

An estimate of heat exchanger thermal capacity may now be made. The water-side capacity rate,

$$C_w = \dot{m}_w C_{p,w}(t_{a1} - t_{w1}) \qquad \textbf{(8.4.6)}$$

while the air-side capacity rate,

$$C_a = \dot{m}_a(h_{a1} - h_{aw}) \qquad \textbf{(8.4.7)}$$

where h_{aw} is the enthalpy appropriate to an air state defined by

$$t_{aw} = t_{w1} \qquad \textbf{(8.4.8a)}$$

$$t''_{aw} = \min(t''_{a1}, t_{w1}) \qquad \textbf{(8.4.8b)}$$

Clearly, if the second law of thermodynamics is not to be violated,

$$q_{max} = \min(C_w, C_a) \qquad \textbf{(8.4.9)}$$

and the actual heat exchanger capacity may be estimated from the definition of heat exchanger effectiveness (8.4.5).

8.5 Summary.

The derivation of rating curves for a series of coils is a time-consuming and costly procedure. It is therefore customary to restrict testing to a matrix of 'standard' coil configurations. In the experimental programme undertaken at the University of Adelaide, rating curves have been derived for half-circuited coils having a fin density of six fins per inch, and one, two and four rows deep. In practice it is frequently desirable to use coil configurations which do not correspond to those contained within the 'standard' matrix, or for control purposes, to consider a coil as being composed of several sections. In the present chapter a methodology for synthesizing such coils using a set of standard components, within the context of a computer model, has been described. Algorithms for predicting the peformance of such composite coils have also been presented.

Chapter 9. Pipe and Duct Networks.

The air involved in an air conditioning cycle is subject to parasitic heating and cooling caused by heat gains or losses through the walls of the air conditioning ductwork, and by the conversion of fan power into heat. Both must be estimated to obtain a closure to the problems posed in Chapter 6. The estimation procedure involves the identification and analysis of the relevant paths through the duct network. In the case of transmission through the duct walls, the effect on the air supplied to the zones, and on the air returned from the zones to the air handling unit will be the cumulative effect of the gains or losses through the walls of the various sections of the appropriate paths. Determination of fan power dissipation requires a model of the fan characteristic. This is the subject of Chapter 10. To determine the operating point on the fan characteristic it is necessary to find the path offering the highest resistance to the airflow. It is desirable that these two problems be handled in a unified manner[100].

In the present chapter a set of abstractions and algorithms are developed for analyzing duct flow problems of the type described above. From a computational point of view it is convenient to regard air conditioning duct systems as a specialized class derived from a more general base class, which may also serve as a base for other related classes, including, in particular, pipe networks. Thus, we define a base class *Network*, which has associated with it class *Fitting* and class *Section*, which provide generic base functions for modelling the fittings and sections which comprise a complete network. Between them, these classes contain a complete set of member functions and variables to enable the pressure drops and heat gains and losses at the various points in a network to be calculated, subject to the availability of information relating to the geometry of the fittings and sections involved and to the properties of the working fluid, which must be supplied by the appropriate derived classes. A class *Duct*, derived from class *Network*, is described in detail, as are a set of derived classes simulating duct sections and fittings. Pipe networks, although important in air conditioning applications, are not described in the present work, which focusses on the air side of the air conditioning cycle. These may be represented by a class *Pipe*, which can be derived in a relatively straightforward manner from base class *Network*.

The algorithms presented in the following relate specifically to the class of networks known as *tree* networks. This restriction is appropriate in that the vast majority of air conditioning duct systems may be adequately described as tree networks. Such networks also occur extensively elsewhere in engineering practice, as noted by Tsal and Adler (1987). While the present work draws heavily upon the methods of Tsal and his coworkers, the problem tackled herein differs fundamentally from those considered in their development of the T-method, namely the specification of optimal duct networks (Tsal et al., 1988a,b), and the 'retrospective' problem of finding the air distribution in a duct system

---

[100] Implementation of the techniques described in this and the next chapter is currently progress, and the facilities dscribed have not yet been fully integrated into the ZEBRA package.

with specified damper settings and fan characteristic (Tsal et al., 1990)[101]. In the present case the problem is 'prospective' in that the air distribution is *specified* by the problem and we seek to find the resistances (imposed for instance by damper settings) which will produce the desired air distribution for a given air flow.

It is recognized that the tree network configuration will not be sufficiently general to describe the chilled water distribution system in most instances; neither will it be sufficiently general to model certain duct configurations, such as the case of a ring duct served by two or more air handling units. In the most general case a pipe or duct network may be represented computationally by a *directed acyclic graph* (see for instance Tremblay and Sorensen, 1984; Ahuja et al., 1993). The data structures described in the following provide for extension to this more general case with minimal modification.

In the remainder of this chapter the term network will be understood to refer to a *tree* network.

9.1. Topology.

In general, as noted above, a flow network may be represented computationally by a *directed acyclic graph*, each *node* of which models one *section* of the network. A section may be defined as a length of conduit of uniform cross-section and uniform mass flow rate, the extent of which is delimited by one of the following:

i. An inlet.
ii. An outlet or terminal.
iii. A change of cross-section, in which case the change of cross-section will be regarded as coinciding with the geometrical centre of the contraction or diffuser whereby the change of cross-section is effected.
iv. A junction, in which case the section is regarded as terminating at the geometrical centre of the junction.

v. An elbow, in which case the section is regarded as terminating at the geometric centre of the elbow.

The flow pressure loss within a section will be determined by the flow velocity and fluid properties, the cross-sectional shape and dimensions of the conduit, the roughness characteristics of the conduit wall, and the presence of *fittings* associated with the section. These latter include contractions, diffusers, bends, dampers, valves and so on. Fittings which delimit a section (contractions, diffusers and junctions, for instance) will be associated with more than one section.

---

[101] The data structures described in the following are eminently suited to form the basis of an implementation of the T-method as originally formulated, in both its optimization and simulation forms.

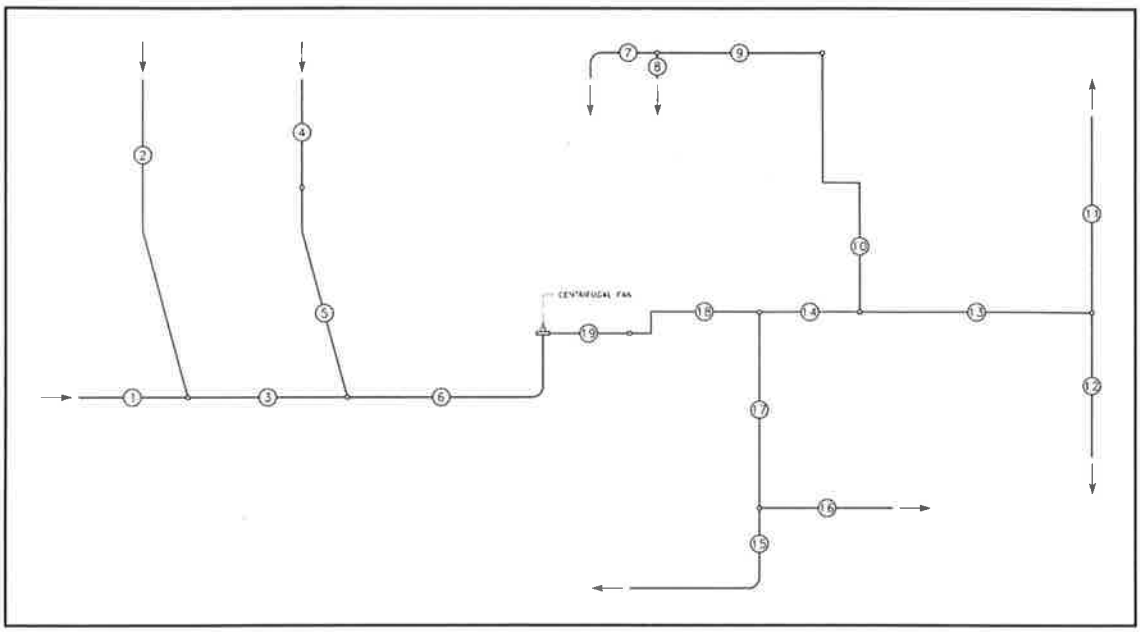


Figure 9.1. Supply and return air duct layouts for the "ASHRAE example" (Tsal et al., 1990).

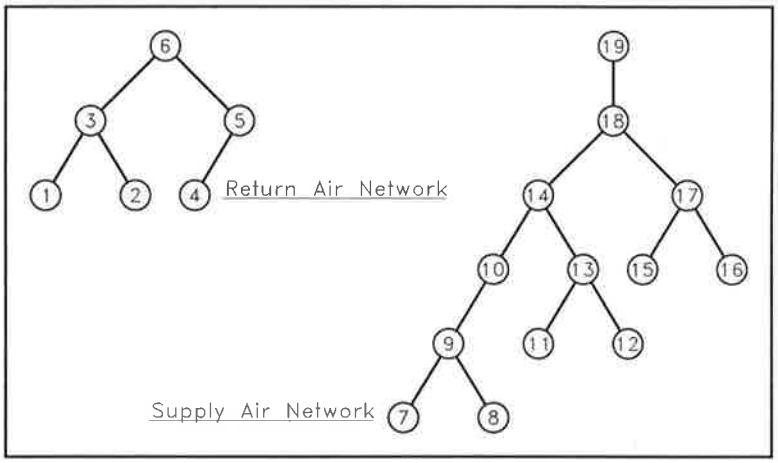


Figure 9.2. Tree representation of the duct layouts for the "ASHRAE example".

In most cases air conditioning duct systems are subject to leakage (ASHRAE, 1989; chapter 32). Such systems may be modelled as *lossy* networks if the static pressure within the duct exceeds ambient atmospheric pressure, or *gainy* networks if duct static pressure is less than ambient atmospheric pressure. Such networks are considered by Ahuja et al. (1993). The effect of leakage is to impose an additional energy burden on the system which, in extreme cases, may become substantial. Leakage may be minimized, if not economically eliminated, by the use of quality constructional techniques. In the present work the effects of leakage are ignored, while noting that there are computational techniques available to model the phenomenon.

A tree may be defined as a graph characterized by the absence of circuits[102]. If we consider the entire duct system for a particular project, this criterion will not be satisfied except in the case of certain systems employing 100% outdoor air. In a very large number of cases

---

[102] The terminology and notation of graph theory are described in a number of works; see for instance Tremblay and Sorensen (1984) and Ahuja et al. (1993). A number of terms of relevance to the current work will be defined below.

however, the supply and return air duct systems, treated separately, may be adequately modelled by tree networks. Consider the supply and return air duct system shown in figure 9.1, based on Tsal et al. (1990), where it is referred to as the "ASHRAE example". Computationally, this duct system may be represented by the two tree networks shown in figure 9.2. Referring to this figure, we may define the following terminology:

a. Node 6 is the *root* node for the return air network; node 19 is the root node for the supply air network.

b. Nodes 1, 2 and 4 in the return air network, and nodes 7, 8, 11, 12, 15 and 16 in the supply air network are *terminal* nodes.

c. The *level* of a node is **n+1** if **n** is the length of the path connecting the node to the root. Thus, in the return air network node 6 is at level 1, nodes 3 and 5 are at level 2, and nodes 1, 2 and 4 are at level 3. Node **a** is said to be at a *higher* level than node **b** if level (**a**) < level (**b**).

d. It is useful to consider a tree as a *directed*[103] graph in which each *arc* connecting an adjacent pair of nodes is orientated in the direction from the higher level to the lower level. A *path* exists from node **a** to node **b** if node **b** may be reached from node **a** by following a sequence of arcs in the *forward* direction. The absence of circuits guarantees that if a path exists between node **a** and node **b** it will be unique. The path from node 19 to node 7 is thus 19-18-14-10-9-7. The *length* of a path is the number of arcs in the path; this path is of length 5. Two nodes are *connected* if a path exists from one of the pair to the other. Nodes 14 and 7 are connected; nodes 11 and 7 are not.

e. Node **b** is a *child* of node **a** if they are connected, and level (**b**) = level (**a**) + 1. Conversely, node **a** is the *parent* of node **b**. Except in the case of the root node, which has no parent, each node will have one and only one parent node. All nodes except the terminal nodes will have at least one child node. The number of child nodes which a particular node has is referred to as the *degree* of the node. If for a particular tree the degree of each node is less than or equal to some number **n**, that tree will be referred to as an *n-ary* tree[104]. The examples shown in figure 9.2 are *binary* trees. The common practice of locating take-offs opposite each other on a main duct means that for the most part **n** = 3 for the cases with which we will be concerned; these are *ternary* trees. The data structures to be developed later in this chapter will be capable of handling the case where **n** is arbitrary. The nodes at a higher level, to which a node is connected, are referred to as its *ancestors*; those at a lower level, to which it is connected, are referred to as its *descendants*. The root node is a common ancestor to all other nodes in the tree.

---

[103] In the present context, this has nothing to do with the direction of the fluid flow.

[104] It is also an example of a *multiway* tree.

f. A *connected subgraph* of a tree is called a *subtree*. Nodes 7, 8, 9, 10, 14, 11, 12 and 13 define a subtree rooted in node 14.

g. A tree or subtree may be *traversed* by *visiting* all the nodes in a defined sequence. In a *preorder* traversal, each node is visited before its child nodes, which are then visited in sequence from the left-hand side of the tree. Thus, the preorder traversal of the return air network visits the nodes in the order 6, 3, 1, 2, 5, 4. In a *postorder* traversal, each node is visited after its child nodes have been visited in sequence. The postorder traversal of the return air network visits the nodes in the order 1, 2, 3, 4, 5, 6[105].

h. Networks in which the flow of fluid is directed away from the root, as in supply air duct systems, are said to experience *diverging* flow. Where the flow is towards the root, the flow is said to be *converging*.

9.2. Definition of Network Analysis Problems.

In order to determine the parasitic heat gains and losses caused by fan energy dissipation and heat transfer through the duct walls, a series of fundamental sub-problems must be solved. They are:

NF	Given a tree network with a mass flow of fluid assigned to each terminal node, find the distribution of fluid flow through each node of the network.

NT1	Given a tree network subject to diverging flow for which the temperature of the fluid entering the root node is known, find the temperature of the fluid entering, and the temperature gain or loss through each node of the network.

NT2	Given a tree network subject to converging flow for which the temperature of the fluid entering each of the terminal nodes is known, find the temperature of the fluid entering, and the temperature gain or loss through each node of the network.

NP	Given a tree network or a subtree thereof for which the flow distribution is known, determine the pressure loss through each path of the network.

---

[105] For binary trees only, *inorder* traversal is defined in which the *left* child node is visited, then the node itself, then the *right* child node.

Data structures and algorithms to solve these problems will be developed first for general networks, following which their specialization to a duct system carrying moist air will be considered. Finally, the use of these basic algorithms and data structures within a software sytem designed to solve operational problems will be described.

9.3. Modelling Generic Flow Networks.

Class *Network* is an abstract base class. Together with its associated base classes, *Section*, *Fitting* and *nNode*, it provides a basic set of operations for modelling generic tree networks. The class is abstract in the sense that a number of the functions declared within this class and its associated classes are pure **virtual** functions since they seek to implement operations which are dependent either on properties of the working fluid, or on factors specific to a particular type of physical network. These functions must be defined within a derived class. The following section describes a set of classes which, when used in conjunction with those described in the present section, from which they are derived, provide a complete set of facilities for simulating duct systems carrying moist air.

The purpose of the present section is to describe the facilities provided by this fundamental set of base classes, together with the underlying physical principles upon which these classes and their derived classes are based.

9.3.1. Class *Section*.

Class *Section* seeks to provide the facilities for describing and modelling a *section* of conduit, as defined in section 9.1. The two fundamental properties of the flow through the section which are of interest are the pressure losses incurred in the section, and the heat transfer through the walls of the conduit. Ductwork for air conditioning applications is available in a number of cross-sectional geometries, of which circular and rectangular ducts are the most important. It is desirable that the facilities provided by the base class be independent of the geometry used. This can be achieved by expressing the geometry in terms of two equivalent diameters which will be defined in the following.

9.3.1.1. Pressure Losses.

The pressure loss in a section of conduit with associated fittings may be found using the Darcy-Weisbach equation:

$$\Delta P = \left(\frac{fL}{D_f} + \sum C \right) \frac{V^2 \rho}{2 g_c} \qquad \textbf{(9.3.1)}$$

in which the first term in parentheses represents a loss coefficient for frictional losses incurred at the walls of the conduit, while the second term is the sum of the local loss coefficients representing the frictional and dynamic losses occurring in the various fittings associated with the section. Within the computational model the fittings are represented by objects of class *Fitting* or one of its subclasses. Pointers to objects of this type associated with a particular object of class *Section* are stored on a linked list, which is a

member variable of class *Section*. The sum of the local loss coefficients is evaluated by invoking a member declared as

```
virtual double Section::SigmaC ();
```

The manner in which this function operates is straightforward. Class *Fitting* and its subclasses have a member function

```
virtual double Fitting::C ();
```

which computes and returns the local friction loss coefficient. Function `Section::SigmaC` simply performs a traversal of the list on which the pointers to the fittings are stored, invokes the appropriate function to evaluate C for the fitting, and accumulates the sum.

The geometrical parameters required to evaluate the wall friction losses for the conduit are the length L, and D_f, which is the diameter of a circular conduit having the same friction loss characteristics as the conduit under consideration. By substituting an appropriate expression for D_f, equation (9.3.1) may thus be used for conduits of arbitrary cross-sectional shape. In the present work, we are only concerned with conduits of circular and rectangular cross-section. For circular conduits, the equivalent-by-friction diameter is identically equal to the actual diameter. For non-circular conduits it is customary to use the *hydraulic diameter*, defined as

$$D_h \equiv 4\frac{A}{P} = 2\frac{hw}{h + w} \qquad \textbf{(9.3.2)}$$

where A is the cross-sectional area of the conduit, and P is the perimeter, both dimensions referring to the interior surface. For a rectangular conduit this reduces to the expression on the right, where h is the height of the conduit, and w is its width. The suitability of the hydraulic diameter as a length scale for estimating the friction factor for rectangular channels has however been questioned (Jones, 1976; ASHRAE, 1989; chapter 32). On the basis of an exhaustive examination of the available experimental data for both laminar and turbulent channel flow, Jones (1976) proposed a correction to the hydraulic diameter which was shown to correlate the available data well. This has already been presented in the context of airflow through coils, and will be repeated here in the notation appropriate to the current application. Jones proposed multiplying the hydraulic diameter by a geometry factor so that

$$D_f = \phi^*\left(\frac{w}{h}\right) D_h \qquad \textbf{(9.3.3)}$$

where it is assumed that $w \geq h$. If the converse is the case, w and h should be interchanged in this and the following expression. Jones suggested the following formulation for the geometry factor:

$$\phi^*\left(\frac{w}{h}\right) = \frac{2}{3} + \frac{11}{24}\frac{h}{w}\left(2 - \frac{h}{w}\right) \quad . \qquad \textbf{(9.3.4)}$$

In the applications with which we will be concerned the fluid flow will in general be turbulent. Turbulent flow in rough-walled channels falls broadly into two flow regimes. At lower Reynolds numbers the friction factor f is a function of both Reynolds number and the relative roughness of the channel walls (ϵ/D_f). With increasing Reynolds numbers friction factor becomes a function of relative roughness alone. The appropriate Reynolds number to use in calculating friction factor is based on the equivalent-by-friction diameter, and the mean velocity, thus

$$Re \equiv \frac{VD_f\rho}{\mu} \quad . \tag{9.3.5}$$

The mean velocity for both circular and non-circular channels can be evaluated in a uniform manner by defining an equivalent-by-velocity diameter which, for a non-circular channel, would be the diameter of a circular channel having the same cross-sectional area. For a rectangular channel

$$D_v = 2\sqrt{\frac{hw}{\pi}} \quad . \tag{9.3.6}$$

The friction factor for flow in a rough-walled circular channel may be evaluated using a relationship due to Colebrook (1939), which was used in constructing the well known Moody chart. This is

$$\frac{1}{\sqrt{f}} = -2\log_{10}\left[\frac{\epsilon}{3{\cdot}7D_f} + \frac{2{\cdot}51}{Re\sqrt{f}}\right] \quad . \tag{9.3.7}$$

The dependence of the friction factor on Reynolds number and relative roughness, as predicted using equation (9.3.7), is shown in figure 9.3.

Equation (9.3.7) must be solved iteratively. Tsal (1989) has proposed an explicit relationship for the friction factor based on a formulation originally developed by Altshul. The Altshul-Tsal formula defines a factor

$$f' = 0{\cdot}11\left[\frac{\epsilon}{D_f} + \frac{68}{Re}\right]^{0{\cdot}25} \quad . \tag{9.3.8}$$

If $f' \geq 0{\cdot}018$,

$$f = f' \quad . \tag{9.3.8a}$$

If $f' < 0{\cdot}018$,

$$f = 0{\cdot}85f' + 0{\cdot}0028 \quad . \tag{9.3.8b}$$

Friction factor as a function of Reynolds number and relative roughness, as predicted using equation (9.3.8), is shown in figure 9.4. Routines using member functions of class *Section* may elect to compute friction factor using either the Altshul-Tsal or the Colebrook equation. A comparison of figures 9.3 and 9.4 shows that the friction factor evaluated

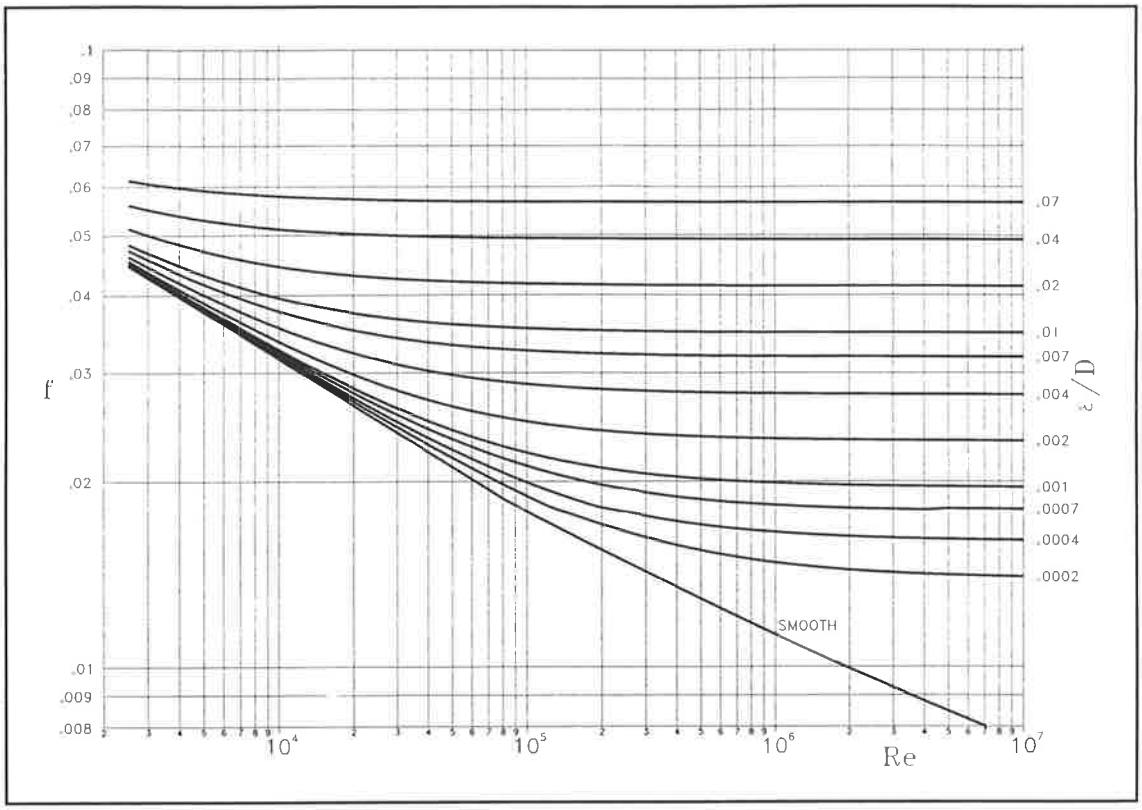


Figure 9.4. Friction factor as a function of Reynolds number and relative roughness for a rough-walled circular channel, as evaluated using the Altshul-Tsal equation.

using the Altshul-Tsal formula will be in close agreement with that predicted using the Colebrook formula provided the relative roughness does not exceed approximately 0·004, and is at its best within the range $0{\cdot}0002 < \epsilon/D < 0{\cdot}001$. With increasing relative roughness, the Altshul-Tsal formula systematically underestimates friction factor. This relationship is however adequate for use over the range of relative roughness levels commonly encountered in air conditioning duct flows, and is used by default in class *Duct*. In a future implementation of class *Pipe*, it may be more appropriate to specify the Colebrook relationship as the default formula for predicting friction factor[106].

Class *Section* provides a member function

```
double Section::PressureDrop ();
```

which calculates and returns the pressure drop for flow through a section of conduit. This function in turn calls member functions to evaluate the Reynolds number of the flow, and the sum of the local loss coefficients for the section. The former is a *pure* **virtual** function, and must be defined in a derived class.

---

[106] Ward-Smith (1980) provides an alternative direct relationship for f which is claimed to agree with the Colebrook equation to within about 1% within the range $4 \times 10^3 < Re < 10^7$ and $0 < \epsilon/D < 0{\cdot}1$. Attempts to implement this relationship have however failed to produce agreement of the order claimed, and since the source of the equation is not cited, it is not possible to verify the form given by Ward-Smith.

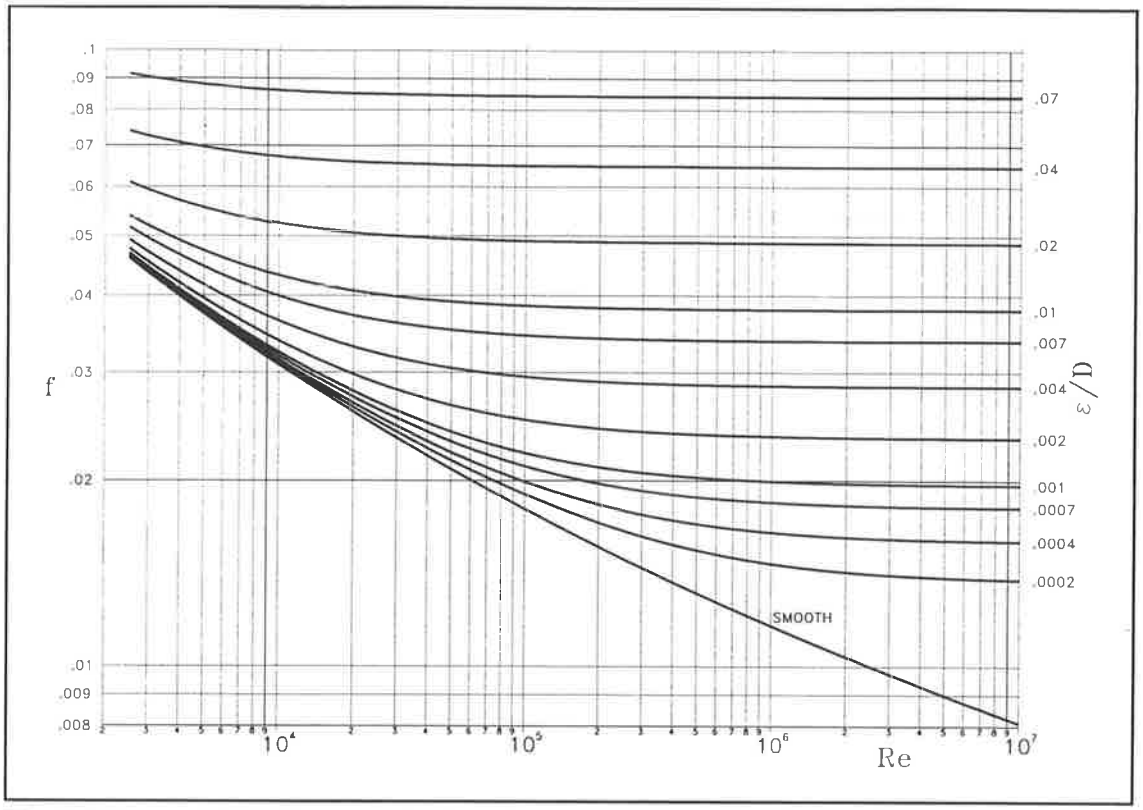


Figure 9.3. Friction factor as a function of Reynolds number and relative roughness for a rough-walled circular channel, as evaluated using the Colebrook equation.

The duct section conductance

$$K_s \equiv \frac{Q_{a,act}}{\sqrt{\Delta P}} \tag{9.3.9}$$

is also calculated. This parameter is of importance in the T method.

9.3.1.2. Heat Transfer.

If fluid enters a conduit at some temperature t_e, and leaves with temperature t, and the conduit is exposed to an external temperature t_a, then the rate at which the fluid exchanges heat with its surroundings may be found from (ASHRAE, 1989; chapter 32)

$$Q_l = UPL\left[\left(\frac{t_e + t_l}{2}\right) - t_a\right] \tag{9.3.10}$$

where Q_l is positive if the fluid loses heat to its surroundings. U is the overall heat transfer coefficient, P is the external perimeter of the conduit, and L is the length of the section. Now, we also have

$$Q_l = \dot{m} C_p (t_e - t_l) \quad . \tag{9.3.11}$$

Thus, equating (9.3.10) and (9.3.11), and defining

$$y \equiv \rho \frac{C_p V A_i}{UPL} = \dot{m} \frac{C_p}{UPL} \qquad \textbf{(9.3.12)}$$

where V is the velocity of the fluid in the conduit, and A_i is the flow cross-sectional area, expressions may be derived relating the temperature of the fluid leaving the section to the temperature of the fluid entering, and vice versa:

$$t_e = \frac{t_l(y + 1) - 2t_a}{y - 1} \qquad \textbf{(9.3.13a)}$$

$$t_l = \frac{t_e(y - 1) - 2t_a}{y + 1} \; . \qquad \textbf{(9.3.13b)}$$

Thus, given a fluid flowing through a section of conduit of specified cross-sectional geometry, the heat transfer problem may be solved provided an estimate for U can be made. This issue will be taken up further in section 9.4 in the context of the flow of moist air through air conditioning ducts. Class Section provides a pure virtual member function

```
virtual double Section::TemperatureRise () = 0;
```

to calculate the change in temperature of a fluid flowing through a section of duct. This function must be defined in a derived class. The derived class must also define a function to calculate the overall heat transfer coefficient for the section. This function is declared in the base class as

```
virtual double Section::Uc () = 0;
```

9.3.2. Class *Fitting*.

Class *Fitting* is an abstract class which provides a base from which classes describing a wide range of pipe and duct fittings can be derived. The facilities provided by this class are fairly sparse, and have largely been determined by considerations relating to the implementation of a set of subclasses to model a range of duct fittings for which tabulated friction loss coefficients are provided in ASHRAE (1989; Chapter 32). The fittings in that publication divide broadly into six classes, and each fitting may be further identified by an index within its class; member variables are provided within the base class to store the class and index numbers for a fitting. Fittings may be further divided into two broad categories, depending upon whether the friction loss coefficient may be considered to be determined solely by the geometry of the friction, or whether the loss coefficient is also a function of flow velocity. A member variable within the base class flags whether the loss coefficient is velocity-dependent or not. In addition to the usual default constructor and accessor functions, class *Fitting* provides two virtual member functions which are of further interest:

i. Function

```
virtual double Fitting::C ();
```

computes and returns the local friction loss coefficient for the fitting. If this is simply a function of the geometry of the fitting, a *constant* value may be established by the constructor. Otherwise, C must be calculated as a function of fluid velocity.

ii. Function

```
virtual void Fitting::SCheck () = 0;
```

which is called by the constructor for a subclass, and conducts a series of context-specific checks on the validity of the fitting the constructor specifies.

Further discussion of the features of this class is deferred until section 9.4, where it will be considered as the base of a hierarchy of classes used to model air conditioning duct systems.

9.3.3. Class *nNode*.

Each section of conduit in a network may be modelled in isolation by constructing an appropriate object of class *Section*, or one of its subclasses, together with a set of objects of class *Fitting* to represent the fittings associated with that section. The purpose of class *nNode* is to provide a means of linking the disparate set of sections into a tree structure such as that shown in figure 9.2, and performing a desired set of manipulations on the structure once it is in place. Each object of class *nNode* represents one node within the network tree, and maintains a pointer to the object of class *Section* with which it is associated. Each node is also responsible for maintaining links to its *child* nodes. Pointers to these are entered onto a singly-linked list, permitting an arbitrary number of child nodes to be handled. The following two boolean variables are also members of class *nNode*:

`term` : Flags whether this is a terminal node.
`sub` : If assigned a value of *true*, this variable indicates that the node is to be used as the root of a subtree in further calculations.

The significance of these two variables will be made clear in section 9.4.

Class *nNode* is provided with a set of member functions which perform housekeeping functions. These are:

i. A *constructor*, which is defined as follows:

```
nNode::nNode (Section& section,
              Network& network,
              boolean subtree = false,
              boolean terminal = false);
```

The first two arguments contain a reference to the section to which this node refers, and a reference to the network to which it belongs. The last two arguments assign

values to member variables `sub` and `term` respectively. These latter variables are assigned a value of *false* by default.

ii. A *destructor* which recursively deletes the subtree rooted in the current node, by removing a pointer to each child node from the list on which it is stored, and deleting the child node[107]. Finally, the section associated with this node is deleted.

iii. A function to access the section associated with this node.

iv. Functions to add child nodes to the head or tail of the linked list.

A second set of member functions have been implemented to solve the problems defined in section 9.2. These operate recursively[108] over a subtree rooted in the current node, and implement the following algorithms:

i. Algorithm **NN1** solves problem **NF**. The mass flow of fluid through the subtree rooted in the current node is calculated by performing a preorder traversal of the subtree, and summing the mass flow through the child nodes of each node visited. To initialize this algorithm, a mass flow must be assigned to each terminal node. The algorithm is implemented in a member function

```
double nNode::SumFlow ();
```

which returns the mass flow through the current node. The logic upon which this algorithm is based is shown in figure 9.5.

ii. Algorithm **NN2** alters the *pressure datum* for a subtree rooted in the current node. If P is the *base pressure* for the current node, and ΔP is the pressure drop through the section associated with the node, then the base pressure for each of the child nodes will be $P' = P + \Delta P$[109]. The *pressure datum* for a subtree is the base pressure for the root node for that subtree. This may be arbitrarily specified, but defaults to 0 for the root node for the entire tree. In computations involving the pressure balancing of air conditioning duct systems, a need will arise to alter the pressure datum for certain subtrees. These will be identified by assigning a value

---

[107] It will be recalled that the operation of *deleting* a C++ object causes the destructor for that object to be invoked.

[108] The use of recursion gives rise to a particularly elegant implementation. Note however that non-recursive forms are often preferred for 'industrial strength' implementations (Tanenbaum et al., 1990), the main reason being that recursion can lead to stack overflow if recursive calls are nested too deeply. In the initial release of this software at least, a recursive formulation has been used since it is considered that the potential disadvantages of this approach would be far outweighed by the complexity of a non-recursive approach.

[109] Note that in the convention adopted here, *base* pressure *increases* in the direction of *increasing* pressure drop. The base pressure for a node is therefore the sum of the pressure drops through its ancestor nodes, referenced to an arbitrary value at the root node.

```
Function SumFlow:
{
        if node has child nodes then
        {
                m ⇐0;                                          (Mass flow through node)
                for each child node t do                       (t is pointer to child node)
                        m ⇐m + t->SumFlow ();
                                        (SumFlow invoked recursively for each child node)
        }

        return mass flow through current node;

}
```

Figure 9.5. Logic for algorithm **NN1**.

of *true* to member variable `sub`. The pressure datum for a subtree may be altered by invoking member function

```
void nNode::PressureDatum (double P);
```

for the root node, where P is the desired new datum. This function performs a postorder traversal of the subtree rooted in the current node, setting the base pressure for the current node to P, and then invoking itself recursively to set the base pressures for the child nodes to P'. The distribution of pressure drops through the subtree will have been calculated beforehand.

iii. Algorithm **NN3** solves problem **NP**. The pressure loss distribution through the subtree rooted in the current node is calculated by performing a preorder traversal of the subtree. The base pressures for the nodes visited are simultaneously calculated. Two modes of operation are possible:

Mode A: The pressure loss distribution is evaluated over the entire subtree.

Mode B: If a node for which `sub` is *true* is encountered during the traversal, the pressure loss distribution for the subtree rooted in that node is not calculated, but the pressure datum for the subtree, and hence the base pressures for its member nodes are adjusted.

Algorithm **NN3** is implemented in a member function

```
void nNode::Pressure (double P,
                      boolean sT);
```

where P specifies the pressure datum for the subtree (defaults to 0), and sT specifies the mode of operation (true for mode A). The pressure distribution is calculated by invoking the above function recursively, as shown in the diagram of figure 9.6.

```
Function Pressure.
Arguments:      P :     Base pressure for the node (default 0·0).
                sT :    true : Mode A.
                        false : Mode B.
{

        Base pressure for node ⇐ P;
        Calculate pressure drop through section;                    (Section 9.3.1.1)
        P ⇐ P + presssure drop;
        for each child node t do                                    (t is pointer to child node)
        {
                if not sT and t->sub then
                        t->PressureDatum (P);
                                (If mode B and root of subtree, adjust pressure datum)
                else
                        t->Pressure (P, sT);
                                                (Otherwise, invoke Pressure recursively)
        }

}
```

Figure 9.6. Logic for algorithm **NN3**.

iv. Algorithm **NN4** solves problem **NT1**, and is implemented in a member function

```
void nNode::TemperatureDiverging (double T);
```

where T is the entering fluid temperature to be assigned to the section associated with the node. The temperature distribution through the subtree is calculated using a simple preorder traversal of the subtree, invoking function `TemperatureDiverging` recursively for each node visited. At each invocation of function `TemperatureDiverging` the following steps are performed, where s is the channel section associated with the node:

a. T is assigned to be the entering temperature for s.

b. The temperature rise (Δt) through the duct is calculated using the methods described in section 9.3.1.2, and the temperature of the fluid leaving the duct assigned to T', such that $T' = T + \Delta T$.

c. Member function `TemperatureDiverging` is invoked for each child node with T' as an argument.

v. Algorithm **NN5** solves problem **NT2**, calculating the temperature and mass flow distributions for a network subject to diverging flow, for which both the fluid entering temperatures *and* the mass flow rates at the terminal nodes have been assigned. The algorithm used is essentially an adaptation of algorithm **NN1**, and is implemented as a member function

```
void nNode::TemperatureConverging ();
```

```
Function TemperatureConverging:
{

        if node has child nodes then
        {
                σm ⇐ 0;                                          (Mass flow through node)
                σmt ⇐ 0;                    (Mass flow through node × Entering temperature)
                for each child node t do                          (t is pointer to child node)
                {
                        t->TemperatureConverging ();
                                                  (Call function recursively for child node)
                        Calculate temperature rise for node t;
                        m ⇐ Mass flow through node t;
                        σm ⇐ σm + m;
                        σmt ⇐ σmt + m × Temperature of fluid leaving node t;
                }
                Mass flow through node ⇐ σm;
                Temperature entering node ⇐ σmt/σm;
        }

}
```

Figure 9.7. Logic for algorithm **NN5**.

The associated logic is shown in figure 9.7.

vi. The ambient temperature associated with a node may be set using member function

```
void nNode::AmbientTemperature (const double T,
                                boolean p);
```

Temperature `Ta` is assigned to the section referenced by the node. If argument `p` is true, the ambient temperature is propagated throughout the subtree rooted in the node by performing a postorder traversal of the subtree, and invoking function `nNode::AmbientTemperature (T, true)` for each node visited.

9.3.4. Class *Network*.

The preceding sections have developed a set of classes from which the various components of a tree network can be assembled. A further class is required to represent the network itself, and to provide a base from which classes representing more specialized kinds of tree networks can be derived. Class *Network* fulfils this purpose.

The implementation of class *Network* is straightforward. The class includes as member variables a pointer to the root node (a variable of class *nNode*), and a variable of the enumerated type

```
enum direction { Diverging, Converging };
```

which flags whether the flow is away from or towards the root. Two constructors are provided[110]. One (the default constructor) takes no arguments, and simply initializes the member variables to default values. The other takes as arguments a pointer to the root node, and the flow direction. A destructor deletes the root node, thus recursively deleting the tree. In addition, member functions are provided to solve the sub-problems defined in section 9.2, as follows:

i. Function

```
double Network::SolveFlowDistribution ();
```

solves problem **NF** for the network by calling member function `nNode::SumFlow ()` for the root node. The mass flow through the root node is returned.

ii. Function

```
void Network::SolveTemperatureDistribution (double t = 0.0);
```

finds the temperature and mass flow distributions through the network by calling the member function for the root node which is appropriate to the flow direction. This will be either function `nNode::TemperatureConverging ()` in the case of converging flow (problem **NT2**), or `nNode::TemperatureDiverging (t)` for the case of diverging flow (problem **NT1**). In the latter case, the argument specifies the temperature at the root node. For converging flow recursive invocation of algorithm **NN5** simultaneously solves for the mass flow distribution. For diverging flow, the mass flow distribution is found by calling function `Network::SolveFlowDistribution ()` before solving for the temperature distribution.

iii. Function

```
void Network::SolvePressureDistribution (boolean sT);
```

determines the pressure distribution by invoking member function `nNode::Pressure (P, sT)` (q.v.) for the root node, with $P = 0$.

Member function

```
void Network::AmbientTemperature (const double Ta);
```

assigns a uniform ambient temperature to all nodes of the network by invoking member function `nNode::AmbientTemperature (Ta, true)` for the root node. This is currently the only means of setting the ambient temperature for a network within Zebra, and while

---

[110] Since this is an abstract base class, the default constructor is **protected**, and may only be invoked by a derived class.

it is recognized as being somewhat rudimentary, given the present lack of a more sophisticated means of calculating the space loads and communicating the results to Zebra, any attempt to improve upon it is probably not justified.

9.4. Modelling Duct Systems for Moist Air.

The preceding section described a set of base classes which may be used to model tree networks of a fairly general nature. These classes are not self-contained, and indeed classes *Fitting* and *Section* are abstract classes; pure **virtual** member functions declared in these classes must be defined in a derived class before this set of classes will be useable. In addition to redefining **virtual** functions declared in the base class, a derived class may also seek to add features to the base class, appropriate to the specific entity it is desired to model.

This section describes the modifications and enhancements which are required to derive a set of classes which can be used to simulate an air conditioning duct system conveying moist air. The presence of humidity in the air requires that modifications be made to the algorithms described in the preceding section for computing pressure loss and heat transfer in a section of the network. The considerations involved in implementing a model of a duct system are not however restricted to those arising from the nature of the working fluid. Other important issues need to be addressed, including:

i. Handling duct sections of various cross-sectional types.

ii. Modelling pressure losses through a representative range of duct fittings.

iii. Controlling pressure drops through various paths in the duct system to achieve pressure balancing.

iv. Specifying and building a model of a duct system.

These issues are addressed in the following discussion, which relates to a *generic* air conditioning duct system. Class *Duct* is derived from class *Network* to provide the additional facilities required to model air conditioning duct systems. The nodes are of class *dNode*, which is a subclass of *nNode*. Considerations specific to supply and return air duct systems can be handled by deriving appropriate classes from the generic base class *Duct*. This issue will be addressed in the following section.

9.4.1. Air Conditioning Duct Sections.

Sections of ductwork are modelled as objects of class *DuctSection*, which is derived from class *Section*. This in turn serves as a base class for two further classes, *RoundSection* and *RectangularSection*. These latter classes essentially provide a 'front-end' to the underlying base class. Two constructors are provided for each subclass. The first constructor permits client code to specify the dimensional data and physical characteristics

of the section. The constructor for class *RectangularSection* is thus defined by the statement:

```
RectangularSection::RectangularSection (double h,
                                        double w,
                                        double l,
                                        double epsilon = 0.0,
                                        unsigned ins = 0);
```

where `h`, `w` and `l` are the internal height and width and the length of the section, `epsilon` is the internal roughness, and `ins` is an index specifying the insulation (see section 9.4.1.3). For class *RoundSection*, the internal diameter replaces the height and width in specifying the cross-section; otherwise, the argument list is the same. The alternative constructor provides a means of constructing a section (and indeed a network) from a set of specifications contained in a text file. Constructors of this type will be considered in section 9.4.4. Regardless of which constructor is used to declare the derived section, the underlying object of base class *DuctSection* will be created by invoking the appropriate constructor:

```
DuctSection::DuctSection (double df,
                          double dv,
                          double pi,
                          double pe,
                          double ai,
                          double l,
                          double eps,
                          unsigned ins);
```

where `df` is the equivalent-by-friction diameter, `dv` is the equivalent-by-velocity diameter, `pi` is the internal perimeter, `pe` is the external perimeter, `ai` is the internal cross-sectional area, and the significance of remaining parameters is as above. Where a constructor of the first type is used for the derived class, the appropriate dimensional arguments are calculated and passed to the constructor for the base class. For a rectangular section, equation (9.3.3) is used for D_f, and equation (9.3.6) is used for D_v. Where the specifications are read from a text file, the constructor for the base class is invoked using a default set of arguments, and the actual values of the member variables subsequently calculated as the information becomes available. Apart from the constructors, the interfaces for the derived classes only provide member functions to enable the user to read the defining cross-sectional dimensions; all internal calculations are conducted on the assumption that the duct section is of round cross-section.

9.4.1.1. Pressure Losses.

The methodology for computing the pressure loss within a section of ductwork has been described in section 9.3.1.1, and is fully implemented in the base class *Section*. It remains for the derived class to provide a function

```
virtual double DuctSection::Reynolds ();
```

which overrides the pure **virtual** function of the same name in the base class, and calculates the Reynolds number, as defined by equation (9.3.5). Relationships for calculating the density and viscosity of moist air have been provided in chapter 4. It should be borne in mind that these properties are functions of the temperature and humidity content of the air. In the interests of accuracy, the temperature distribution through the duct system should thus be evaluated before the pressure loss is evaluated.

```
Function TemperatureConverging:
{

        if node has child nodes then
        {
                σm ⇐ 0;                                          (Mass flow through node)
                σmh ⇐ 0;                          (Mass flow through node × Entering enthalpy)
                σmW ⇐ 0;                  (Mass flow through node × Entering humidity ratio)
                for each child node d do
                {
                        d->TemperatureConverging ();
                                                  (Call function recursively for child node)
                        t ⇐ Temperature of air leaving node d;
                                        (Calculate temperature rise through child node)
                        W ⇐ Humidity ratio of air leaving node d;
                        h ⇐ Enthalpy of air leaving node d;          (Find h from W and t)
                        m ⇐ Mass flow through node d;
                        σm ⇐ σm + m;
                        σmh ⇐ σmh + m × h;
                        σmW ⇐ σmW + m × W;
                }
                Mass flow through node ⇐ σm;
                h ⇐ σmh/σm;
                W ⇐ σmW/σm;
                Temperature of air entering node ⇐ t;       (Find t from W and h)
                Humidity ratio of air entering node ⇐ W;
        }

}
```

Figure 9.8. Algorithm **NN5** modified for psychrometric mixing.

9.4.1.2. Heat Transfer.

In calculating the temperature distribution within a duct system conveying moist air it is necessary to calculate the humidity ratio distribution simultaneously. This poses no problems for a diverging flow for which the air properties at the root node have been specified (problem **NT1**); the humidity ratio is simply propagated unchanged to all parts of the network, and may be accounted for by a simple modification to algorithm **NN4**. In the case of converging flow however, the flow converging at each junction must be *psychrometrically* mixed. This is performed using a modified form of algorithm **NN5**, the logic for which is shown in figure 9.8. The new algorithms are implemented as functions `dNode::TemperatureConverging ()` and `dNode::TemperatureDiverging (T)`, which override the functions of the same name in the base class.

Index	a	b	Description
0	0·0580	3·3093	Uninsulated sheetmetal
1	0·0911	1·5510	Liner, mechanically fastened, air side spray coated, 12mm, 32kg/m³
2	0·0949	1·0455	Liner, mechanically fastened, air side spray coated, 25mm, 24kg/m³
3	0·0863	0·9268	Liner, mechanically fastened, air side spray coated, 25mm, 32kg/m³
4	0·0604	0·9811	Liner, mechanically fastened, air side spray coated, 25mm, 48kg/m³
5	0·1023	0·5956	Liner, mechanically fastened, air side spray coated, 50mm, 32kg/m³
6	0·0131	0·9898	Rigid fibrous glass board
7	0·0076	0·9133	Externally insulated sheet metal (50mm, 12kg/m³ fibrous glass, faced), 50% compression
8	0·0082	0·7216	Externally insulated sheet metal (50mm, 12kg/m³ fibrous glass, faced), 0% compression
9	0·0982	0·6085	Flexible duct, pervious liner
10	0·0000	1·0438	Flexible duct, impervious liner

Table 9.1. Properties of insulating materials described by ASHRAE (1989; Fundamentals, chapter 32).

9.4.1.3. Insulation.

Curves showing the variation of overall heat transfer coefficient as a function of the mean velocity of the air within a duct have been published by ASHRAE (1989; Fundamentals, chapter 32) for a number of common insulating materials. These can be approximated by a linear fit of the form

$$U = aV + b \qquad (W/m^2.K) \quad . \tag{9.4.1}$$

Values of the coefficients a and b for the insulating materials described by ASHRAE are tabulated in table 9.1. A member function

```
virtual double DuctSection::Uc ();
```

which overrides the pure virtual function of the same name in the base class, and returns the value of the overall heat transfer coefficient corresponding to the current air velocity in the duct section. The insulation type is identified by an index (member variable

`insulation`), corresponding to the first column of table 9.1. The range of insulation types supported is necessarily small, and should be enlarged in future. In particular, it is desirable to provide support for a library of user-defined insulation characteristics.

9.4.2. Duct Fittings.

The appendix to chapter 32 of ASHRAE Fundamentals, 1989, tabulates the loss coefficients for an extensive range of duct fittings. The fittings described therein are classified into seven groups, of which six are of interest in the following (treatment of the seventh group, fan-system connections, will be deferred until chapter 10). These are:

Group 1 : Entries
Group 2 : Exits
Group 3 : Elbows
Group 4 : Transitions
Group 5 : Junctions
Group 6 : Obstructions

Two important subgroups of group 6 may be identified; dampers and gates. This system of classification is quite appropriate for the present purposes, and has been used as the

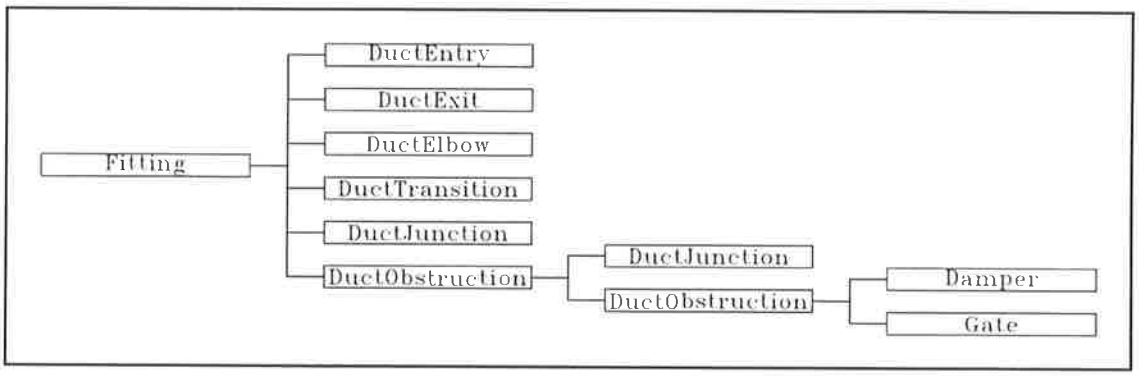


Figure 9.9. A hierarchy of classes to represent air conditioning duct fittings.

basis for a hierarchy of classes (figure 9.9). The purpose of this hierarchy of classes is to support the classification system of the ASHRAE Fundamentals, and to provide a means of calculating the local loss coefficient through any given fitting as a function of the flow conditions in the duct section to which it belongs. The entire range of fittings, with a minimal number of exceptions[111], is supported. Some reservations must be expressed regarding the ASHRAE tabulations. The data have been collated from a variety of sources, and are presented without any comment regarding their accuracy. For the same reason, their is considerable variation in the format of presentation, even within a group. Nevertheless, the classificatory scheme suggested by ASHRAE is appropriate, and the

[111] Class *Gate* has been implemented, but has not been made available to Zebra users, since it is considered that dampers form a better means of controlling flow. Of the remaining fittings, all but fitting 6-9 are supported.

manner in which the data are tabulated is convenient. The present set of classes has been developed as a basis which can be further explored in the future[112].

In constructing a model of a duct system, and subsequently calculating the pressure drops through the system, the question arises as to which duct section owns a particular fitting. In the case of exits, entries, obstructions and dampers there is no ambiguity in this regard; the fittings can be referenced to one and only one section. For these fittings, the pressure loss is calculated as

$$\Delta p_t = C\frac{\rho V^2}{2g_c} \qquad \textbf{(9.4.1)}$$

The remaining classes of fitting (elbows, transitions and junctions) join two or more sections which may have different cross-sectional areas and different flow rates through them. In calculating the pressure loss, it is essential that the value of C used be referenced to the section in which the flow velocity in equation (9.4.1) is measured. The possibility of ambiguity arising is resolved as follows.

i. Elbows and transitions. A duct system is constructed (section 9.4.4) commencing at the root. Each elbow or transition is regarded as a *terminator* for the adjoining section closer to the root of the system, to which it belongs. This convention applies regardless of the direction of flow through the duct system. A pointer to the object representing the fitting is stored on the list of fittings of the section to which it belongs. In addition, each object of class *DuctElbow* or class *DuctTransition* maintains a pointer to *both* adjoining sections. Let subscript o denote the section to which the fitting belongs, and subscript i denote the other adjoining section. Where the loss coefficient calculated from the ASHRAE tabulations is referenced to section i, the required coefficient may be derived as

$$C_o = C_i\left(\frac{V_i}{V_o}\right)^2 \qquad \textbf{(9.4.2)}$$

---

[112] Miller (1971, 1978) provides an extensive collation of experimental measurements with the results classified according to their predicted accuracy. Further analysis of a wide range of fittings can be found in Ward-Smith (1980), which provides a thorough review of the entire field of internal flow. Computational fluid dynamics (using a commercial package such as PHOENICS) offers a further means of exploring loss coefficients in fittings of arbitrary geometry, and covering a wide operating envelope. Furthermore, it provides a means of systematically investigating the effect of interaction between closely-coupled fittings, a topic on which experimental data is almost non-existent. Note also that since this work was undertaken ASHRAE has considerably extended the duct fitting database to include some 228 fittings of round and rectangular cross-section, with provision to extend the selection to include fittings for oval ducts in future (see the 1993 Edition of the ASHRAE Fundamentals Handbook). The database is now available in electronic format for linkage with duct design programmes, and its suitability for use with the Zebra package should be investigated.

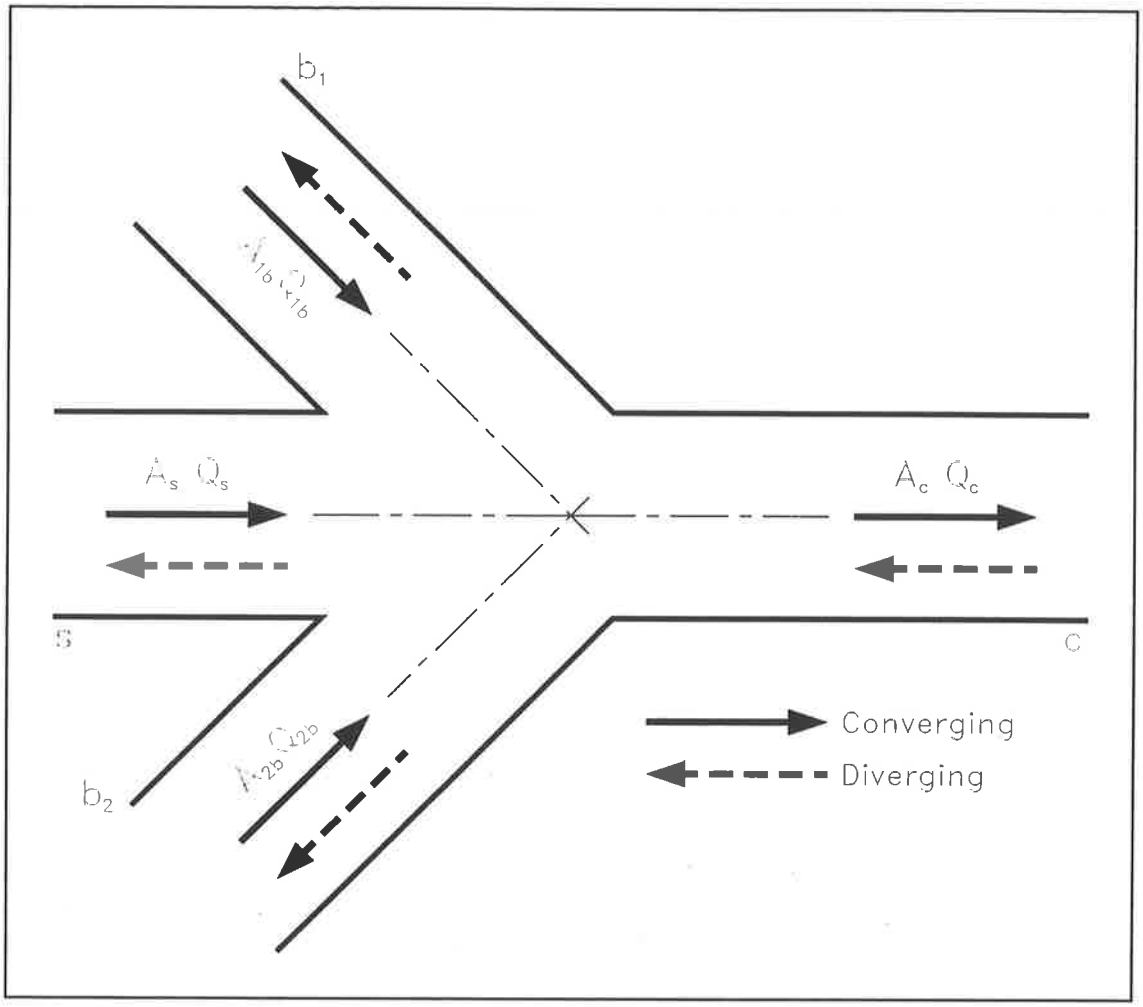


Figure 9.10. Notation to indicate the sections joined by a duct junction in the most general case.

ii. Junctions. In the most general case described by ASHRAE, a junction (figure 9.10) may join up to four sections, identified by the nomenclature:

c : Combined section. This is always the section nearest the root.
s : Straight section; continuation of the combined section.
b_1, b_2 : Branch sections.

Of these, either section s or section b_2 may be omitted. The fitting is regarded as a terminator for section c, and a pointer to the object representing it is stored on the list of fittings for that section. The junction will also cause a pressure drop in each branch, and a pointer to its representative object must be stored on the fitting lists for each branch (but *not* for section s)[113]. The object representing the junction

[113] When the destructor for an object of class *DuctSection* is invoked, the destructor for each object on its list of fittings will be invoked in sequence. Since a pointer to a specific junction will be stored by several sections, precautions must be taken to ensure that the junction is deleted by the destructor for the combined section only, since any attempt to delete an object which has already been deleted will be catastrophic.

maintains a pointer to each adjoining section. In the ASHRAE tabulations, the loss coefficients are in each case referenced to section *s*. Thus, $C_{c,s}$ is the coefficient for a pressure loss through the straight section, referenced to the combined section, and $C_{c,b}$ is the loss coefficient for a branch section, again referenced to the combined section. The latter coefficient may be referenced to a branch using the relationship

$$C_b = C_{c,b}\left(\frac{V_c}{V_b}\right)^2 \quad . \tag{9.4.3}$$

In the ASHRAE classification scheme, each type of fitting may be identified by a reference containing two components, the first of which identifies the group to which the fitting belongs, while the second provides an index within the group. Thus, code 4-3 specifies a fitting identified as *"Transition, Rectangular, Three Sides Straight"*. When a constructor for a subclass of class ***Fitting*** is invoked, the components of the code are stored in separate fields within the base class. Various other items of information must be supplied when an object representing a fitting is created. Pointers to the section to which it belongs, together with any adjoining sections, will enable the dimensions of the fitting to be determined. The appropriateness of specifying a particular fitting will also be determined by a consideration of the sections to which it is attached; one cannot connect a square fitting to a round duct section. Additional information may be specified by one or more optional parameters. Thus, the constructor for class ***DuctJunction*** is defined by the statement

```
DuctJunction::DuctJunction (DuctSection* sc,
                            DuctSection* ss,
                            DuctSection* sb1,
                            DuctSection* sb2,
                            unsigned ind,
                            double P1 = 0,
                            double P2 = 0);
```

where `sc`, `ss`, `sb1` and `sb2` are pointers to the objects representing the adjoining sections (`ss` or `sb2` may be zero), `ind` is the index of the fitting within group 5, and `P1` and `P2` are optional parameters which default to zero.

When a constructor for one of these classes is invoked, certain actions are performed. A series of checks are performed to determine whether the fitting is suitable for use in conjunction with the sections to which it is joined, and in the network type (converging or diverging) which is being constructed. Secondly, steps are taken to facilitate calculation of the local pressure loss coefficient during the simulation process. For this purpose, fittings may be classified into two broad categories; those for which the loss coefficient is solely a function of the geometry of the fitting, and those for which it is also determined by the flow velocity. For those fittings in the first category, the loss coefficient may be stored as a simple number, calculated when the object representing the fitting is created. The coefficient is extracted from the lookup tables using a low-order polynomial interpolation algorithm in one or more dimensions, as appropriate (Press et al., 1988). For those fittings for which the loss coefficient is velocity dependent, it proves possible in all cases tabulated in the ASHRAE Fundamentals to extract from the tables a one-dimensional

array expressing the loss coefficient as a function of a velocity-dependent parameter, once the geometry of the fitting has been specified. Again, polynomial interpolation is used. The loss coefficient may be variously expressed as a function of velocity, Reynolds number, or (for junctions), a ratio of flow rates or velocities. This is again indicative of the multiplicity of disparate sources which have been used in assembling the ASHRAE tables.

With the exception of dampers and gates, the fittings which we have described are not adjustable. The pressure loss coefficient is determined solely by the fixed geometry of the fitting, and the flow velocity. Dampers and gates are the means by which we seek to control the flow through various parts of the duct system by imposing an adjustable pressure loss coefficient. Accordingly, it is desirable that the interfaces for classes *Damper* and *Gate* provide access to a number of facilities in addition to those provided by the classes describing fixed fittings. The following discussion relates specifically to class *Damper*; a parallel set of facilities are provided by class *Gate*.

The ASHRAE Fundamentals provides performance data for four types of damper:

i. Simple butterfly damper for a round duct section.
ii. Simple butterfly damper for a rectangular duct section.
iii. Multiple-blade damper for a rectangular duct section, having parallel blades.
iv. Multiple-blade damper for a rectangular duct section, having opposed blades.

In practical applications, the latter two types are the most important. The most basic operations we may wish to perform upon a damper are those of setting the blades to a specific angle (θ), and setting the damper fully open ($\theta = 0$). These facilities are provided by the following functions:

```
void Damper::Open ();
void Damper::SetAngle (const double theta);
```

the implementation of which is trivial. In the process of controlling and balancing the flow through a duct system, it is also necessary to be able to adjust the damper setting to produce a desired pressure drop. This operation is performed by a function

```
double Damper::Angle (const double dP);
```

which calculates and returns the blade angle necessary to produce a specified pressure drop, expressed as the additional pressure drop required in addition to the wide-open value. In the tabulations presented for dampers in ASHRAE Fundamentals, the loss coefficients are expressed as function of damper dimensions and blade angle alone; no velocity dependence is implied. Thus, with the damper dimensions fixed, we can extract a one-dimensional array from the lookup tables expressing loss coefficient as a function of blade angle, thus:

$$C_\theta = f(\theta) \quad . \tag{9.4.4}$$

Given a target loss coefficient C, which is related to our target pressure drop by equation (9.4.1), we seek then to solve

$$g(\theta) \equiv C - C_\theta = 0 \quad . \tag{9.4.5}$$

The solution is readily found by invoking algorithm **zero** (Brent, 1971) with the bracketing interval defined by the first and last elements of the one-dimensional array.

The above procedure will readily generalize to handle the case where the damper performance data shows a velocity dependence.

9.4.3. Temperature and Humidity Distribution.

Class *Duct* defines a member variable `a`, which is of type *AirState* (chapter 4) to store the condition of the air at the root of the duct system. The mass flow, temperature and humidity distributions within a duct system can be calculated by invoking member function

```
void Duct::SolveAirDistribution ();
```

For diverging flow:

a. The value of variable `a` must be set before calling the above function.
b. A solution is first obtained for the mass flow distribution by calling member function `Network::SolveFlowDistribution ()` for the root node.
c. The humidity ratio in the section referenced by the root node is set equal to the value specified by `a`, and member function `dNode::TemperatureDiverging (T)` called for the root node, with the dry-bulb temperature of variable `a` assigned to argument `T`. The temperature and humidity distributions throughout the duct system are thus found.

For converging flow:

a. The dry bulb temperature and humidity ratio for each terminal node must be set before the above function is called.
b. The temperature and humidity distributions through the duct system are found by invoking member function `dNode::TemperatureConverging ()`. Note that this function simultaneously finds the mass flow distribution in the system.
c. The dry-bulb temperature and humidity ratio of the air leaving the root node are assigned to member variable `a`.

In either case, the mass flow of air through the terminal nodes must be set before invoking function `Duct::SolveAirDistribution ()`.

9.4.4. Pressure Balancing of Duct Systems.

Algorithm **NN3**, used in conjunction with algorithm **NN2** (section 9.3.3) provides the basic computational tool for evaluating the pressure distribution through a subtree rooted in a particular node. Additional considerations need to be taken into account when evaluating the pressure distribution in an air conditioning duct system. For a supply air duct system (diverging flow) the most important of these are:

i. The pressure at which the air is discharged from each terminal of the duct system is determined by the desired operating pressure of the zone which the terminal serves. This will usually be atmospheric pressure, or slightly higher to counteract the possibility of infiltration of outdoor air into the space.

ii. The pressure drop through each path of the system must be balanced to suit the pertinent flow rates. This is achieved through the use of dampers, or other adjustable pressure drops.

Similar considerations apply in the case of return air duct systems. For the moment, the discussion will for the most part relate specifically to the supply air situation.

The supply air duct system model supported by Zebra can be understood by reference to figure 9.11, which shows the duct system arrangement for (a) a three-zone system, and (b) a single-zone system. In this model:

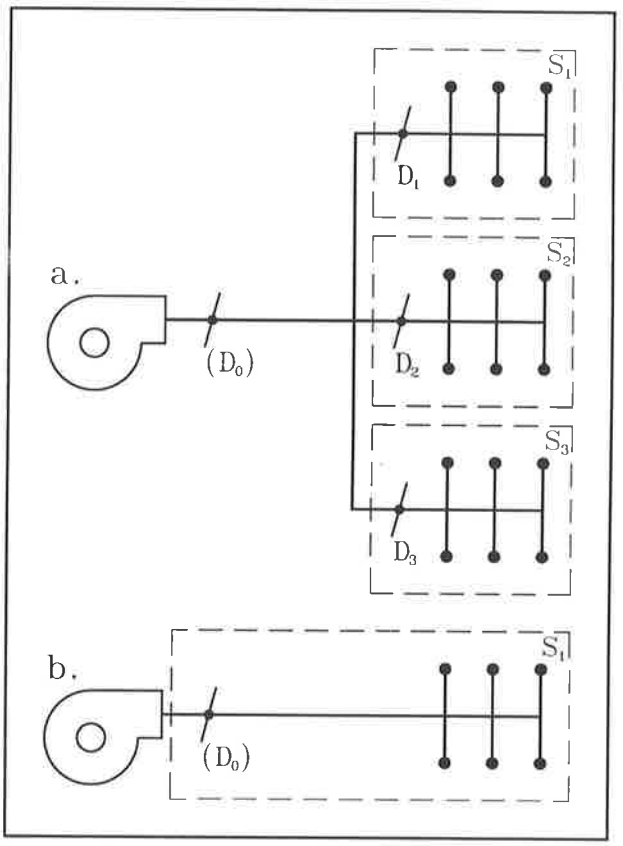


Figure 9.11. Layout of subtrees for (a) a multizone and (b) a single-zone system.

i. The duct system may optionally have a damper in the root node, which is typically used in a supply air duct system as a discharge damper to control the fan (chapter 10). In a return air or outdoor air duct system, this damper will be used to control the flow through the duct system (see section 9.5).

ii. Each zone is served by one or more terminal nodes. It is assumed that the flow through each terminal serving a particular zone is identical, and that balance among the terminals serving a zone is maintained by an adjustable pressure drop of unspecified type. Furthermore, at balance the additional pressure drop thus imposed on at least one terminal serving each zone will be zero.

iii. The terminals serving each zone comprise the terminal nodes of a subtree rooted in a node with a damper. For a multizone system, member variable `nNode::sub` (section 9.3.3) is *true* for the root node of each subtree. The damper may be omitted from the root node for a single zone system; if present, its purpose is to

control the fan, rather than to achieve balance[114]. For both single and multizone systems, member variable `nNode::sub` is *false* for the root node of the tree. When balance is achieved for a multizone system, at least one of the balance control dampers will be fully open.

Two additional classes are defined to support the computation of the pressure distribution in a balanced air conditioning duct system. The interfaces for these are listed in their entirety in the following. Class *Terminal* provides a representation for a terminal node in a duct system. The interface for this class is defined as

```
class Terminal
{
  dNode* d;      // Pointer to the terminal node
  double p;      // Extra pressure drop (Pa) required to balance flow
                 // through terminal
public:
  Terminal (dNode* dn) : d (dn), p (0.0) {}
  ~Terminal () {}
  dNode& Node () { return *d; }
  double Temperature () const;
  void Temperature (const double t);
  double HumidityRatio () const;
  void HumidityRatio (const double W);
  double MassFlow () const;
  void MassFlow (const double m);
  double DP () const { return p; }
  void DP (const double dP) { p = dP; }
  double PressureDrop ();
};
```

The member variables for this class include a pointer to the associated terminal node, and the additional pressure drop which must be imposed upon the flow through this terminal to equalize the flow through all terminals serving a zone. This latter variable may be set and read using the member functions `DP`. The remaining member functions are for the most part self-explanatory, and include functions to read and set the temperature, humidity ratio and mass flow rate of the air discharged from a terminal. Member function `Terminal::PressureDrop ()` returns the pressure at the terminal, defined as (see section 9.3.3) $P' = P + \Sigma\Delta P$, where P is the base pressure for some node established as a datum (the root of either the tree representing the duct system, or a subtree containing the terminal), and $\Sigma\Delta P$ is the sum of the pressure drops over a path joining, and including, the datum node and the terminal node.

Referring to figure 9.11, it is computationally convenient to define a class describing, and defining a series of operations pertaining to a subtree. For the purposes of the present definition, those subtrees which are of interest to us are delineated in the case of a multizone system by a root node which contains a damper, and one or more terminal nodes. In such a system, the root node of a subtree cannot also be the root node of the tree describing the duct system to which it belongs. For a single-zone system, the sole subtree is identical with the the tree describing the duct system.

---

[114] The damper is therefore regarded as belonging to the duct system, rather than the subtree; see later.

A subtree is described by the following class:

```
class Subtree
{
  dNode* root;              // Pointer to the root node
  Splist<Terminal> t;       // List of pointers to terminals
  double dP;               // Pressure drop (Pa) through balanced subtree
  double p;                 // Operating pressure (Pa) of space
  Damper* damper;           // Pointer to the damper
public:
  Subtree (dNode* r,
           Damper* d = 0)
  : dP (0.0), p (0.0), root (r), damper (d) {}
  ~Subtree () {}
  dNode& Root () { return *root; }
  Splist<Terminal>& T () { return t; }
  Damper* D () { return d; }
  void PressureDrop (const double dp) { dP = dp; }
  double PressureDrop () const { return dP; }
  void Pressure (const double p) { Subtree::p = p; }
  double Pressure () const { return p; }
  void MassFlow (const double m);
  void Balance ();
};
```

The above class maintains pointers to the root node and damper, and a list of pointers to the associated terminals. Variable `dP` is the pressure drop (Pa) between and including the root node and each terminal node in a *balanced* subtree. Variable `p` is the operating pressure (Pa) of the space into which each terminal node of the subtree discharges, referenced to atmospheric pressure. Member function `Subtree::MassFlow (m)` sets the mass flow rate through the subtree by apportioning the specified quantity equally among the terminal nodes. The member function of major interest is `Subtree::Balance ()`, which evaluates the pressure distribution through a balanced subtree. This function performs the following operations:

i. If the subtree is controlled by a damper, the damper is set fully open.

ii. The pressure distribution through the subtree is evaluated by invoking function `nNode::Pressure (P)` (Algorithm **NN3**; section 9.9.3) for the root node of the subtree. The pressure datum for the subtree is set to 0 by invoking the function with the argument `P` = 0. The pressure distribution so evaluated is not yet balanced.

iii. The list of terminal pointers is traversed, and the pressure drop for each path connecting the root node to a terminal node (returned by function `Terminal::PressureDrop`) is compared. Variable `Subtree::dP` is set equal to the maximum path pressure drop (ΔP_{max}).

iv. The list of terminal pointers is again traversed, and the additional pressure drop through each terminal is calculated. That is, for terminal i, the balance pressure is calculated as

$$\Delta P_i' = \Delta P_{max} - \Delta P_i \qquad \textbf{(9.4.6)}$$

The problem of balancing the entire duct system may now be tackled. Class *Duct* has two member variables which are of particular significance in balancing a duct system:

```
class Duct : public Network
{
  ⋮
protected:
  ⋮
  Map<unsigned, Subtree*> Dp;
  double P;
  ⋮
public:
  ⋮
  void Balance ();
};
```

If the duct system contains n subtrees (in the sense defined above), a pointer to the object representing each is stored in the associative array `Duct::Dp`, together with a unique index in the range [1..n]. Member variable `Duct::P` stores the pressure at the root of the duct system, *referenced to atmospheric pressure*. Assume that all dampers in the system are fully open, and that the flow within each subtree has been balanced by invoking function `Subtree::Balance ()`. It is now possible to find the pressure distribution throughout the entire duct system, and in particular to evaluate the pressure loss (ΔP_i) between the root node and the terminal nodes for each subtree i. If the operating pressure of each zone is set to atmospheric pressure, then the pressure at the root of the system, *referenced to atmospheric*, is

$$P = \Delta P_{max} = \max_{1}^{n} \Delta P_i \qquad \textbf{(9.4.7a)}$$

for a diverging system, and

$$P = -\Delta P_{max} \qquad \textbf{(9.4.7b)}$$

for a converging system. In the more general case the operating pressure for the various zones will not necessarily be equal to atmospheric pressure, and the duct system may in fact serve zones operating at different pressures. In this case, the situation becomes slightly more complex. Let p_i be the operating pressure for the zone served by subtree i, and $\Delta P_{d,i}$ be the additional pressure loss which must be induced by partially closing the damper if balance is to be achieved throughout the system. That is

$$\Delta P_{d,i} = |P - \chi \Delta P_i - p_i| \qquad \textbf{(9.4.8)}$$

where χ takes a value of +1 for a diverging network, and -1 for a converging network[115]. Thus, we wish to find P such that

$$\Delta P_{d,min} = \min_{1}^{n} \Delta P_{d,i} = 0 \quad . \qquad \textbf{(9.4.9)}$$

---

[115] Bear in mind that in the convention adopted here, pressure drops are *always* positive. Pressures P and p_i are gauge pressures, referenced to atmospheric. Generally, P may be expected to be positive for supply air networks, and negative for return air and outdoor air networks.

Pressure balancing for the entire duct system is performed by **virtual** member function `Duct::Balance ()`, the logic for which is shown in figure 9.12.

The index i' for which equation 9.4.9 is satisfied will be stored, as will the system characteristic, K_{ts}, defined by

$$\Delta P_{sys} = \left(\frac{Q_s}{K_{ts}} \right)^2 \qquad \textbf{(9.4.10)}$$

where $\Delta P_{sys} = \Delta P_{i'}$ is the appropriate pressure loss to be used in evaluating K_{ts} in the above.

A damper in the root node (D_0 in figure 9.11) may be used for fan control in the case of a supply-air duct, and for balancing the system in the case of any of the duct systems (see chapter 10). Let P_0 be the pressure at the root of the duct system with this damper fully open, and let P' be the pressure which we require at the root node for the designated control purpose. Then it is necessary to close damper D_0 to induce a pressure drop ΔP^+ in excess of that corresponding to the fully open position, where

$$\Delta P^+ = P_0 - P' \qquad \textbf{(9.4.11a)}$$

for a converging system, and

$$\Delta P^+ = P' - P_0 \qquad \textbf{(9.4.11b)}$$

for a diverging system. The damper setting required to achieve this may be found by calling member function

```
double Duct::AdjustDamper (const double dP);
```

which in turn invokes function `Damper::Angle (dP)` to find the blade angle required to effect the desired control action.

9.4.5. Specifying and Building a Model of a Duct System.

The preceding sections have described a set of operations to manipulate a duct system model which is already resident in computer memory in the form of a multiway tree. Initially however, the model must be assembled from a description supplied in the form of a sequential series of instructions. In the following a recursive strategy is described for interpreting a set of instructions to build a tree network describing a duct system and its associated fittings.

For this purpose, alternative constructors are supplied for classes *RectangularSection* and *RoundSection* (see section 9.4.1). For class *RectangularSection* the alternative constructor is defined by the statement

```
{
        if fan control damper is present then
                Open fan control damper fully;
        for each subtree i do
                Balance subtree;                          (Invoke Subtree::Balance)
        if single-zone system then                        (Set pressure at root of duct system)
        {
                if diverging flow then
                        P ⇐ P_1 + ΔP_1
                else
                        P ⇐ P_1 - ΔP_1
        }
        else
        {
                Calculate pressure drops through remainder of duct system;
               (Use Algorithm NN3 with P = 0, sT = false; resets datum values for subtrees)
                i_max ⇐ 0;
                P ⇐ 0;
                for each subtree i do                     (Find pressure at root of duct system)
                {
                        if diverging flow then
                        {
                                P' ⇐ p_i + ΔP_i;
                                if P' > P then { i_max ⇐ i; P ⇐ P'; }
                        }
                        else
                        {
                                P' ⇐ p_i - ΔP_i;
                                if P' < P then { i_max ⇐ i; P ⇐ P'; }
                        }
                }
                for each subtree i ≠ i_max do             (Balance flow through system)
                {
                        Δp ⇐ P - χΔP_i - p_i;
                                                  (χ = +1 for diverging flow, -1 for converging flow)
                        Set damper angle to give excess pressure drop ΔP;
                }
        }
}
```

Figure 9.12. Logic for an algorithm to perform pressure balancing for a duct system.

```
RectangularSection::RectangularSection (TextSource& t,
                                        Duct& d,
                                        int r = 0,
                                        int s = 0,
                                        RectangularSection* u = 0);
```

The constructor for class *RoundSection* is identical. The significance of the arguments in the above is as follows:

`t:` A reference to an object of class *TextSource* from which the duct specifications will be read (see section 2.4).

`d` : A reference to the object representing the duct system to which the section belongs.

`r` : Denotes the *level* within the network tree of the node which will be constructed for the section (see section 9.1). Thus, for the root node a level of 1 is specified for `r`, this value being incremented by 1 for each successive level of node.

`s` : Denotes the level of the node within the subtree to which it belongs. To create a node at the root of a subtree, a value of 1 is specified for argument `s`. For zones within a multizone system which belong to no subtree, a value of `s` = 0 is specified.

u : Contains a pointer to another object of the same class. If this argument is not null, it indicates to the constructor that the section which is being created must have the same cross-sectional dimensions as the section accessed by the pointer. Thus, for a rectangular section the height and width will match those of the referenced section; for a round section, the diameters will match. The ability to pass this type of information is useful in the case of certain types of elbows and junctions, where the cross-sectional dimensions of the duct downstream of the fitting must be the same as that upstream, and for various types of junctions, where the cross-sectional dimensions of the various branches must be the same. If this argument is null, the appropriate dimensions are extracted from the text source. Regardless of the source of the defining dimensions, the derived dimensions are then calculated and set, and control passed to a member function of the base class, `DuctSection::Input (t)`.

Class *Duct* maintains two **private** singly-linked lists which are used in constructing the network:

```
Splist<Terminal> t;      // List of terminal nodes
Splist<NodeLevel> dl;    // List of node levels
```

where *NodeLevel* is a simple class containing a record of a node and its level:

```
struct NodeLevel
{
  dNode* d;
  int level;
  NodeLevel (dNode* dn, int l) : d (dn), level (l) {}
};
```

Since class *DuctSection* is declared to be a **friend** of class *Duct*, it has full access to these lists.

Assembly of a duct system model is initiated from within the constructor for class *Duct* when a record prefixed with either of codes `DSECTION_BEGIN_ROUND` or `DSECTION_BEGIN_RECT`[116] is read, causing the appropriate constructor to be invoked[117]:

---

[116] Both of these codes may be omitted. The algorithms described in chapter 6 prescribe appropriate action to take if the duct system is not specified. In the early stages of a project, the information to specify the duct system may well not be available, and even in the later stages of the project, the

```
(void) new RectangularSection (t, &this, 1);
```

This function and the constructor for class *RoundSection* in essence serve merely to establish the cross-sectional dimensions of the section, before passing control to function

```
void DuctSection::Input (TextSource& t,
                         Duct& d,
                         int r,
                         int s);
```

where the arguments have the same meaning as previously. The fundamental layout of the function is the same as that described in section 2.4; records are extracted from the text source and processed according to their prefix code until a terminating record is read. Before execution enters this loop however, certain local variables are initialized:

```
boolean terminal = false;  // Indicates whether this is a terminal node
DuctSection* ss = 0;       // Pointer to the continuation section
DuctSection* sb1 = 0;      // Pointer to branch section 1
DuctSection* sb2 = 0;      // Pointer to branch section 2
Damper* damper = 0;        // Pointer to damper for this section
```

Within the loop, the action taken is dependent upon the value of the prefix code read. Three of the possible codes contain either control information, or information specific to the duct section itself:

`SECTION_DIMENSION` introduces a record specifying the length and (optionally) the internal roughness of the section. This record is mandatory.

`SECTION_INSULATION` introduces a record specifying the insulation type (see section 9.4.1.3). This defaults to 0 (uninsulated sheetmetal).

`SECTION_END` terminates input for the record. This record is also mandatory.

The remaining codes which are valid at this point specify duct fittings. The basic format of these records is similar. The code specifies the class of fitting to be created. This is followed by an index which identifies the exact type of the fitting. Finally, the record may contain a number of parameters specific to the fitting type, as determined by the class and index. Once a valid record has been read, an object of the appropriate class is created and a pointer appended to the list of fittings. In the case of junctions a pointer will also be appended to the list of fittings for one or more adjacent sections. Depending upon the fitting type, a number of other actions may be taken which will determine the manner in which the network is constructed. These will be specified below. For each section no

---

designer may feel that the effort required to specify the ductwork is not justified. In such a case, the supply and return air duct gains for each zone may be supplied by the user. If either of these codes is specified, the duct system must be specified in its entirety according to the rules established below.

[117] Keyword **this** is a pointer to the object for which we are currently executing a member function, in this case an object of class *Duct*.

more than one record with the same code may be specified, except in the case of code DUCT_OBSTRUCTION_BEGIN. Codes DUCT_ELBOW_BEGIN, DUCT_TRANSITION_BEGIN and DUCT_JUNCTION_BEGIN *terminate* a section. In the case of a diverging network, code DUCT_EXIT_BEGIN will also terminate a section, as will code DUCT_ENTRY_BEGIN in the case of a converging network. No more than one code for a terminator type may be specified for a section, and that code must be specified following all other codes specifying fittings for the section. The available classes of fitting are identified by the following codes:

DUCT_ENTRY_BEGIN introduces a record specifying a duct entry (class *DuctEntry*). For a diverging network one record with this code must be present if this section is at the root (`r = 1`), and must be specified before any the following fitting codes are specified. This code is invalid for sections at any higher level. For a converging network, the presence of a record with this code signifies that the section will be located at a terminal node of the tree (`terminal` ⇐ *true*).

DUCT_EXIT_BEGIN introduces a record specifying a duct exit (class *DuctExit*). For a converging network one record with this code must be present if this section is at the root (`r = 1`), and must be specified before any the following fitting codes are specified. This code is invalid for sections at any higher level. For a diverging network, the presence of a record with this code signifies that the section will be located at a terminal node of the tree (`terminal` ⇐ *true*).

DUCT_ELBOW_BEGIN introduces a record specifying an elbow (class *DuctElbow*). If a record with this code is read, it must be immediately followed by a code specifying the commencement of input for the adjacent section at the next higher level (code DSECTION_BEGIN_ROUND or code DSECTION_BEGIN_RECT, as appropriate). The specified constructor is then invoked to create a new object of appropriate type, and a pointer to the new object assigned to the local variable `ss`. In all cases except fitting 3-10 (elbow, rectangular section, with variable inlet and outlet areas) the inlet and outlet sections for the section are the same. Typically then, the constructor for the adjacent section is invoked by a statement such as the following[118]:

```
ss = new RoundSection (t, d, r + 1, s ? s + 1 : s,
                       (RoundSection*) this);
```

---

[118] The ternary operator (Kernighan and Ritchie, 1988) takes the form

conditional_expression ? *expression*$_1$: *expression*$_2$

and may interpreted as

`if` *conditional_expression evaluates to true* `then`
expression$_1$;
`else`
expression$_2$;

Invoking the constructor recursively as we have done here causes the entire subtree rooted in the node at the next higher level to be created before control returns to the statement following the constructor. Local variable `ss` is also passed as an argument to the constructor for the fitting.

`DUCT_TRANSITION_BEGIN` introduces a record specifying a transition between sections of different shape or cross-sectional area (class *DuctTransition*). The above remarks concerning elbows apply in this case also, with the exception that the duct section will always change between the input and output of the transition, and the final argument in the call to the constructor for the adjacent section will thus generally be omitted (default to 0).

`DUCT_JUNCTION_BEGIN` introduces a record specifying a junction (class *DuctJunction*). Depending upon its type, the junction may have up to three additional sections adjacent to it at the next higher level. In the most general case the constructors will be invoked in the sequence

```
ss = new RoundSection (t, d, r + 1, s ? s + 1 : s,
                       (RoundSection*) this);
 :
sb1 = new RoundSection (t, d, r + 1, s ? s + 1 : s);
 :
sb2 = new RoundSection (t, d, r + 1, s ? s + 1 : s,
                       (Roundsection*) sb1);
```

The above sequence of constructors is appropriate to the case where the main (continuation) section has the same section as the combined section, and each of the branches have the same section. Either of branches s or b_2 may be omitted in the above. The situation described earlier in connection with duct elbows is generalized in this case in that each subtree rooted in a node at the next higher level is created recursively by invoking the constructor for the section at the root of the subtree. Local variables `ss`, `sb1` and `sb2` are passed as parameters to the constructor for the fitting. A pointer to the fitting is appended to the list of fittings for each of the branch sections, as well as for the combined section.

`DAMPER_BEGIN` introduces a record specifying a damper (class *Damper*). Unless this section is at the root (`r` = 1), it will be designated as the root of a subtree in the narrower sense defined in section (9.4.3); `s` is assigned a value of 1. A damper may not be specified if the section is already designated as belonging to a subtree (`s` > 1). A pointer to the damper created is assigned to local variable `damper`.

`DUCT_OBSTRUCTION_BEGIN` introduces a record specifying a fixed obstruction (class *DuctObstruction*). Such fittings may be specified anywhere and in any combination, subject only to the general rules regarding placement defined above.

Having read the records pertaining to a particular section, a series of operations are performed to created an object of class *dNode* referencing the current duct system, to link the node to its children, and to update the housekeeping structures to permit this node in turn to be linked to its parent. The sequence of operations proceeds as follows:

i. If this section is at the root of the tree (`r` = 1), and a damper has been specified for the section (`damper` ≠ 0), the damper specified will be the fan control damper for the duct system. A pointer to the damper is assigned to member variable `Duct::d`, and a value of 0 is assigned to local variable `damper`.

ii. An object of class ***dNode*** referencing the section is created. A section will lie at the root of a subtree, defined in the narrower sense above, if `s` = 1, or if it lies at the root of the network for a single-zone system (`r` = 1 and list `Duct::t` not empty). For such sections, member variable `dNode::sub` for the associated ***dNode*** is assigned a value of *true*. Member variable `dNode::term` is assigned a value of *true* if this is a terminal node (`terminal` = *true*).

iii. At any point during the construction process, list `Duct::dl` may contain objects referencing nodes at the same level as the current node, or at the next higher level. These latter objects, which reference the child nodes for the current node, are located at the front of the list. Objects are removed from the front of the list in sequence, until either the list is empty, or an object referencing a node at the same level as the current node is removed. In the latter case, the object removed must be reinserted at the head of the list. Otherwise, the node referenced is inserted into the list of child nodes for the current node.

iv. An object of class ***NodeLevel*** is constructed taking a pointer to the current node, together with its level as arguments. This object is inserted at the head of list `Duct::dl`.

v. If the current node is a terminal node (`terminal` = *true*), a new object of class ***Terminal*** is created taking a pointer to the node as argument, and inserted at the head of list `Duct::t`.

vi. If the node has been designated to be the root of a subtree in step (ii) above, a new object of class ***Subtree*** is constructed taking a pointer to the current node and the local variable `damper` as arguments. The elements of list `Duct::t` will be the terminal nodes for the subtree; they are removed from that list and placed on the list of terminal nodes for the newly created object. Finally, an element of the associative array `Duct::Dp` is created, taking the next available index in the range [1..*n*] as its key, and a pointer to the object of class ***Subtree*** as its value.

The final step in assembling the network is taken in the constructor for class ***Duct***. It will be recalled that the construction process was inititated by a call to the constructor for one of the subclasses of class ***DuctSection***. When control returns from that function call, list `Duct::dl` contains one element which is removed, and the associated node pointer assigned to member variable `Duct::root`. Member variable `Duct::specified` is assigned a value of *true* to signify that a valid duct system has been specified.

9.5. Specific Duct System Types.

The preceding discussion has considered air conditioning duct systems from a generic point of view. It is convenient to classify the various types of duct system encountered in air conditioning practice according to the specific functions they perform. The following classifications are found to be useful:

i. Supply-air duct systems are used to deliver treated air from an air-handling unit (the *source*) to a number of *targets*. In the case of a distributed unit, the targets are the various zones controlled by the unit. In the case of a central outdoor air treatment plant, the targets are the distributed units supplied by the central unit. Every source will supply one and only one supply-air duct system; the mapping between the sources and the targets will be one-to-many. The flow in such systems *diverges*.

ii. Return-air duct systems are used to return air from the zones back to a distributed air handling unit. Each distributed unit will have at most one return-air duct system associated with it; 100% outdoor air units will have none. The flow in such systems *converges*.

iii. Outdoor-air duct systems are used to convey *untreated* outdoor air to a unit. All outdoor air units will be served by an outdoor-air duct system, as will standalone distributed units. In principle, the mapping between outdoor-air duct systems and the units they serve could be one-to-many; in the current study only a one-to-one mapping will be considered. Such systems are modelled as *converging* in the present study, thus establishing commonality with the preceeding types in that member variable `Duct::P` is the gauge pressure at the point at which the duct system joins the casing for the three types. This consideration is of importance in balancing the pressures in a system (chapter 10).

Classes have been derived from base class *Duct* to represent each of the above types of duct system. A fourth classification of duct system may be defined:

iv. Exhaust systems extract spent air from a system and exhaust it to the atmosphere at a rate which is equal to the rate at which outdoor air is supplied to the system, adjusted by an allowance for exfiltration or infiltration.

While duct systems of this latter type are of importance to the overall pressure balance of an air conditioning system, for the purposes of the present study, it has not proved necessary to model this type of duct system in detail[119]. The characteristics of the classes developed to describe the remaining duct system types are outlined below.

---

[119] For the purposes of the pressure balancing algorithms developed in this and the next chapter, it is only necessary to consider the fact that a specified quantity of exhaust air will be extracted from the system at a defined point. The exhaust system can be modelled as an independent entity according to the principles developed in the present chapter, if so desired.

9.5.1. Supply-Air Duct Systems (Class *SA_Duct*).

It is necessary to differentiate between those duct systems which supply treated air to zones, and those which supply treated air to distributed air units. Member variable `SA_Duct::dest` of an enumerated type

```
enum Dest { Z, D };
```

performs this function; `Z` indicates that the target for the treated air is a zone, `D` that it is a distributed unit. The target type is determined from the context when the specifications for the duct system are read from file. Topologically, the major difference between the two types of duct system is that in a system supplying air to distributed units each damper-controlled subtree will have only one terminal node; in a system supplying air to zones, each such subtree will in general have several terminal nodes. In either case, the mapping between the damper-controlled subtrees in the duct system and the targets will be one-to-one, and is stored in an associative array

```
Map<unsigned, unsigned> I;
```

in which the key contains an index to the target, while the value contains an index to the subtree.

In common with the other subclasses of class *Duct*, this class defines a member function to extract the specifications for the duct system from a text source. This function calls the corresponding member function of the base class, and then performs a series of input operations specific to the subclass. The class also provides member functions to perform the following major operations:

i. Set the mass flow rate through each damper-controlled subtree to the value demanded by the target. Where there are multiple terminal nodes within the subtree, the flow is aportioned such that each passes an equal amount.

ii. Set the condition of air at entry to the target equal to that at the terminal nodes of the corresponding subtree. Where there is more than one terminal node, the air condition which would result from psychrometrically mixing the air supplied by each is used.

9.5.2. Return-Air Duct Systems (Class *RA_Duct*).

The return-air duct system model used by Zebra is based on the assumption that the following conditions hold:

i. The air exhausted from the various zones served by a unit is gathered into one or more plenum chambers, which are located at the terminals of a common return-air duct system. Each plenum may have a number of zones associated with it; each zone will exhaust into one and only one plenum. The mapping between the plenum chambers and the damper-controlled subtrees is one-to-one.

ii. The air passing through the plenum is subjected to a temperature rise (0 by default), but is not subject to a pressure drop. Where several zones exhaust into a common plenum, the operating pressure within the plenum will be set equal to the lowest zone pressure; it is assumed that the extractors for the remaining zones incorporate some mechanism to induce the pressure drop necessary to achieve balance.

iii. Spill air may be extracted from the system at one of two points; from the plenum chamber, or from a location adjacent to the root of the duct system. Where the spill is exhausted from the plenum chambers, it is exhausted from each in proportion to the mass flow passing through the plenum.

The model outlined above offers considerable flexibility. In particular, the plenum may be regarded as a *notional* concept, and the situation where the air is removed from a zone by a *single* extractor without passing through a plenum is readily handled. The model is not however the most general possible. The case where the air is removed directly from a zone by multiple extractors is not handled. This is essentially the inverse of the supply-air situation, and the model could readily be adapted to handle that case using the methodologies described above. In addition, in principle the spill air can be extracted from any point in the return-air circuit. It would be possible to provide for the insertion of an extraction point at an arbitrary position in the return-air duct system, but at the cost of some loss of elegance in the solution algorithm. This feature may be implemented in a future release of the software. In its present form the model is adequate to handle the majority of systems with which the author and his colleagues have been concerned.

The plenum is modelled using a class, the interface for which is defined as follows:

```
class Plenum
{
  double dT;       // Plenum temperature rise (deg. C)
  double p;        // Operating pressure (Pa) referenced to atmospheric
  Slist<int> l;    // List of identification codes for zones feeding into
                   // plenum
  double m_in;     // Mass flow (kg/s) of air entering plenum
  double m_out;    // Mass flow (kg/s) of air leaving plenum
  AirState A_out; // State of air leaving plenum
public :
  Plenum () : dT (0.0), p (0.0), m_in (0.0), m_out (0.0),
              A_out (Network::pressure) {}
  ~Plenum () {}
  void TemperatureRise (const double dT = 0.0) { Plenum::dT = dT; }
  double TemperatureRise () const { return dT; }
  double OperatingPressure () const { return p; }
  double MassFlowIn () const { return m_in; }
  double MassFlowOut () const { return m_out; }
  double Spill () const { return m_in - m_out; }
  Slist<int>& Zlist () { return l; }
  AirState& AirOut () { return A_out; }
  void Set (double spill = 0.0);
};
```

In essence, this class is quite straightforward. The class uses a singly-linked list of pointers to reference the zones which feed into the plenum. The sole modifier for the class is

function `Plenum::Set (spill)`, which takes as it argument the mass flow rate of the air to be spilled *from the plenum*. This function performs the following actions:

i. The operating pressure for the plenum is set equal to the minimum of the operating pressures for the zones which feed into it.

ii. The mass flow of air into the plenum is obtained by summing the flows through the zones exhausting into the plenum. The mass flow of air leaving the plenum is equal to the mass flow entering less the spill (if extracted from the plenum).

iii. The state of the air leaving the plenum is found by psychrometrically mixing the entering air streams, and incrementing the dry bulb temperature by the prescribed amount[120].

Class *RA_Duct* maintains an associative array which establishes a one-to-one mapping between the damper-controlled subtrees in the duct system, and the plenum chambers:

```
Map<unsigned, Plenum*> I;
```

An additional member variable flags the location of the spill-air takeoff. This is of the enumerated type

```
enum Position { Rt, Pl };
```

where `Rt` indicates a location adjacent to the root of the duct system, and `Pl` indicates a location in the plenum. The class provides a set of member functions, the most important of which sets the mass flow rate and air state at each terminal node of the duct system by invoking function `Plenum::Set (spill)` for each plenum. Where the spill air is extracted from the plenum, the quantity to be spilled from each is established in proportion to the rate of flow entering each.

In order to establish the pressure balance for the entire fan/duct system it is necessary in the most general case to provide dampers controlling the flow through the root sections of both the return-air and outdoor-air duct systems. For any set of flow conditions, the position of one of these dampers may need to be modulated from the fully-open position to achieve balance. The means of achieving this have been described above in section 9.4.4.

---

[120] Many load estimation programmes, such as TEMPER (Australian Construction Services, 1987) can be used to provide an estimate of the heat gained by the air passing through a plenum. A detailed consideration of the heat transfer processes occurring within a plenum is provided by Kimura (1977).

9.5.3. Outdoor-Air Duct System (Class *OA_Duct*).

The outdoor-air duct model is relatively simple, and comprises a network having a single path, flow through which is controlled by a damper in the root section. The complement of member functions is correspondingly small. A member function

```
void OA_Duct::Set ();
```

is provided to set the operating pressure at entry to the sole damper-controlled subtree to 0 (atmospheric), the mass flow of air equal to the specified rate, and the condition of the air entering the terminal node equal to ambient conditions.

Pressure balance is effected in a manner similar to that described above for a return-air duct system.

9.6. Integration with the System Model.

In the algorithms described in chapter 6 it was assumed that *a priori* estimates are available for the temperature gains associated with heat transfer through the duct walls, and with the dissipation of fan energy. If the duct systems connected to a unit have been fully specified, as described in the preceding sections, it is possible to calculate these quantities as an integral component of the simulation process. To accommodate this process, the algorithms of chapter 6 must be modified accordingly.

Broadly speaking, three steps are involved in evaluating the above-mentioned temperature gains for a system:

i. The mass flow distribution for each duct system involved is found after assigning known flow rates to the terminal nodes of the duct system (problem **NF**).

ii. With the properties (temperature and humidity) of the air entering the duct system specified, the distribution of these properties through the duct system are found (problems **NT1** and **NT2**).

iii. The pressure loss distributions through the duct systems are found (problem **NP**), and the pressure balance among the various duct systems is found.

Step (iii) involves a consideration of pressure losses through components other than the duct system, principally filters and coils, and of the pressure gain necessary to offset the various losses, which is imparted by the fan. This step depends upon, and necessarily follows steps (i) and (ii) above. Further consideration of the calculation of the pressure balance will be deferred until chapter 10. In the following, solution of steps (i) and (ii) will be described, first for a distributed unit, and then for the much simpler case of a central outdoor-air treatment unit.

9.6.1. Distributed Units.

It will be recalled from the discussion of section 6.3.6 that the first step in initializing a system model prior to commencing a simulation run is establish internal consistency among the zones served by each supply-air connector. As has been shown, this may be handled in a straightforward manner if the duct temperature gains are known *a priori*.

The mass flow of air through a zone in a VAV system is determined by the temperature of the entering the zone (equation 6.1.9), which in turn depends upon the duct temperature gain. In this situation, steps (i) and (ii) above are interdependent, and an iterative procedure must be used to find the temperature gains. The supply-air condition is initially located as described in section 6.3.6 to ensure that the moisture staircase iteration proceeds from a psychrometrically feasible point. This will establish the condition of the air entering the duct system. For a single-zone system, we may then proceed in the following manner:

i. Let $\delta\tilde{t}$ be an estimate for the temperature gain experienced by the air in traversing the duct system. This establishes the temperature of the air entering the zone, and the mass flow of air through the zone, and hence the duct.

ii. The corresponding mass flow and temperature distributions for the duct system are found by invoking function `SA_Duct::SolveAirDistribution ()`. This gives an updated estimate $\delta\tilde{t}'$ for the temperature gain.

To solve the problem we seek to find some $\delta\tilde{t}$ such that

$$f(\delta\tilde{t}) = \delta\tilde{t} - \delta\tilde{t}' = 0 \qquad \textbf{(9.6.1)}$$

which may be solved using a root-finding procedure such as zero. A suitable bracketing interval $[a..b]$ is found by letting $b = a = 0$, and incrementing b in steps of 1 °C until we find some b for which

$$sign\ f(a) \neq sign\ f(b) \quad . \qquad \textbf{(9.6.2)}$$

The procedure readily generalizes to handle a multizone system, where we seek now to minimize

$$f_1(\delta\tilde{t}_1, \ldots, \delta\tilde{t}_n) = \sum_{i=1}^{n} (\delta\tilde{t}_i - \delta\tilde{t}_i')^2 \qquad \textbf{(9.6.3)}$$

This is achieved using procedure smsno (see section 6.3.3), where we let $\delta\tilde{t} = 0{\cdot}5$ °C as a first estimate.

For a CAV system, the mass flow rate of air through the zones, and hence through the supply-air duct system, is known[121]. We seek to find an initial estimate for the supply-air condition which is at once consistent with the zone thermostat settings, and psychrometrically feasible. We can guarantee that a trial point will satisfy the second condition if we confine our search to the locus described by all points for which the dew-point temperature, $\tilde{t}''_{SA} = \tilde{t}_{SA} - \delta t$, where $\delta t = 2{\cdot}5\,^{\circ}\mathrm{C}$ should be adequate to allow for the fan temperature gain in all practical situations. The required initial estimate for the supply-air condition is thus the point on the locus for which Δt, as defined by equations (6.3.6-6.3.8), is zero. In other words, we seek to find the uniquely defined point for which

$$\Delta t = f(\tilde{t}_{SA}) = 0, \quad \tilde{t}''_{SA} = \tilde{t}_{SA} - \delta t \quad . \tag{9.6.4}$$

This point is readily found by using procedure zero in conjunction with a function to evaluate $f(\tilde{t}_{SA})$ for a given trial point $\tilde{t}_{SA}$. To find an upper bound for the bracketing interval $[a..b]$, the condition of the air in each zone is fixed at a point determined by its thermostat set point, and an arbitrarily chosen relative humidity (60% say). Let $t_{i,i}$ be the temperature of the air entering zone i corresponding to this zone condition. Then, an appropriate upper limit for the bracketing interval is

$$b = \min_{i=1}^{n} t_{i,i} \quad . \tag{9.6.5}$$

A lower limit to the bracketing interval may then be found by setting $a = b$, and decrementing a in steps of $1\,^{\circ}\mathrm{C}$ until (9.6.2) is satisfied.

With a consistent set of zone conditions established, the mass flow, temperature and humididty distributions through the return-air duct system are readily found (section 9.5.2). The supply and return-air temperature gains thus calculated are stored on a zone-by-zone basis in the associative array `DController::ZInfo` (see section 6.3.2). These will be subject to a small degree of fluctuation as the moisture staircase iteration proceeds, but for all practical purposes may be regarded as constant, thus avoiding the need to repeat the duct simulation at each step of the moisture staircase iteration.

The temperature gain through the outdoor-air duct may also be evaluated using the methods developed, if it is significant. This is stored in member variable `OA_Duct::dT`, which defaults to 0.

9.6.2. Outdoor Air Units.

If the fan and duct temperature gains are specified *a priori*, a solution for the thermal performance of the unit is readily found as described in section 6.4.2. If these are to be determined, an iterative solution procedure becomes necessary. Efficient algorithms for solving the problem in this latter case are described in section 10.3.2.

---

[121] It is in fact the actual *volume* flow rate of air which remains constant in a CAV system. See chapter 10 for a discussion of this point.

Chapter 10. Fans and Fan-System Interaction.

10.1. Introduction.

The previous chapter described a set of algorithms for calculating the heat gain and pressure loss associated with air flowing through a duct system. The pressure deficit incurred as a result of losses in the duct system and other components, including in particular the coils and filters, must be balanced by a total pressure gain across the fan. The shaft power absorbed by the fan in working against the pressure loss,

$$P_f = \frac{Q_f \Delta p_t}{\eta_f} \tag{10.1.1}$$

where,

Q_f = Actual volume air flow through the fan,
Δp_t = Total pressure rise across the fan,
η_f = Fan efficiency (total),

while the electrical power input required by the fan motor,

$$P_e = \frac{P_f}{\eta_m} \tag{10.1.2}$$

where η_m = Motor efficiency.

A fraction of the power is dissipated within the air stream, causing a rise in temperature across the fan. That is,

$$\dot{m} C_p \Delta t = \dot{m} \Delta h = \chi P_e \tag{10.1.3}$$

where Δt is the temperature rise across the fan, and Δh is the increase in enthalpy. $\chi = 1$ if the motor is located within the air stream, and $\chi = \eta_m$ if it is located external to the air stream. Thus, while the electrical power consumed by the fan (equation 10.1.2) is a direct debit to the energy budget for the system, the fan also contributes indirectly to the energy budget in that the fan energy dissipated as heat increases the refrigeration load on the cooling coil.

The fan total pressure rise comprises the following components:

$$\Delta p_t = \Delta p_{ts} + FSE_i + FSE_o \tag{10.1.4}$$

where,

Δp_{ts} = System total pressure rise,
FSE_i = Fan system effect pressure loss caused by fan inlet conditions,
FSE_o = Fan system effect pressure loss caused by fan outlet conditions.

The stack effect, which may contribute significantly to the pressure rise across the fan under certain circumstances (ASHRAE, 1989; chapter 32) has been ignored in the above. Once the terms in the above equation have been evaluated, the fan operating point can be

found from the fan characteristic, which will have been determined by the manufacturer on the basis of experimental measurements. The operating point also determines the fan efficiency (η_f).

From the point of view of the present study, which is concerned with the thermal performance of air conditioning systems, the operating characteristics of a fan are mainly of interest in determining the rate at which fan energy is dissipated in the airstream. The data structures described in the following section have been designed with this end in mind. This work should be regarded as being of a preliminary nature only. The longer term aim is to use the classes developed as the basis upon which a database can be constructed which will serve as a repository for a wide range of data, including physical dimensions and acoustical data. The projected database will provide a tool for fan selection, possibly used in conjunction with an appropriate expert system. Additional areas in which the methodology is currently deficient, or may be extended, will be indicated in the text.

10.2. Simulating Fan Performance.

10.2.1. Fan Performance Characteristics.

From the point of view of the present study, the parameters which are of major interest in describing fan performance are those occurring in equation (10.1.1), namely Q_f, Δp_t and η_f, together with the rotational speed (N) of the fan. Graphically, fan performance curves can be presented by plotting contours of fan speed and efficiency versus flow rate and pressure, as shown in figure 10.1, which shows the characteristics for a Richardson type CYD 300 centrifugal fan with backward-curved aerofoil blades. Figure 10.1 has been constructed using tabular data available from the manufacturer (Richardson Pacific Ltd., 1989). The presentation used by Richardson Pacific Ltd. is fairly typical, and comprises fan speed and power tabulated as a function of volume flow rate and *static* pressure rise across the fan. As is customary, the experimental data upon which the tabulation is based have been reduced to the density (1·204 kg/m³) of dry air at ASHRAE standard conditions. The steps in deriving a graphical presentation such as that of figure 10.1 from tabulated data are as follows:

i. Static pressure is converted to total pressure as

$$\Delta p_t = \Delta p_s + \frac{1}{2}\rho v^2 \qquad \textbf{(10.2.1)}$$

where v is the velocity of the air leaving the fan.

ii. Efficiency is calculated using equation (10.1.1).

iii. The data points are interpolated to a rectangular grid using a combination of Laplacian and spline interpolation, and the contours drawn using a proprietary plotting package (Young and Van Woert, 1989).

The fine structure in the efficiency contours of figure 10.1 is undoubtedly an artefact caused by the low relative precision of the tabulated fan power ratings.

The gridded data used to produce figure 10.1 comprises two arrays, each of 51 × 51

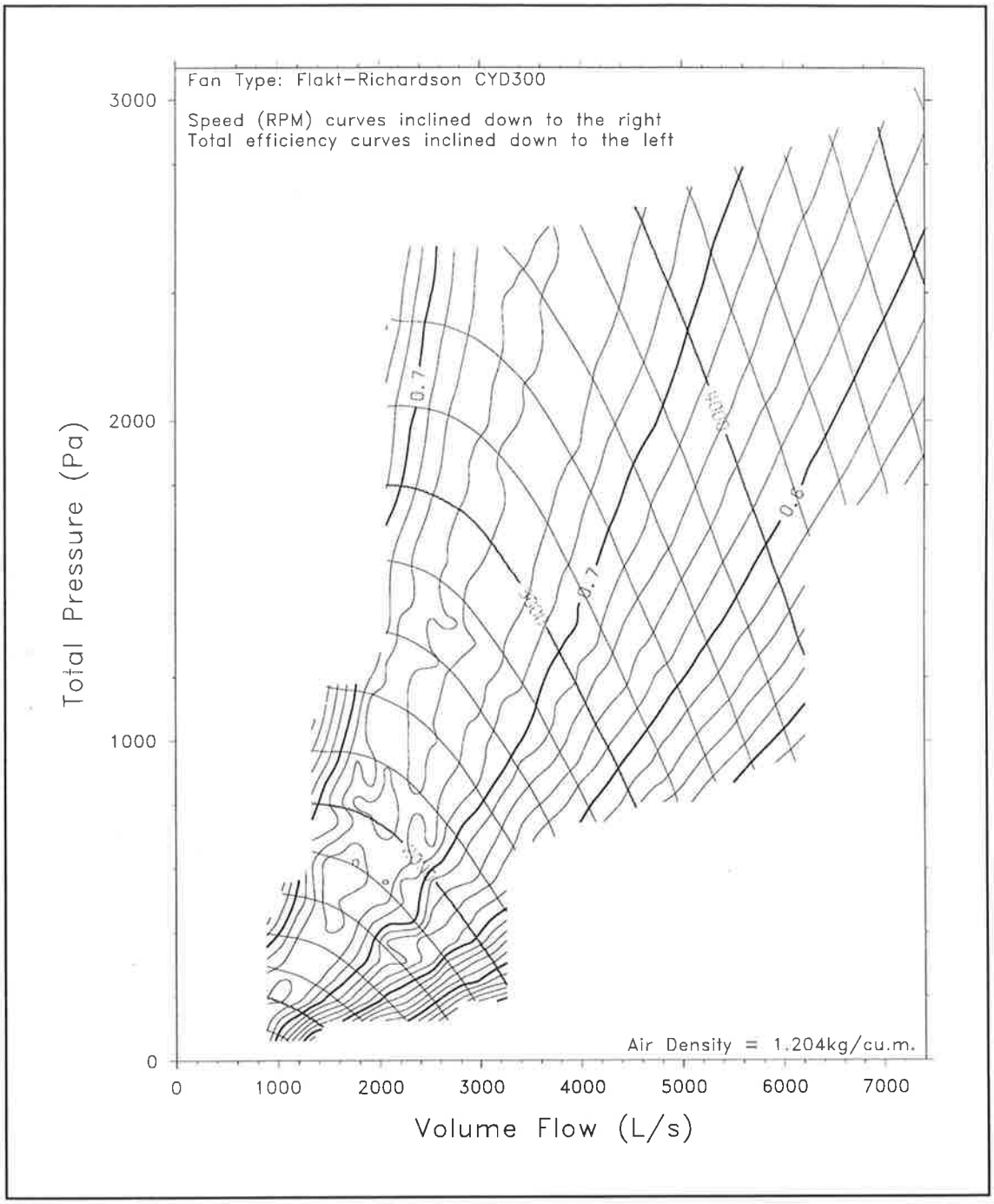


Figure 10.1. Characteristics for a centrifugal fan with backward-curved blades (Richardson type CYD 300).

elements, and while a representation of this form *could* be used as the basis for storing and manipulating fan performance data, a more compact representation is desirable. A suitable

representation may be obtained by recourse to the fan laws (Osborne, 1977; ASHRAE, 1988). In the present work the fan laws have not been used directly, but rather the data have been expressed in terms of a set of dimensionless coefficients based on the fan laws (Osborne, 1977). The relevant coefficients are, the *volume coefficent*,

$$\phi = \frac{Q_f}{u\left[\frac{\pi d^2}{4}\right]} \tag{10.2.2}$$

the *total pressure coefficient*,

$$\psi = \frac{\Delta p_t}{\frac{1}{2}\rho u^2} \tag{10.2.3}$$

and the *power coefficient*,

$$\lambda = \frac{\phi\psi}{\eta_f} \tag{10.2.4}$$

in which d is the *impeller diameter*, and u is the *impeller peripheral velocity*,

$$u = \pi d N \quad . \tag{10.2.5}$$

Expressed in terms of these coefficients, it is found that the data from which figure 10.1 has been constructed can be collapsed onto two curves, representing total pressure coefficient and power coefficient as a function of volume coefficient, as shown in figure 10.2. Use of the above coefficients accounts for variations in the density of the air passing through the fan, and also permits a common set of curves to be used for a geometrically similar range of fans, although some caution is urged in scaling in this manner for critical applications, since the standard of finish, and consequently the performance of fans within a range, will typically improve with increasing size (Osborne, 1977).

For use in computational applications, it is necessary to fit a function to the data points of figure 10.2. In view of the noise which has already been remarked upon in connection with the efficiency data in particular, it is desirable that the function fitted should perform some smoothing on the data. The data of figure 10.2 have been fitted in the least squares sense by a cubic spline, which is also shown in figure 10.2. The properties of the cubic spline representation used will be described in the following section.

10.2.2. Least Squares Cubic Spline.

The considerations involved in fitting a cubic spline to a randomly-distributed set of data points in a plane have been described in some detail by Hayes and Halliday (1974), and the techniques developed in that work form the basis for the one-dimensional spline fitting routines used in the current work. The following will briefly describe the properties of the cubic spline representation chosen, and the fitting procedure. More complete details will be found in the references cited in this connection.

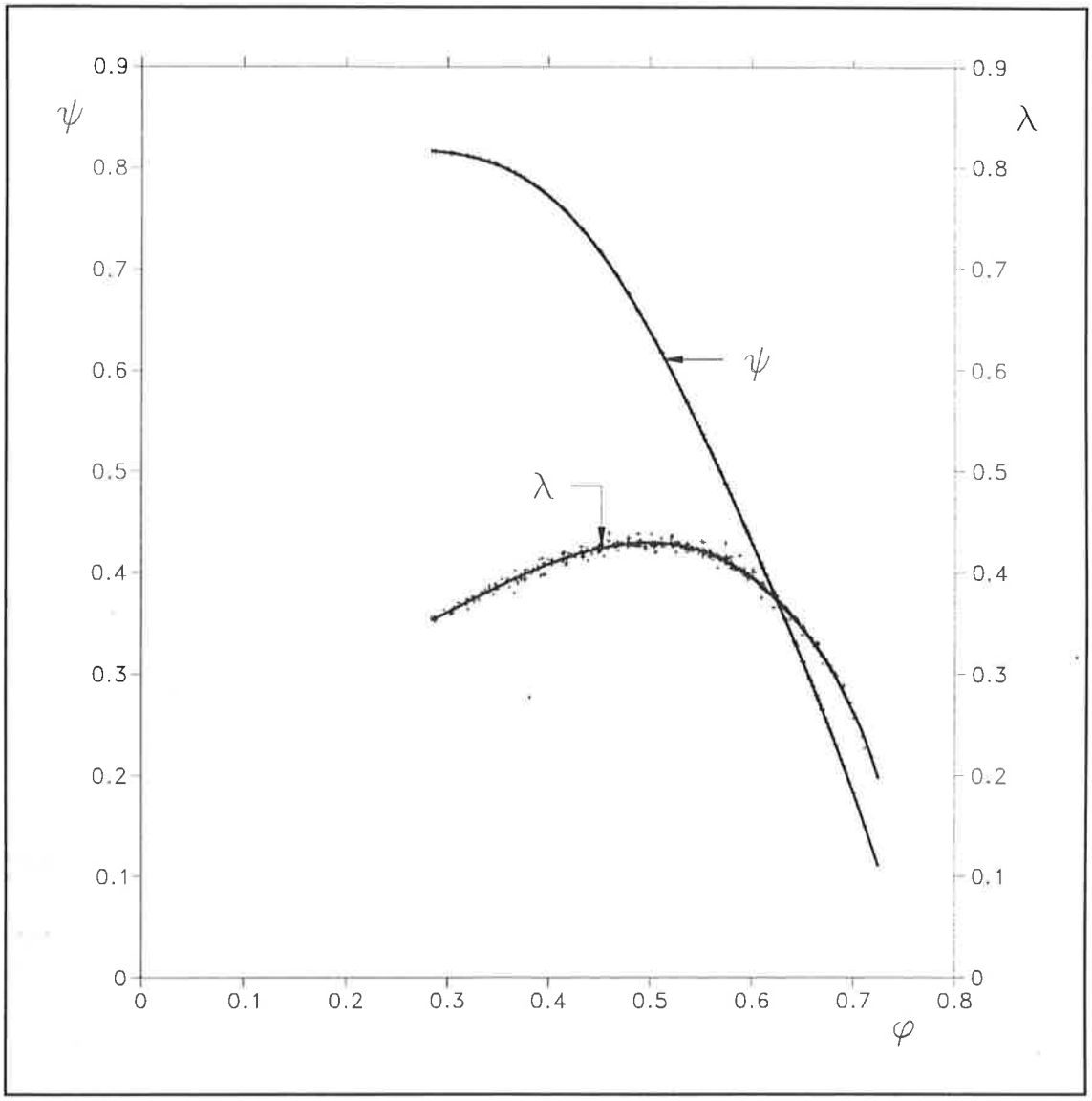


Figure 10.2. Fan characteristics of figure 10.1 expressed in terms of dimensionless coefficients.

Consider an interval $[a, b]$ on the real axis, and a set of knots defined within the interval such that $a \leq \lambda_1 \leq \ldots \leq \lambda_h \leq b$. Then, a cubic spline $s(x)$ can be constructed within the interval $a \leq x \leq b$, such that:

i. In each of the intervals, $x \leq \lambda_1$; $\lambda_{j-1} \leq x \leq \lambda_j$, $s(x)$ is a polynomial of degree 3.

ii. The first and second derivatives of $s(x)$ are continuous at the interior knot points (λ_j, $j = 1, \ldots h$). In general, the third derivative will be discontinuous at these points.

In approximating a set of data points (x_r, f_r), $r = 1, 2, \ldots, n$ by a cubic spline defined over a set of knots, we seek to find the spline coefficients which will minimize the sum of squares of the residuals,

$$SSR = \sum_{r=1}^{n} [f_r - s(x_r)]^2 \quad . \tag{10.2.6}$$

A cubic spline may be expressed in terms of its polynomial coefficients. Computational schemes based on this representation however tend to be numerically unstable. It is computationally advantageous to express the cubic splines in terms of *B-splines* or fundamental splines (Hayes and Halliday, 1974). A cubic B-spline $M_i(x)$ is defined to be a spline with knots λ_{i-4}, λ_{i-3}, λ_{i-2}, λ_{i-1}, λ_i, which is zero everywhere except in the range $\lambda_{i-4} < x < \lambda_i$. In order to construct the set of B-splines necessary to calculate $s(x)$ at any point within $[a, b]$, it is necessary to define eight knot points in addition to those interior to the interval:

$$\lambda_{-3} \le \lambda_{-2} \le \lambda_{-1} \le \lambda_0 \le a \tag{10.2.7a}$$

and

$$b \le \lambda_{h+1} \le \lambda_{h+2} \le \lambda_{h+3} \le \lambda_{h+4} \tag{10.2.7b}$$

where, while the additional knot points can be arbitrarily specified subject to the above restrictions, it is usually convenient to select them such that $\lambda_{-3} = \lambda_{-2} = \lambda_{-1} = \lambda_0 = a$, and $b = \lambda_{h+1} = \lambda_{h+2} = \lambda_{h+3} = \lambda_{h+4}$. The cubic spline may thus be evaluated for some point $a \le x \le b$ as

$$s(x) = \sum_{i=1}^{h+4} \gamma_i M_i(x) \tag{10.2.8}$$

where it is understood that for some $\lambda_{u-1} \le x \le \lambda_u$, the only non-zero B-splines, $M_i(x)$ are those for which $u \le i \le u+3$. The non-zero B-splines at a point may be evaluated in a stable and efficient manner using a recurrence relationship due to Cox (1972).

The problem of fitting a cubic spline to a set of data becomes one of finding a least squares solution for the coefficients γ_i in the set of observation equations

$$\sum_{i=1}^{h+4} \gamma_i M_i(x_r) = f_r \ , \qquad i = 1, 2, \ldots, n \tag{10.2.9}$$

which may be written in matrix form as

$$\boldsymbol{A}\gamma = \boldsymbol{f} \tag{10.2.10}$$

where **A** is an $n \times (h + 4)$ matrix, for which element $A_{ir} = M_i(x_r)$, γ is a vector of length $h + 4$ containing the spline coefficients, and **f** is an r-vector containing the data values.

The overdetermined (for $n > h + 4$) system (10.2.10) may be solved by:

i. Reducing matrix **A** to upper triangular form. This is done by processing the observations on a row-by-row basis using standard Givens rotations, as described by Cox (1981). This simultaneously reduces vector **f** to a vector in which the non-zero elements are contained within the first $h + 4$ elements. Note that each row of **A** has only four adjacent non-zero elements, and consequently transforms to a band

matrix of bandwidth 4. Cox (1981) shows how to take advantage of this structure to reduce the computational effort.

ii. The transformed system is readily solved by back-substitution, having first accounted for rank deficiency, if any. This should not occur in the present application.

A class `CubicSpline` has been implemented to handle a range of operations associated with cubic splines. Such a class will be required to fulfil two basic types of function:

i. Calculating the spline coefficients to produce the least squares fit to a set of data.

ii. Evaluating the cubic spline at a point, using a set of cubic spline coefficients which have either been calculated or otherwise supplied.

A partial listing of the public interface for this class follows:

```
class CubicSpline
{
  :
public :
  CubicSpline (const float xl,
               const float xu,
               const int h,
               const int q);
  CubicSpline (const int h,
               const int q,
               float* lambda,
               float** gamma = 0);
  ~CubicSpline ();
  float Evaluate (const float x,
                  const int iq = 0);
  int Update (const float x,
              float* z);
  int Solve ();
  :
};
```

Note:

i. The class supplies two constructors. The first takes as arguments the upper and lower bounds of the interval over which the spline is defined, together with the number of internal knots. Argument `q` specifies the number of data sets (f) for which a cubic spline fit is sought in the interval; an arbitrary positive number may be specified. The internal knot points set up by this constructor are evenly distributed within the interval. If the second form of the constructor is used, the knot points must be specified by argument `lambda`, and a set of spline coefficients may optionally be specified by argument `gamma`. These latter may have been evaluated by a previous invocation of the class.

ii. Member function `Evaluate (x, iq)` calculates the value of $s(x)$ at point x, where the second argument specifies the data set for which a solution is sought ($1 \le iq \le q$; defaults to 1).

iii. Member functions `Update` and `Solve` are concerned with determining a least squares fit to one or more data sets. The overdetermined linear system is actually solved by invoking an instance of class `LinearLeastSquaresBanded`, which is derived from class `LinearLeastSquares`. The classes implement linear system solvers using the methodology of Cox (1981). The base class provides a set of virtual member functions to solve for a system with a full observation matrix, while the derived class overrides the virtual functions to take advantage of the increased computational efficiency possible when the observation matrix is banded. The steps in obtaining a solution are:

a. The arguments to member function `Update (x, z)` specify the data set members (`z`) corresponding to a given point (`x`). If an invocation of class `LinearLeastSquaresBanded` is not available, one is created. One row of the observation matrix is set up, and function

```
void LinearLeastSquaresBanded::Update (float* x, float* z);
```

called to process it.

b. When all data points have been processed, member function `Solve ()` may be called to solve for the spline coefficients. This invokes function

```
int LinearLeastSquaresBanded::BackSubstitution ();
```

to solve the transformed upper triangular matrix by back substitution, unpacks the results matrix to obtain the spline coefficients, and deletes the current invocation of the linear least squares solver.

The quality of the cubic spline fit may depend critically upon the selection of the interior knot points. The curves of figure 10.2 were obtained using five interior knot points selected using an algorithm due to Dierckx (1977, 1981), which attempts to determine a near-optimal knot point placement to fit a particular distribution of data points. The fit so produced is greatly superior to earlier efforts using evenly spaced knot points, in which the fitted curves showed marked deviations from the data points, particularly near the ends of the curves where the data points are less densely distributed.

10.2.3. Fan Control.

The operating point of a fan is determined by the point at which the fan characteristic and the system characteristic intersect. Ignoring velocity dependence of the loss coefficients for certain duct fittings, the system characteristic may be approximated by equation 9.4.10. A wide range of devices are available to match the fan characteristics to the system characteristics in such a manner that the system flow requirements will be met. For a CAV

system the characteristics may be matched fairly simply during commissioning by finding the damper settings which will balance the system, and locking them in position. In the case of a VAV system some means of dynamically modulating the air flow must be provided. Eck (1973) provides a wide-ranging review of control strategies. The methods which are commonly used in air conditioning applications fall into three broad categories:

i. Control by altering the system characteristic. This is undoubtedly the simplest method of controlling the fan, and probably the least efficient. Using this strategy, the fan is operated at constant speed, and the conductance of the system altered until the desired flow rate is achieved. Two variants of this method are in common use.

   In the first the fan operating point rides up the fan characteristic at constant speed in response to increasing system pressure induced by closing the VAV boxes. This technique is known as *riding the fan curve*. Modulating the air flow in this manner can result in a considerable increase in static pressure throughout the entire duct system at part load, leading possibly to over-pressurization of the VAV boxes, which will result in an increase in air flow, and in noise generation. The problem can be partially alleviated by using a two- or three-speed drive on the fan, which will also result in energy savings, at least in the case of the more efficient backward-curved aerofoil bladed fans.

   Alternatively, the system characteristic may be altered by the use of a *discharge damper*, thus avoiding the increase in static pressure throughout the entire system which is associated with the former method. The use of a two- or three-speed fan may be advantageous from the point of view of energy conservation in this case also.

ii. Altering the fan characteristic by the use of *variable inlet guide vanes*. This technique is used at part load to impart a rotation to the entering air in the direction of rotation of the fan impeller, which results in a decrease in static pressure and power consumption for a given flow rate. Eck (1973) describes a wide range of such devices for both axial and centrifugal fans. In air conditioning applications this technique is most commonly used in conjunction with backward-curved aerofoil bladed fans.

iii. Use of a *variable-speed drive* on the fan is probably the most efficient means of modulating air flow (Eck, 1973), and in general should be specified wherever the budget permits. Speed control until very recently has mostly been achieved by the use of a hydraulic or magnetic clutch or similar device, rather than by direct control of the motor speed. Such devices do have transmission losses associated with them. In the present work, because of the rapid growth of the use of variable frequency control systems in conjunction with high efficiency motors, losses have been assumded to be negligible.

Algorithms are here developed for just two of the possible range of fan control strategies; control by variable-speed drive, and control by the use of a discharge damper in

conjunction with a single-speed motor[122]. Let the fan characteristic and power curves by represented by the functions

$$\psi = s_1(\phi) \tag{10.2.11a}$$

and

$$\lambda = s_2(\phi) \tag{10.2.11b}$$

where in the present work $s_1(\phi)$ specifies the cubic spline fit, as described in section 10.2.2.

For a variable-speed drive, Δp_t and Q_f are specified, and we seek to find u and hence N. To do this we combine equations (10.2.2) and (10.2.3) and eliminate u to give

$$\frac{8\rho Q_f^2}{\pi^2 d^4 \phi^2} s_1(\phi) - \Delta p_t = 0 \tag{10.2.12}$$

which can readily be solved for ϕ using a standard root-finding routine such as zero. A solution for the fan speed follows, while the power coefficient may be found using equation (10.2.11b). In the case where $s_1'(\phi) \leq 0$ for all points within the interval $[a, b]$ over which the spline fit is defined, the interval $[a, b]$ may be used as the bracketing interval for solving equation (10.2.12)[123]. This is the only situation with which we be concerned in the present study, but it is worth considering the modifications to the procedure which must be made if the peak of the characteristic is contained within the interval $[a, b]$. In this case two, and for fans with forward-curved blades and various other fan types, three points which satisfy equation (10.2.12) may lie within the interval $[a, b]$ if the required value of ψ is sufficiently close to peak. We are normally only interested in solution points to the right of the peak, and it is prudent, if not always necessary, to restrict the search to that region. This refines the location of the peak. The derivatives of a cubic spline may readily be found from its B-spline basis (Butterfield, 1976; de Boor, 1977). A procedure for locating the peak of a characteristic might thus be structured as follows:

i. Procedure zero is used to find a point ϕ_1 within the interval $[a, b]$ for which $s_1'(\phi_1) = 0$. If $s_1''(\phi_1) < 0$, ϕ_1 is the required peak.

ii. If $s_1''(\phi_1) > 0$, ϕ_1 will be a local minimum. Now we know that if the characteristic has a local minimum, the peak must lie to the right of the local minimum. Thus, we may repeat the search of step (i) over the interval $[\phi_1 + \epsilon, b]$, where ϵ is some small increment in ϕ. The value ϕ_2 within this interval for which $s_1'(\phi_2) = 0$ will be the required peak.

---

[122] It is intended at a later stage to develop algorithms suitable to model the case of fan control using inlet guide vanes.

[123] If no solution lies within the interval $[a, b]$, it probably means that the fan has been incorrectly specified for the application.

Where control is to be achieved by means of a discharge damper, Q_f and N are specified, thus fixing the operating point of the fan, and we seek to find the total pressure rise across the fan which corresponds to this operating point. Thus, ϕ is found from equation (10.2.2), and ψ is found from equation (10.2.11a). The appropriate value for Δp_t thus follows from the definition of ψ (10.2.3). The discharge damper setting which produces the required system pressure must then be found. This will be taken up again in section 10.3

10.2.4. Motor Efficiency.

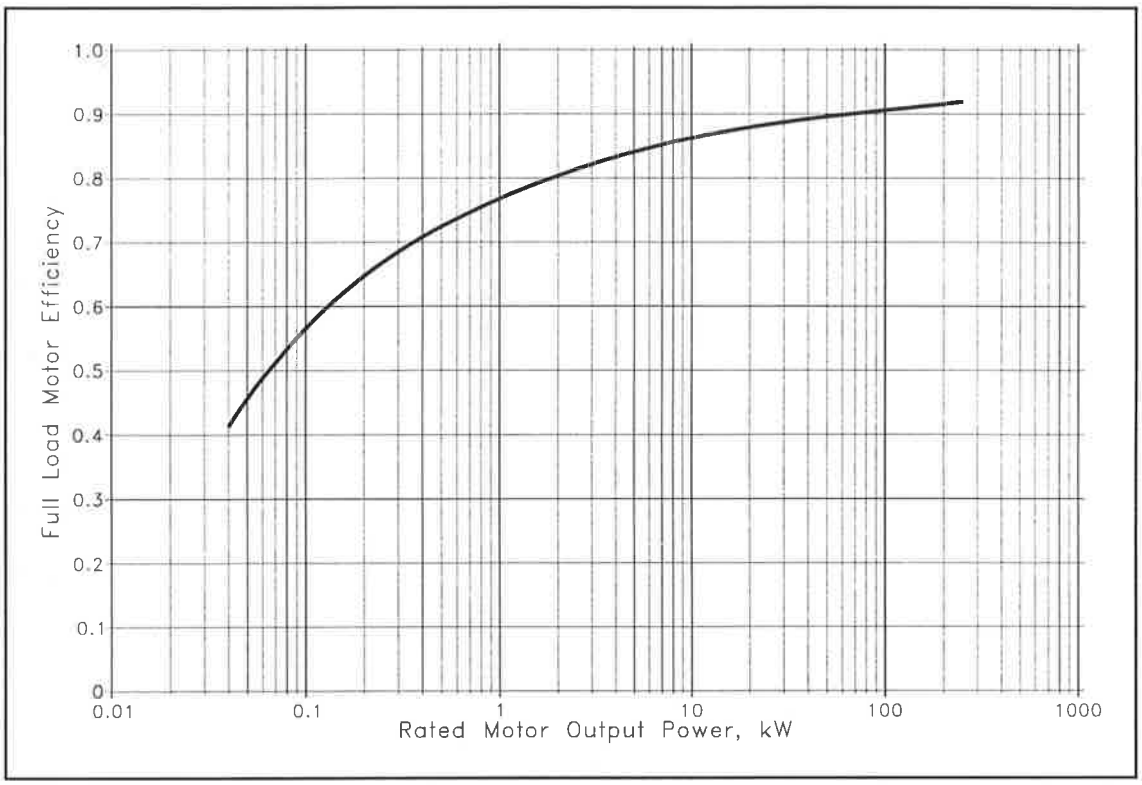


Figure 10.3. Full load motor efficiency as a function of output power for a typical range of electric motors.

Australian Construction Services (1988a) have tabulated full load motor efficiency as a function of rated motor output power for a typical range of squirrel cage induction open type electric motors, ranging in output power from 0·04 to 25 kW. This data may be fitted using least-squares cubic splines, as shown in figure 10.3, and is used for estimating electric power consumption and heating of the air stream in the present work. Efficiency will in fact vary somewhat according to the actual load on the motor, and according to the motor speed (ASHRAE, 1989; Chapter 26). For more critical work, it is desirable that this data be supplemented by manufacturer's data for the actual motor used in a specific application.

10.2.5. Fan System Effect.

It is customary to test a fan with open inlet or with a length of straight duct attached to the inlet, and with a length of straight duct attached to the outlet. The straight duct should be of sufficient length to permit full static pressure recovery to occur. If a fan is operated in a configuration which differs from the test configuration, the rated performance of the fan may not be achieved. The *fan system effect*, as this departure from rated performance is termed, may be expressed as an additional system pressure loss (equation 10.1.4), and calculated as

$$FSE = C_0 \frac{\rho V_0^2}{2 g_c} \qquad (10.2.13)$$

where
C_0 is the fan system effect loss coefficient, and
V_0 is the fan inlet or outlet velocity, as appropriate.
For centrifugal fans, V_0 is based on the area at the inlet collar in the case of factors which affect performance at the fan inlet, and on the fan outlet velocity, based on the fan outlet area in the case of factors which affect the performance at the fan outlet. For axial fans, V_0 is based on the area calculated from the fan diameter.

The loss coefficients for a range of fan system effect factors are tabulated in the appendix to chapter 32 of the 1989 edition of ASHRAE Fundamentals. Computationally, these are handled in a manner which is entirely analogous to that used for duct fittings (section 9.4.2). Specifically, a class ***FanSystemEffect*** is derived from class ***Fitting***. Its constructor is defined as

```
FanSystemEffect::FanSystemEffect (Fan* f,
                                  unsigned ind,
                                  int IP1 = 0,
                                  double P1 = 0,
                                  double P2 = 0);
```

where `f` is a pointer to the fan, `ind` is the index of fitting within group 7, and `IP1`, `P1` and `P2` are optional parameters which default to zero. Member function `FanSystemEffect::C ()` may then be invoked to calculate the fan system effect loss coefficient appropriate to the current fan operating conditions.

Class ***FanSystemEffect*** is in fact an abstract class. Objects of this class must be created as belonging to one of the derived classes ***FanSystemEffectIn*** or ***FanSystemEffectOut***. These latter classes provide a single member function; a constructor which contains code to check whether the fan system effect specified by argument `ind` is relevant to fan inlet or outlet conditions, as the case may be.

10.2.6. Classes for Simulating Fan Subsysytems.

Fan subsystems are modelled using a coordinating set of classes. The major classes involved are shown in figure 10.4. Classes *FanSystemEffect* and *CubicSpline* have been described in the foregoing section. A brief description of the remaining classes follows.

10.2.6.1. Class *Motor*.

This is a fairly rudimentary class which performs the major function, within the context of the present application, of estimating the motor efficiency using the cubic spline fit of figure 10.3. The constructor,

```
Motor::Motor (const double r = 0);
```

accepts the motor power rating as its sole argument. Motor efficiency is calculated by spline interpolation using member function

```
double Motor::Efficiency ();
```

If no rating is specified (`r` = 0), or if the specified value is out of range of figure 10.3, a default efficiency of 0·89 is used.

10.2.6.2. Class *Fan* and Derived Classes.

Class *Fan* is central to the fan simulation process. This class provides a **protected** constructor

```
Fan::Fan (TextSource& t,
          fantype ft = Centrifugal);
```

which permits the fan specifications to be read from an object of class *TextSource* (section 2.4). The fan characteristic and efficiency curve are stored in an object of class *CubicSpline*, which is accessed by member variable `Fan::S`. A pointer to an object of class *AirState* establishes the properties of the air entering the fan.

From the point of view of the user, the member functions of major interest, in addition to the constructor, are the two functions implemented to find the fan operating point using the algorithms developed in section 10.2.3. For a fan using a variable speed control, the operating point may be found by invoking member function

```
double Fan::Speed (const double m,
                   const double p);
```

The fan speed corresponding to specified mass flow rate and total pressure rise is calculated and returned. Member function

```
double Fan::Pressure (const double m,
                      const double N);
```

calculates and returns the pressure rise dictated by a specified combination of mass flow rate and fan speed. These functions also calculate the total fan efficiency corresponding to the operating point, which may be used to find the shaft power consumption (equation 10.1.1).

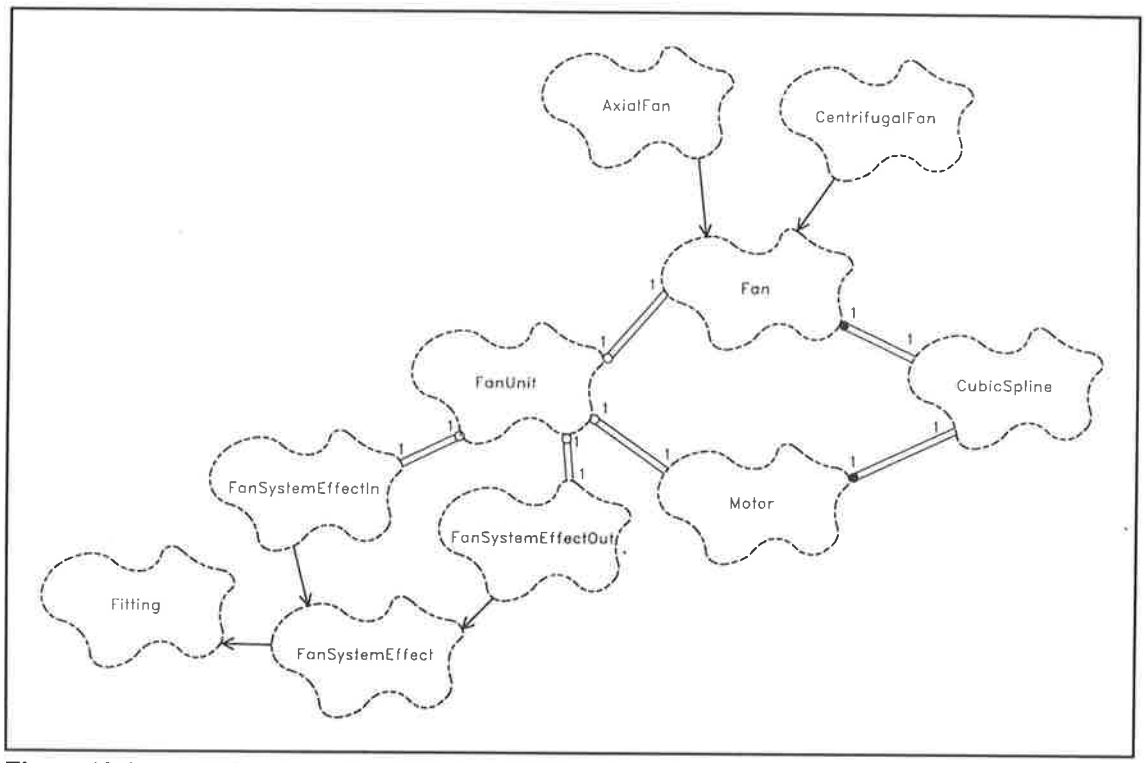


Figure 10.4. Major classes involved in modelling a fan subsystem.

Class *Fan* serves as a base class for derived classes *CentrifugalFan* and *AxialFan* which implement the constructors

```
CentrifugalFan::CentrifugalFan (TextSource& t);
```

and

```
AxialFan::AxialFan (TextSource& t);
```

respectively. The main features contributed by the derived classes are that they define geometrical information specific to the particular fan type. This additional data is required to calculate the system effect factors.

At a future date it is desirable that the hierarchy of classes which has been established about class *Fan* should be extended, both in the direction of increased specialization, and in the direction of increased generalization. In particular, it is desirable that class *Fan* should comprise a subclass of a generic class representing rotodynamic machines, which would also have a class representing pumps as a subclass.

10.2.6.3. Class *FanUnit.*

For the most part the system simulation software interacts with the fan subsystem by means of calls to the member functions of class *FanUnit.* This class maintains a pointer to an object of class *Fan* (`FanUnit::f`), and one of class *Motor* (`FanUnit::m`)[124]. The class defines two local enumerated types,

```
enum motorposition { in, out };
enum controltype { speed, pressure };
```

the first of which flags whether the drive motor is in or out of the air stream, while the second indicates whether the flow through the fan is to be controlled by varying fan speed (use of a variable speed drive), or by varying the pressure rise required across the fan (use of a discharge damper or riding the fan curve)[125]. Other member variables include two pointers to objects of class *FanSystemEffect* (`FanUnit::fse_in` and `FanUnit::fse_out`), permitting loss coefficients to be calculated for fan system effects at the fan inlet and outlet, and a pointer to an external object of class *AirState*, which establishes the inlet condition to the subsystem.

The major member functions are:

i. `FanUnit::FanUnit (TextSource& t);`

The constructor for the class extracts the specifications for the various components from an object of class *TextSource* (section 2.4). Note that the user may elect to omit the specifications, in which case the specifications for the motor and fan system effect factors must also be omitted, and member variables `f`, `m`, `fse_in` and `fse_out` will remain null pointers. If a fan is specified, a motor must also be specified, but the specifications for the fan system effect factors may be omitted. If no fan is specified, the user may elect to specify a fan temperature rise; this will default to zero.

ii. `FanUnit::~FanUnit ();`

The destructor for the class invokes the destructors for the components of the class, described above, if they have been specified.

iii.
```
void FanUnit::Control (const double d1,
                       const double d2);
```

performs a control action on the fan. The first argument specifies the required mass flow rate of air through the fan. If the fan is controlled by a variable speed drive,

---

[124] In the longer term it is intended that this class should be generalized to handle the situation where fans are operated in parallel or in series.

[125] By default, it is assumed that the fan is located within the air stream, and that a variable speed drive is employed.

```
Input :  Fan control variable - speed
                 ṁ        : Mass flow rate through fan
                 Δp_t     : Total pressue rise required
         Fan control variable - pressure
                 ṁ        : Mass flow rate through fan
{

        if fan characteristics are specified then
        {
                if Inlet fan system effect is specified then
                        Calculate FSE_i;
                else
                        FSE_i ⇐ 0;
                if Outlet fan system effect is specified then
                        Calculate FSE_o;
                else
                        FSE_o ⇐ 0;
                if Fan control variable = speed then
                {
                        ΔP_f ⇐ Δp_t + FSE_i + FSE_o;      (ΔP_f is pressure rise across fan)
                        Find operating point for ṁ, ΔP_f;   (Call Fan::Speed (m, p))
                }
                else
                {
                        N ⇐ Design fan speed;
                        Find operating point for ṁ, N;
                                                  (Call Fan::PressureRise (m, n))
                        Δp_t ⇐ ΔP_f - FSE_i - FSE_o;
                }
                η_f ⇐ Efficiency for fan operating point;
        }
        else
                η_f ⇐ Default efficiency;          (75% unless otherwise specified)
        η_m ⇐ Fan motor efficiency;
        P_f ⇐ Shaft power;                                  (Equation 10.1.1)
        P_e ⇐ Electrical power;                             (Equation 10.1.2)
        Δt ⇐ Temperature rise;                              (Equation 10.1.3)

}
```

Figure 10.5. Logic for a function to determine the operating point for a fan.

the second argument specifies the total pressure rise which must be supplied by the fan. Where fan control is effected by modulating the system pressure drop, the second argument is omitted. In this latter case, the fan operating point is determined by the flow rate of air through the fan, and by the fan design speed. Additional pressure losses resulting from fan-system interaction are handled internally. The function also handles the case where the fan characteristics have not been specified. In this situation a default efficiency (0·75 unless otherwise specified) is used to calculate fan power consumption and temperature rise. The logic for this function is shown in figure 10.5.

iv. `double FanUnit::TemperatureRise ();`

calculates the temperature rise across the fan, if a fan has been specified, according to equation (10.1.3). If no fan has been specified, an optional temperature rise will be returned (see (i) above).

v. `double FanUnit::FanSystemEffectIn ();`
`double FanUnit::FanSystemEffectOut ();`

These functions calculate and return the pressure losses for the system effect factors at inlet and outlet, if they have been specified. By default, a value of 0 is returned.

vi. `double FanUnit::Power ();`

calculates and returns the electric power consumed by a fan motor (equation 10.1.2).

10.3. Fan-System Interaction.

A fan operates between two pressure levels, P_1 and P_2 where $P_2 > P_1$, to deliver air at a flow rate Q_s. Pressure level P_2 is determined by the pressure losses within the supply-air path, which comprises not only the pressure loss through the duct system, but also losses incurred by flow through the coils and filters[126]. Similarly, the fan inlet pressure P_1 will be determined by losses incurred by flow through either the return-air or the outdoor-air path.

Where several duct systems are fed from a common source, or discharge into a common sink, the pressure drop through each such duct system must be balanced, so that the pressure at the root of all such duct systems will be the same[127]. Consider thus the situation where n supply-air duct systems are fed by a fan, and let P_i' be the pressure at the root of the i'th duct system. Further, define $\wp_i$ such that

$$\wp_i = P_i' + \Delta P_{i,coil} + \Delta P_{i,filter} \qquad (10.3.1)$$

where either the coil or the filter, or both, may be omitted from the above equation. Then

$$P_2 = \max_{i=1}^{n} \wp_i \qquad (10.3.2)$$

---

[126] Other losses may be incurred, due for instance to factors involved in the casing design, but only losses incurred in the duct and duct fittings, and in the coils and filters are considered in the present study. It is notionally simple to add an ahu casing loss to the system.

[127] It will be recalled from the discussion of chapter 9 that the algorithms developed therein calculates the pressure at the root of a tree network *referenced to atmospheric*, which serves as a common datum.

and let i_{max} be the duct system for which $\wp_i = P_2$. Then, the damper at the root node of each duct system, $i \neq i_{max}$, must be closed progressively until $\wp_i = P_2$. The manner in which this is done has been described in section 9.4.4.

Consider now the case of a system having a coil and a filter in both the return-air and the outdoor-air paths (configuration (g)[128]), and define

$$\wp_R = P_R' - \Delta P_{R,coil} - \Delta P_{R,filter} \tag{10.3.3}$$

for the return-air path, and

$$\wp_O = P_O' - \Delta P_{O,coil} - \Delta P_{O,filter} \quad . \tag{10.3.4}$$

Then,

$$P_1 = \min [\wp_R, \wp_O] \tag{10.3.5}$$

and the damper for the duct system having the higher exit pressure is closed until $\wp_R = \wp_O$. This procedure is subject to some variation, depending upon the exact configuration of the return-air and outdoor-air paths. Thus:

i. For a terminal unit fed from a central outdoor-air plant (configurations (b), (d) and (e)), the *casing pressure* $P_c = \wp_R$, and this is stored as member variable `D_AHU::p`. The control action required to balance the outdoor-air and return-air flows is taken by the supply-air duct system for the central outdoor-air unit (see section 10.4).

ii. For a unit in which the outdoor-air and return-air streams are combined before being treated through a common coil (configurations (e) and (f)), $\wp_R = P_R'$ and $\wp_O = P_O'$, and the air streams must be balanced before entering the coil such that

$$P_c = \min [\wp_R, \wp_O] \quad . \tag{10.3.5}$$

Then, the pressure at entry to the fan,

$$P_1 = P_c - \Delta P_{coil} - \Delta P_{filter} \quad . \tag{10.3.7}$$

iii. For units supplying 100% outdoor air (configurations (c) and (h)), $P_1 = \wp_O$, and no balancing on the inlet side is required.

The first step in estimating the pressure balance for a system involves finding the fan operating point by invoking member function `FanUnit::Control (d1, d2)` (see section 10.2.6.3) with the appropriate arguments. Where fan control is effected using a variable-speed drive, the fan speed is adjusted to match the specified mass flow rate and pressure drop. If discharge dampers are used, the fan total pressure rise Δp_t is calculated to match

---

[128] Refer to figure 6.5 in this and the following.

the specified mass flow rate and constant fan speed. If $\Delta p_t < P_2 - P_1$, the fan is too small for the duty required[129]. Otherwise, define an additional pressure drop

$$\Delta P^+ = \Delta p_t - (P_2 - P_1) \quad . \tag{10.3.8}$$

The discharge damper[130] must now be closed down to supply the required additional pressure drop ΔP^+.

The sequence of operations to balance the pressures within a system which are described above are encapsulated in a function

```
DController::Balance ();
```

Prior to invoking this function, the pressures at the roots of each duct system must be evaluated. These may be found using the techniques of section 9.4.4, provided each duct system has been fully specified. In any simulation sequence, these pressures are calculated once only, prior to initiating a moisture staircase iteration, and after the mass flow and temperatures for the duct systems have been evaluated. At this stage, the system conductance (K_{ts}; equation 9.4.10) is also calculated. As the moisture staircase iteration proceeds, the volume flow rate of air will adjust to account for variations in the air density. Use of the system characteristic to update the duct system pressure drops as the air density varies is much more efficient than performing a full duct system simulation at each step of the moisture staircase iteration. For a particular duct system the pressure at the root of the duct system may be calculated as

$$P_{sys} = \left(\frac{Q_s}{K_{ts}} \right)^2 + P_i' \tag{10.3.9}$$

where P_i' is the operating pressure at the terminal node(s) of the subtree which satisfies equation (9.4.9).

Equation (10.3.9) is sufficiently accurate for use over the restricted range of variation in the flow rate which will be encountered in a typical sequence of moisture staircase iterations, and indeed, the contribution of the variation of the duct losses to that variation is likely to be quite minor by comparison with the contribution due to the coil. The pressue drop through the coil is computed according to the method described in section 7.7. During the initialization process the surface of the coil will be assumed to be fully dry if this is an isolated run, or the first in a sequence of runs. For succeeding runs in a sequence,

---

[129] A similar situation arises in variable speed operation (see section 10.2.6.3). In either case, the current policy is to exit with error. In the longer term, it would be desirable to provide the option of operator intervention when this situation arises. This would make the software suitable for fan selection.

[130] For a system with multiple supply-air duct systems, the discharge damper for each must be closed in unison to maintain balance.

where the Zebra programme is not reloaded between runs, the coil will be initialized with the surface in whatever state was predicted by the preceding run.

Even if the duct system cannot be specified, it is still possible to estimate the fan temperature rise if the user can supply an estimate for K_{ts}, based either on data obtained from a separate duct design programme, on the basis perhaps of measurements from an existing duct system, or as the result of an educated guess. In this case, it is necessarily assumed that the zones are at atmospheric pressure. Otherwise, provided the value of K_{ts} supplied is accurate, the only loss resulting from this simplification is the ability to predict the damper settings required to balance the system. A typical design cycle might then proceed through three stages of sophistication:

i. The duct systems are not specified, and neither is a value for K_{ts} specified. In this case the user may supply an estimate for the duct temperature gains. Since the pressure rise across the fan cannot be calculated, the fan temperature rise will also need to be specified. If either of these temperature rises is not specified, a default value of 0°C is assumed.

ii. The system characteristics are specified by assigning values of $K_{ts} > 0$, for the supply-air and return-air duct systems, at least. The required fan power, and hence the temperature rise across the fan, may now be calculated using either a specified set of fan characteristics, or a specified or default fan efficiency (see section 10.2.6.3). The duct temperature gains must still be specified by the user.

iii. Full specifications are provided for all duct systems. The computation of fan and duct temperature gains is now performed internally, probably with greater accuracy than is otherwise possible. It is now also possible to estimate the damper settings required to balance flow through the system.

Finally, it has been assumed throughout this work that CAV operation implies constant *mass* flow rate. In fact, it is the *actual volume* flow rate which remains constant over the range of operation of the system. The algorithms developed in this and the preceding chapter assume that the damper settings will adjust to maintain balance with constant mass flow over the range of operation of the system. There is some justification for this approach within the context of this study. The system will in fact only be balanced to provide the specified flow distribution at one point in its operating range; that at which it is commissioned. The balancing dampers are locked in the positions which provide balance at this one point, and at all other operating conditions the flow distribution adjusts in response to the system resistances. Since the conditions pertaining during commissioning are often not specified, one set of conditions is as likely to have obtained as any other. If it is desired to study the redistribution of flow which occurs over the operating range of a system, a set of algorithms similar to those described by Tsal et al. (1990) will need to be implemented. This can readily be done using the data structures described, and there are additional reasons for adding this capability to Zebra in future, since this will provide the user with a means to examine aspects of system operation such as

a. the sensitivity of the system to errors in setting the dampers during commissioning, and
b. the effect of progressive clogging of filters on system balance.

As currently configured, the Zebra system model makes no provision for a return-air fan. This can be accommodated by extending the concepts already developed.

10.3.1. Filters.

The pressure loss through a filter may be evaluated using equation (9.4.1), with a suitable friction loss coefficient. The most convenient means of deriving the friction loss coefficient for a particular filter is usually by reference to the manufacturer's rating curves or tables. Class *Filter* is defined which provides member functions to extract the local friction coefficient from a text source, and to return the value to a calling routine. In the system model considered in this study, filters are always associated with connectors. The appropriate velocity to use in evaluating the pressure drop is selected as follows:

i. If the connector has a coil associated with it, the face velocity for the coil is used, otherwise

ii. If the connector has a fully specified duct system associated with it, the air velocity through the root section of the duct system is used, otherwise

iii. It is not possible to evaluate the filter pressure drop explicitly. If the user is specifying a system conductance, the value selected should contain an allowance for the filter pressure drop.

10.3.2. Outdoor-Air Units.

When the duct and fan temperature gains are to be found as part of the solution procedure, the procedure described in section 6.4.2 must be modified as follows:

1. The mass flow rate of air through the unit is established using equation (6.4.1). At the same time preliminary steps are taken to initialize flow conditions within each terminal unit served by the subsystem; the zone conditions are set to a state which is mutually consistent, and which is consistent with the supply-air condition, and the temperature and flow distributions, together with the pressure balances in the duct systems, are calculated. These operations establish the unit casing pressures, and do not require that the condition of the pretreated air supplied to these units be known.

2. The pressure loss through the outdoor-air duct is calculated, together with the temperature loss or gain occurring in that duct system. This establishes the condition of the air entering the unit.

3. For a *draw-through* unit, the coil-on conditions are established. If the water flow rate is fixed, the coil-off conditions may also be calculated at this stage. Otherwise, they must be calculated within the following iterative loop:

   i. Let $\delta\tilde{t}_f$ be an estimate for the fan temperature gain. Thus, if the supply-air temperature is thermostatically controlled, the chilled water flow rate must be modulated until the coil-off dry-bulb temperature is equal to the thermostat control temperature, less $\delta\tilde{t}_f$, as described in section 6.4.2.

   ii. The supply-air condition is thus established with a dry-bulb temperature equal to that at coil-off, incremented by $\delta\tilde{t}_f$, which is also the condition with which the air enters the supply-air duct.

   iii. If the supply-air duct system configuration is fully specified, the flow, temperature and pressure distributions through the system are calculated as described in chapter 9. If the duct system conductance alone is specified, this may be used to calculate the pressure drop through the duct system.

   iv. The operating point for the fan may now be found as described earlier in the present section. The configuration is identical to that of a distributed unit using 100% outdoor air. This results in a new estimate $\delta\tilde{t}_f'$ for the fan temperature gain.

   We seek then to find some estimate $\delta\tilde{t}_f$ for which

$$f(\delta\tilde{t}_f) = \delta\tilde{t}_f - \delta\tilde{t}_f' = 0 \qquad \textbf{(10.3.10)}$$

   where the objective function $f(\delta\tilde{t}_f)$ is calculated as above, and the root found using procedure zero. To establish the bracketing interval $[a..b]$, let $a = b = 0°C$, and increment b in steps of 0·5°C until inequality (9.6.2) is satisfied.

   The procedure for a *blow-through* unit is identical, except that the coil-on condition is dependent upon the trial value for $\delta\tilde{t}_f$, and hence the coil-off condition s for both control modes must be determined within the iterative loop.

4. This step is identical to step (3) of section (6.4.2).

Chapter 11. Simulation of DX Systems.

The discussion thus far has been restricted to systems using chilled water as a coolant. In the development of the data structures associated with cooling coils which are described in chapters 7 and 8, we have anticipated the need to make provision for the use of various refrigerants as alternative working fluids at a later stage. The present chapter essentially serves two purposes. The main purpose is to present some preliminary considerations concerning integration of DX systems into the modelling and design methodology which is the major theme of the thesis. The opportunity is also taken to describe and develop a unified methodology for computing the thermodynamic and transport properties of a working fluid. This is undertaken within the context of the broader aim, and comprises the first of three areas which must be addressed in connection with the task of extending the design and analysis methodology to handle DX systems. These are:

i. Specification of a working fluid. A methodology is presented based on a hierarchy of classes which effectively comprise a database and set of utility functions to manipulate the properties of fluids. These are derived from an abstract base class *fluid* which describes and defines a set of **public** member functions which evaluate and return the thermodynamic state of a *pure* fluid and its transport properties at an appropriately defined state, or following a thermodynamic process. Class *fluid* specifically addresses fluids which are pure or azeotropic mixtures. Techniques for handling more general fluid mixtures are discussed.

ii. Modelling of system components. In our work on chilled water systems the emphasis has been placed first and foremost on the cooling coil. There are good reasons for this. Our research has suggested that the dehumidifying performance of the cooling coil is arguably the most important factor determining the overall efficiency of the system. In conventional practice the cooling coil is also the least well understood component in the system. In order to model the cooling coil within the context of a system, it has also been found necessary to model the other major components of the air distribution system, namely the fan and duct network, to at least a minimal degree of sophistication. The same considerations apply of course to DX systems. The situation changes however when we consider the coolant side of the system. In our modelling work on chilled water systems we have not yet considered components other than the chilled water coil, although the discussion of the data structures we have designed for duct systems and fans foreshadows future effort in this area. There is some justification for this neglect. In most if not all instances chilled water is circulated to the various units from a central chiller plant which will usually have been specified in advance; the information available to the person making a coil selection is restricted to the temperature of the chilled water, and possibly a constraint on the volume of water available. Of course, significant savings in capital and running costs may be achieved if the chilled water distribution system is regarded as a whole, and our eventual aim is to provide the designer with the simulation and optimization tools which will facilitate this approach. In the meantime, in the context of current

consulting practice, the cooling coil can meaningfully be considered separately from the remainder of the chilled water distribution system.

The majority of DX systems are, in contrast, packaged units in which the cooling coil is closely coupled with the remaining components in the refrigeration cycle. The performance and compatibility of the various components will critically determine the performance and controllability of the system as a whole. The designer will therefore normally need to design the system as an integral unit.

Some preliminary considerations in the development of a hierarchy of classes to model refrigeration system components are presented.

iii. Control strategies. For the chilled water systems considered in chapter 6 part-load control is readily achieved by throttling the flow of chilled water. If this results in an unacceptable reduction in the coolant velocity, the LFV/HCV strategy offers a means of restoring coolant velocity by selectively deactivating coil circuits. The control of DX systems under varying load conditions is less straightforward. Friction within the tubes of the evaporator coil will cause the pressure of the evaporating refrigerant to fall progressively as it traverses the circuit. As a consequence, the temperature of the refrigerant decreases progressively, until evaporation is complete; only then does the temperature of the superheated vapour start to rise. If insufficient heat is available to evaporate the refrigerant fully, the flow is throttled by the action of a thermostatic expansion (TX) valve. The purpose of the TX valve is to ensure that the refrigerant is delivered to the compressor as a superheated vapour. However, the effect of throttling the flow is to impose an additional pressure drop, which further reduces the evaporator temperature. If this falls to the point where frosting occurs, the air side heat transfer will be impeded, and the TX valve will further throttle the flow in an attempt to maintain the superheat of the vapour. This contributes a positive feedback which further aggravates the frosting, control is lost and the valve is no longer able to maintain the required superheat, and consequently liquid refrigerant will be passed to the compressor and cause serious damage. Thus, the TX valve provides stable control only within a limited range about its design operating point.

A number of strategies have been devised in an effort to overcome the part-load control problems outlined above. However all result in an increase in energy consumption. The increasing availability of variable-speed compressors at a reasonable cost provides the possibility of achieving stable operation across the operating range of the system without incurring an energy penalty, especially when the control of refrigerant flow by varying compressor speed is combined with the use of a strategy for selectively deactivating coil circuits at part load, thus maintaining refrigerant velocity and avoiding vapour/liquid separation. The characteristics of systems which take advantage of the control opportunities offered by these extra degrees of freedom are currently being investigated.

Computer modelling offers a ready means of assessing the performance of such systems in comparison with competing control strategies across their range of

operation. A suitable model may be constructed by modifying the system simulation algorithms presented in chapter 6 to use the selected control parameters as the variables, in place of chilled water flow rate, in the iterative solution procedure. This will necessitate replacing the computational entities representing the chilled water coils with an equivalent representation for DX coils, and coupling the model with a model of the refrigerant circuit using an appropriate combination of system components and a refrigerant. The development of suitable representations for these latter entities has been considered in (i) and (ii) above.

The development of control strategies for DX systems is an area of active research, and will not be considered further in the present work.

11.1. Modelling Working Fluids.

The need to compute the thermodynamic and transport properties of fluids arises repeatedly in the simulation of thermal systems. It is possible to identify a set of operations which will provide the developer of system simulation software with a common framework for computing the properties of any pure fluid. These operations are built on a common set of thermodynamic principles, and require, to complete the implementation, the definition of an equation of state together with a set of auxiliary equations specific to the particular working fluid. Considerable economy of expression can be achieved if these operations are encapsulated in a common base class.

Class *fluid* is a base class for the computation of the properties of and processes involving fluids in the liquid and vapour phases. Class *fluid* is an abstract base class, which means that it is an error to declare objects belonging to this class. The intention is that *fluid* will provide a base class from which classes describing the properties of specific fluids (e.g. water) or classes of fluids (e.g. refrigerants) may be derived by specifying an appropriate set of functions and parameters. Given that these are correctly specified in the derived classes, class *fluid* provides a unified basis for the computation of fluid properties, and for the computation of ideal processes involving fluids.

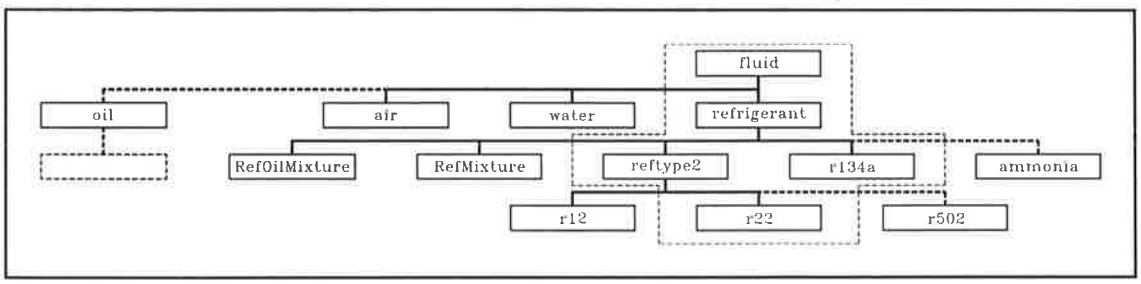


Figure 11.1. Hierarchy diagram for class *fluid* and its subclasses.

A projected hierarchy of classes derived from *fluid* is shown in figure 11.1. The broken line encloses those classes which have already been developed. A partial implementation of class water is also available; this is used in the implementation of the classes connected with the simulation of chilled water coil banks, which have been described in chapters 7 and 8. To implement a derived class, a *complete* set of functions specific to the particular

working fluid must be defined, either within the derived class, or within one of its base classes (also derived from *fluid*). This factor makes *fluid* only directly suitable for representing pure fluids, or azeotropes. More general mixtures of fluids may be handled by defining the fluid-specific member functions to specify rules for mixing two or more components, each of which will in themselves be represented by subclasses of class *fluid*. Classes *RefMixture* and *RefOilMixture* are tentatively included in the hierarchy diagram with this application in mind. An additional situation may arise where it is desired to use fluid as a base class for fluids such as air, which may not be treated as a pure fluid in the saturated liquid or wet vapour states, but may reasonably be treated as such in the superheated vapour state, which may be only state of interest in a particular range of applications. To handle such cases, it is recommended that those of the fluid-specific functions which do not refer to the vapour state should be implemented as dummy functions, providing appropriate error handling (calling them will be an error). Note that the process functions provided will not be suitable for such derived fluid classes, since they use the saturation line as a boundary point for a search interval. For this reason these functions have been declared **virtual**, as have most of the other member functions of class *fluid*, allowing them to be redefined in the subclasses. It is recommended that the same policy be followed in respect of any subclasses derived from *fluid*.

There are significant advantages to be derived from structuring the hierarchy of classes in the manner shown, since this permits a wide range of refrigerants, including pure refrigerants, azeotropes and non-azeotropic blends, and even refrigerants mixed with oil to be treated computationally in a consistent manner. Consider a function which takes as an argument a reference to an object of class *Refrigerant*, and performs some processing on it, as in the following code fragment:

```
void SimulateSystem (refrigerant& r)
{
  :
  (void) r.State (fluid::SUPERHEATED_VAPOUR, t);
  double h = r.Enthalpy ();
  :

}
```

In applications which use this function the argument passed need not be a reference to an object of class *refrigerant*; it can equally well be a reference to any subclass of *refrigerant*. The exact type need not be known when the code is compiled; the example function will call the member functions for the correct subclass at runtime. This mechanism is known as *late binding*. Thus, we may use function `SimulateSystem` in the following manner:

```
  :
  r134a r1;
  r22 r2;
  :
  switch (type)
  {
    case 1 : SimulateSystem (r1); break;
    case 2 : SimulateSystem (r2); break;
    :
  }
```

Support for new refrigerants may thus be added by making simple modifications at the topmost level of the programme. These will be completely transparent to the remaining modules which will accommodate the new refrigerant without recompiling.

In the following we present a detailed description of the structure of the base class fluid, before showing how this can be used to derive a set of base classes for refrigerants, and hence classes specific to particular refrigerants.

11.1.1. Class *fluid*.

11.1.1.1. Internal State.

Class *fluid* provides two means of computing and accessing fluid properties. The first, and generally preferred method is by updating a set of member elements **private** to *fluid*. These elements define the *internal* state of the fluid in the following. The current internal state of the fluid (or target internal state in an operation designed to update the internal state) may take one of the following values:

```
enum state { UNDEFINED, SUBCOOLED_LIQUID, SATURATED_LIQUID,
             WET_VAPOUR, SATURATED_VAPOUR, SUPERHEATED_VAPOUR };
```

where *state* is an enumerator local to class fluid. New objects of class *fluid* are always initialized to state `UNDEFINED`; an attempt to access the fluid properties while the object is in this state will return meaningless results[131]. The internal state of the object may be altered by a call to function `State`, or one of the process functions (functions for isentropic compression, isobaric condensation, isenthalpic expansion and isobaric evaporation are currently supported). These functions alter the *thermodynamic* properties only. The transport properties are treated in a similar manner to the psychrometric properties which form the member elements of class *AirState*. The member variable representing each transport property is an object of class property. Objects of this class have two fields, one containing the current value of the property, and the other indicating whether its value at the current state point has already been calculated, or not. Each transport property is flagged as unknown initially and following a change in the thermodynamic state of the fluid, insofar as this will effect a change in the transport properties. Thus, the transport properties are calculated only as an applications programme needs to access them, and are not recalculated if they are further accessed without the thermodynamic state being altered. A set of **public** functions has been provided to access the fluid properties appropriate to the current internal state. These functions take no arguments. The following set is provided:

```
state CurrentState ();
double Temperature ();          T, K
double Pressure ();             P, kPa
```

---

[131] Future releases of the software should treat any attempt to access the properties of objects in this state as an error, and provide appropriate mechanisms for dealing with it. The C++ exception handling mechanism (Stroustrup, 1991), which is now featured in most implementations of the language, provides a sophisticated mechanism for trapping and handling errors of this sort.

`double Enthalpy ();`	h, kJ/kg
`double Entropy ();`	s, kJ/kg.K
`double SpecificVolume ();`	v, m³/kg
`double SaturationTemperature ();`	T_{sat}, K
`double SaturationPressure ();`	P_{sat}, kPa
`double VapourSpecificHeatP ();`	C_p, kJ/kg.K (see footnote [132])
`double VapourSpecificHeatV ();`	C_v, kJ/kg.K
`double SaturatedVapourVolume ();`	v_g, m³/kg
`double SaturatedVapourEnthalpy ();`	h_g, kJ/kg
`double SaturatedVapourEntropy ();`	s_g, kJ/kg.K
`double DeltaVolumeVapourisation ();`	v_{fg}, m³/kg
`double DeltaEnthalpyVapourisation ();`	h_{fg}, kJ/kg
`double DeltaEntropyVapourisation ();`	s_{fg}, kJ/kg.K
`double SaturatedLiquidVolume ();`	v_f, m³/kg
`double SaturatedLiquidEnthalpy ();`	h_f, kJ/kg
`double SaturatedLiquidEntropy ();`	s_f, kJ/kg.K
`double Quality ();`	x
`double LiquidViscosity ();`	μ_l, Pa.s
`double LiquidThermalConductivity ();`	k_l, W/m.K
`double SurfaceTension ();`	σ, N/m
`double LiquidIsobaricSpecificHeat ();`	$C_{p,l}$, kJ/kg.K
`double VapourViscosity ();`	μ, Pa.s
`double VapourThermalConductivity ();`	k, W/m.K

A second set of functions is provided to evaluate and return specific fluid properties without altering the internal state. Function `State` and the process functions have been written to alter the internal state of the object from its current state to a target state with minimal redundancy of computation; using those functions in preference to this second set will be computationally more efficient in most applications. This latter set of functions parallel the preceding set, from which they are distinguished in that they take one or more arguments as required to identify the thermodynamic state at which they are to be evaluated.

---

[132] Functions to evaluate and return the isobaric and isochoric specific heat for a superheated vapour have not yet been implemented. It is suggested that these be calculated using the following expressions (McLinden et al., 1989):

$$C_v = T\left(\frac{\partial S}{\partial T}\right)_V$$

$$C_p = C_v - T\frac{\left(\frac{\partial P}{\partial T}\right)_V^2}{\left(\frac{\partial P}{\partial V}\right)_T}$$

where the partial derivatives required in the above expressions can be evaluated by differentiating the equation of state for the working fluid.

11.1.1.2. Fluid-Specific Functions.

To derive a valid (non-abstract) class using *fluid* as a base, the derived class, or one of its base classes, must define a number of functions to calculate various properties of the fluid which the derived class seeks to describe. In class *fluid*, these functions are declared as *pure* **virtual** functions; it is this fact which makes *fluid* an abstract base class. These functions are declared to be **protected** in class fluid, thus making them accessible to classes derived from fluid, while protecting them from access by unauthorized code. It is recommended that any classes derived from *fluid* should also define these functions to be **protected virtual** functions, thus enabling the derived classes in their turn to be used as base classes for the derivation of further classes. The functions supplied must not alter the values of any of the internal variables of class *fluid*.

The functions required are declared within class *fluid* in the form:

```
virtual double fluid::Peqs (double V, double T) = 0;
```

A complete list of those which need to be defined in a derived class is as follows:

1. `Peqs` Returns absolute pressure of a fluid in the vapour state as a function of specific volume and absolute temperature.

2. `Cp0` Returns ideal gas specific heat as a function of absolute temperature.

3. `heqn` Returns enthalpy of a fluid in the vapour state as a function of absolute pressure, specific volume and absolute temperature.

4. `Seqn` Returns entropy of a fluid in the vapour state as a function of specific volume and absolute temperature.

5. `Vliq` Returns specific volume of a fluid in the liquid state as a function of absolute temperature.

6. `Psatf` Returns saturation pressure as a function of absolute temperature.

7. `Tsatf` Returns saturation temperature as a function of pressure[133].

8. `dPdT` Returns dP/dT for a fluid in the saturated vapour state as a function of absolute temperature.

9. `Cpliq` Returns isobaric specific heat for a fluid in the liquid state as a function of absolute temperature.

---

[133] The expression implemented by ***Tsatf*** need be *approximate* only. The sole internal use of this function is to provide an initial estimate for the saturation temperature, which is then refined by inverting the expression implemented by ***Psatf***.

10. `dVdT` Returns $(\partial v/\partial T)_p$ for a fluid in the liquid state as a function of absolute temperature and specific volume.

11. `muliq` Returns dynamic viscosity of a fluid in the liquid state as a function of absolute temperature.

12. `kliq` Returns thermal conductivity of a fluid in the liquid state as a function of absolute temperature.

13. `sigmaf` Returns surface tension as a function of absolute temperature.

14. `muvap` Returns dynamic viscosity of a fluid in the vapour state as a function of absolute temperature.

15. `kvap` Returns thermal conductivity of a fluid state as a function of absolute temperature.

A derived class must allocate memory for and initialize any constants required by the expressions which are implemented in the above functions. In addition, a derived class must initialize a set of thermophysical constants which are used internally and may also be accessed using a set of **public** functions, as follows:

`double MolecularWeight ();`	M_w, kg/kmol
`double NormalBoilingPoint ();`	T_{br}, K
`double FreezingPoint ();`	T_{fr}, K
`double CriticalTemperature ();`	T_c, K
`double CriticalPressure ();`	P_c, K
`double CriticalVolume ();`	V_c, K
`double CompressibilityFactor ();`	z_c (see footnote [134])
`double GasConstant ();`	R, kJ/kg.K

11.1.1.3. Function `State`.

Function `State` transforms the internal state of an object of class *fluid* from its state immediately prior to the call, to a specified target state. If the transformation is effected successfully, a value of *true* is returned; otherwise `State` returns a value of *false*. The fluid-specific functions listed above are invoked to perform the transformation. In structuring the function, every effort has been made to minimize the computational effort involved in transforming the fluid from its current state to a target state.

Function `State` in fact refers to three functions which differ only in their argument lists. The C++ overloading mechanism (Stroustrup, 1991) makes this possible. Each calling protocol invokes a distinct solution algorithm. These are described below.

---

[134] Compressibility factor at the critical point.

1. Protocol 1 is to be used when the target state of the fluid lies within the *supercooled liquid* or *superheated vapour* regions. The function is declared by the statement

```
boolean fluid::State (state s,
                      double Targ,
                      double Parg);
```

where,

`s` is a variable of the enumerated type *state*, and may take a value of `SUBCOOLED_LIQUID` or `SUPERHEATED_VAPOUR`.

`Targ` is the target absolute temperature (K). If a client function specifies a value of `Targ` = 0·0, and the current state is not `UNDEFINED`, the target state is to be reached via an isothermal process, in which case member variable `fluid::isothermal` is set *true*.

`Parg` is the target absolute pressure (kPa). If a client function specifies a value of `Parg` = 0·0, and the current state is not `UNDEFINED`, the target state is to be reached via an isobaric process, in which case member variable `fluid::isobaric` is set *true*.

Member variable `fluid::isothermal` will also be set *true* if the target temperature is the same as the current temperature; likewise, member variable `fluid::isobaric` will be set *true* if the target pressure is the same as the current pressure.

Now let

P	be the target pressure,
T	be the target temperature,
P_{sat}	be the saturation pressure corresponding to the target temperature, $P_{sat} = f(T)$,
T_{sat}	be the saturation temperature corresponding to the target pressure, $T_{sat} = f(P)$,
P'	be the current pressure,
T'	be the current temperature,
P_{sat}'	be the saturation pressure corresponding to the current temperature, $P_{sat}' = f(T')$,
T_{sat}'	be the saturation temperature corresponding to the current pressure, $T_{sat}' = f(P')$.

The target and current states for the remaining thermodynamic variables are identified using the same notation. We now need to consider two cases; that where the target state is a subcooled liquid, and that where it is a superheated vapour.

Case a. Target state is a *subcooled liquid*. The logic of an algorithm to treat this case is shown in figure 11.2. Certain preliminary checks are carried out first. If an *isothermal* process has been specified, the target state can only be reached if

$$P \geq P_{sat}' \quad . \tag{11.1.1}$$

Similarly, if an *isobaric* process has been specified, the target state will only be reachable if

$$T \leq T_{sat}' \quad . \tag{11.1.2}$$

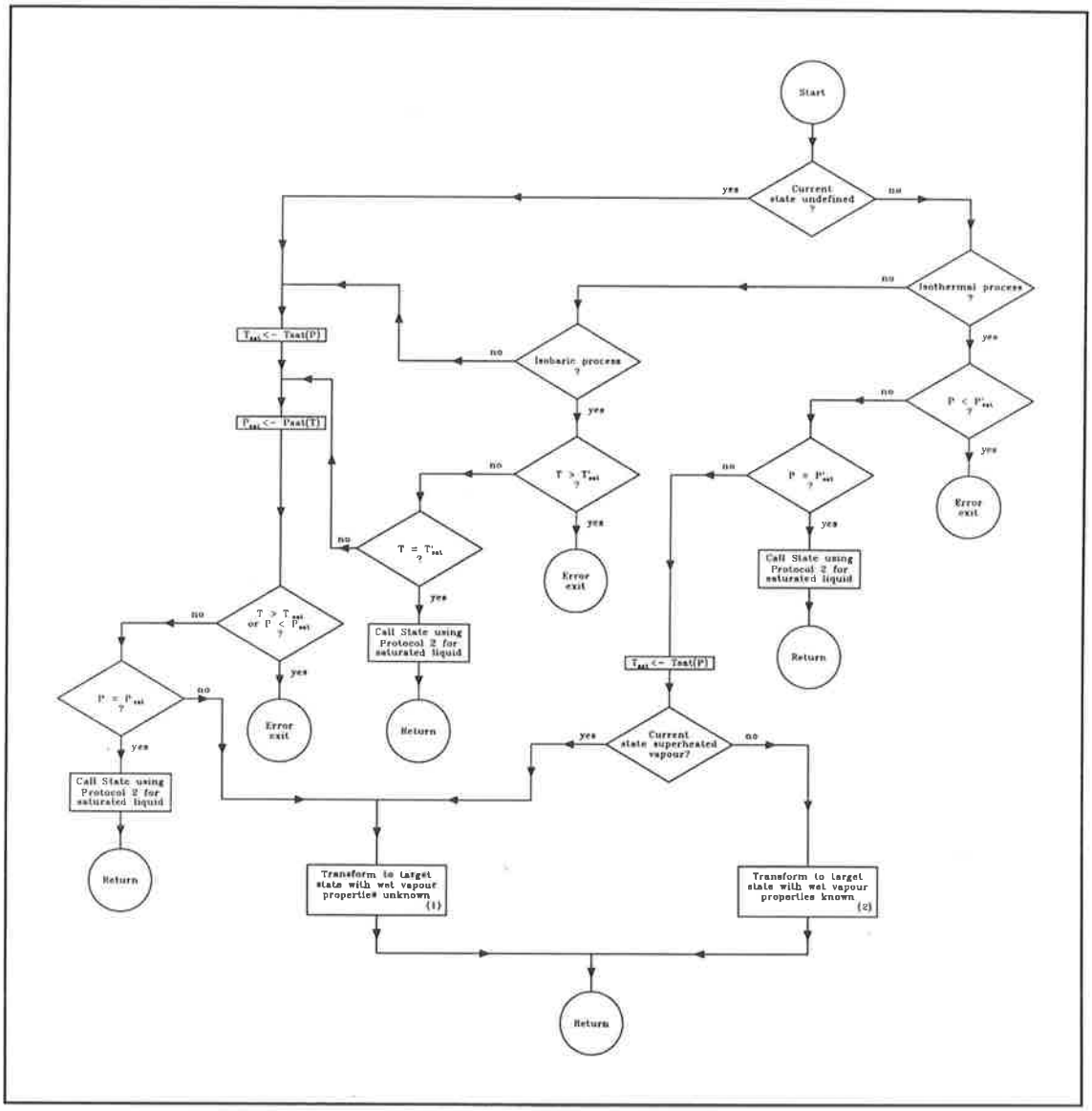


Figure 11.2. Logic to transform a fluid of arbitrary state to a subcooled liquid state.

For any process, the target state must satisfy the criteria

$$P \geq P_{sat} \tag{11.1.3a}$$

and

$$T \leq T_{sat} \quad . \tag{11.1.3b}$$

If any of the relations (11.1.1-11.1.3) are satisfied by strict equalities, the designated target state will in fact be a *saturated* rather than a *subcooled* liquid, and will be processed accordingly; a call to function `State` will be made using calling protocol 2. Otherwise, the following target state for the fluid will be found using the following sequence of operations:

1. The gas properties for a fluid saturated at pressure P_{sat} are calculated. Specific volume is found by invoking member function `SpecificVolume` with P_{sat} and T as arguments. This will invert the equation of state as implemented in the fluid-specific member function `Peqs`. The corresponding values for enthalpy and entropy are then found by calling the fluid-specific functions `heqn` and `Seqn` respectively.

2. The corresponding saturated liquid properties are calculated. The specific volume is calculated by invoking member function `Vliq` for the fluid, whence

$$v_{fg} = v_g - v_f \ . \tag{11.1.4}$$

The saturated liquid enthalpy follows from the Clausius-Clapeyron equation:

$$h_{fg} = Tv_{fg}\left(\frac{dP}{dT}\right)_{sat} \tag{11.1.5a}$$

$$h_f = h_g - h_{fg} \tag{11.1.5b}$$

and the entropy,

$$s_{fg} = \frac{h_{fg}}{T} \tag{11.1.6a}$$

$$s_f = s_g - s_{fg} \tag{11.1.6b}$$

where $(dP/dT)_{sat}$ is calculated using member function `dPdT`.

3. The target state may now be reached by an isothermal process commencing at the saturated liquid state. For such a process it can be shown that (Smith and Van Ness, 1975)

$$h - h_f = \int_{P_{sat}}^{P} (1 - \beta T)v\,dP \tag{11.1.7a}$$

and

$$s - s_f = -\int_{P_{sat}}^{P} \beta v\,dP \tag{11.1.7b}$$

where

$$\beta \equiv \frac{1}{v}\left(\frac{\partial v}{\partial T}\right)_P \quad . \tag{11.1.7c}$$

Now, for a fluid in the liquid state, specific volume is only a weak function of pressure, and the pressure dependency may be neglected for all but the most demanding applications. The above equations may thus be integrated to give the following expressions for the enthalpy and entropy of the subcooled liquid:

$$h = h_f + \left[v_f - T\left(\frac{\partial v}{\partial T}\right)_p\right](P - P_{sat}) \tag{11.1.7}$$

and

$$s = s_f + \left(\frac{\partial v}{\partial T}\right)_p (P - P_{sat}) \quad . \tag{11.1.8}$$

In the above, $(\partial v/\partial T)_p$ can be evaluated using member function `dVdT` for the working fluid. The specific volume at the target state, $v = v_f$.

If the process required to reach the target state is isothermal, and the current state of the fluid is not `UNDEFINED` or `SUPERHEATED_VAPOUR`, the saturation properties for the fluid will already be known, and steps (1) and (2) above may be omitted. If furthermore the prior state of the fluid is `SUBCOOLED_LIQUID`, $(\partial v/\partial T)_p$ will already be known, and need not be recalculated.

Case b. Target state is *superheated vapour*. The logic of an algorithm to treat this case is shown in figure 11.3. The similarities between this and the preceding case will be apparent, and no further explanation is necessary.

2. Protocol 2 is to be used when the target state of the fluid is a *saturated liquid* or a *saturated vapour*. The function is declared by the statement

```
boolean fluid::State (state s,
                      double darg,
                      int iarg);
```

where,

`s` is a variable of the enumerated type *state*, and may take a value of `SATURATED_LIQUID` or `SATURATED_VAPOUR`.

`darg` is a double precision argument, the meaning of which is determined by the third argument.

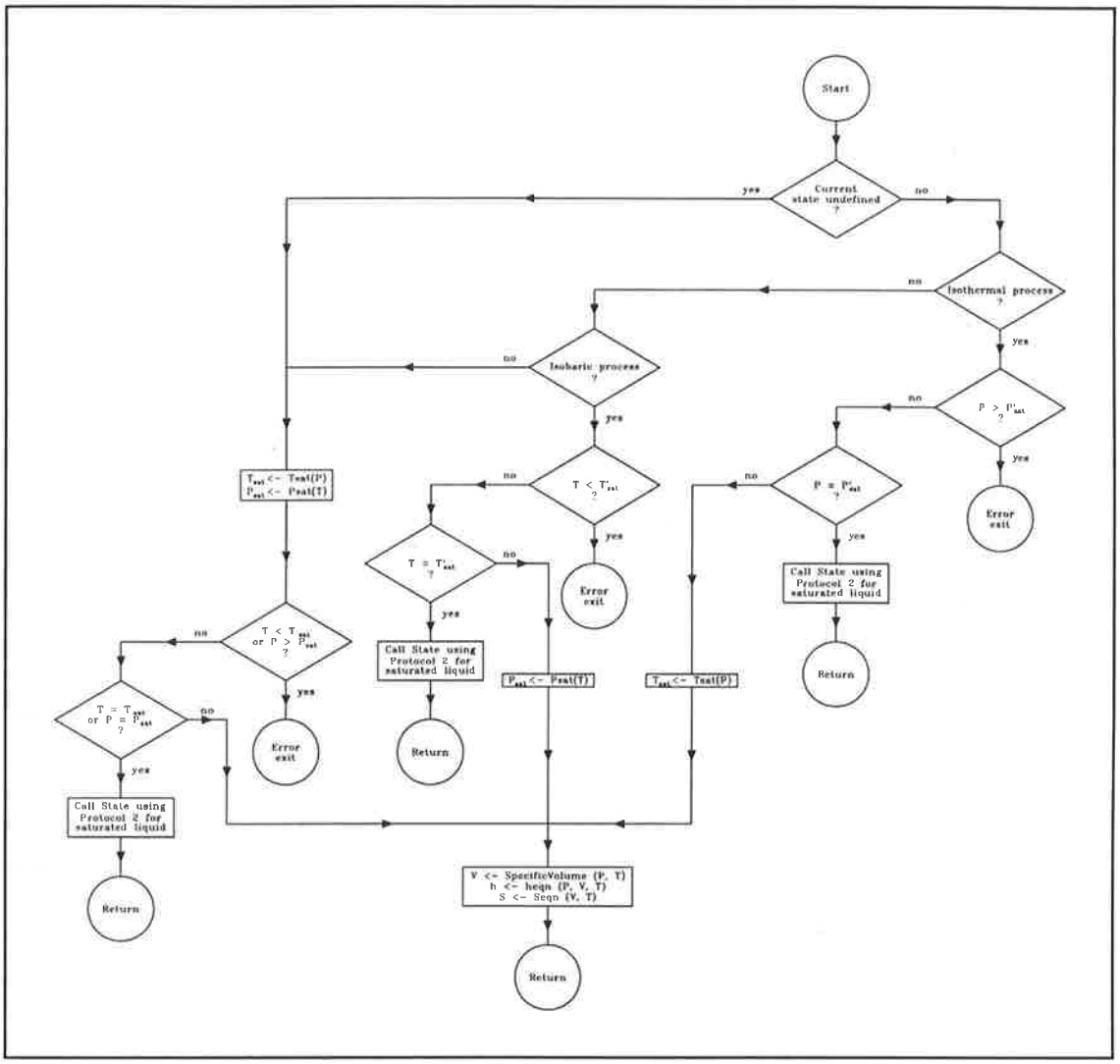


Figure 11.3. Logic to transform a fluid of arbitrary state to a superheated vapour state.

`iarg` is an integer variable which flags the interpretation which is to be placed on the second argument:

`iarg` = 0 means that `darg` is an absolute temperature (K). If a client function specifies values of `darg` = 0·0 and `iarg` = 0, the target state is to be reached via an *isothermal* process.

`iarg` = 1 means that `darg` is an absolute pressure (kPa). If a client function specifies values of `darg` = 0·0 and `iarg` = 1, the target state is to be reached via an *isobaric* process.

If an isothermal process is specified, and the current state of the fluid is `SATURATED_VAPOUR`, `WET_VAPOUR`, `SATURATED_LIQUID` or `SUBCOOLED_LIQUID`, the properties of the fluid in the saturated liquid and saturated vapour states will already be known. Otherwise they must be calculated using equations (11.1.4-11.1.6). The current fluid properties can subsequently be set accordingly, with x = 0·0 for a saturated liquid, and x = 1·0 for a saturated vapour.

3. Protocol 3 is to be used when the target state of the fluid is a *wet vapour*. The function is declared by the statement

```
boolean fluid::State (state s,
                      double darg1,
                      double darg2,
                      int iarg);
```

where,

`s` is a variable of the enumerated type *state*, and may only take the value `WET_VAPOUR`.

`darg1` and `darg2` are double precision arguments representing a pair of fluid properties. The properties represented by these arguments are determined by the fourth argument.

`iarg` is an integer variable which flags the interpretation which is to be placed on the second and third arguments:

`iarg` = 0 : (`darg1`, `darg2`) ⇒ (T, v) (default)
`iarg` = 1 : (`darg1`, `darg2`) ⇒ (T, x)
`iarg` = 2 : (`darg1`, `darg2`) ⇒ (P, v)
`iarg` = 3 : (`darg1`, `darg2`) ⇒ (P, x)

If a client function specifies a value of `darg1` = 0·0 when `darg1` represents pressure, the target state is to be reached from the current state following an isothermal process. Similarly, if `darg1` = 0·0 when `darg1` represents temperature, an isobaric process is to be used.

If x is specified as an input parameter (*iarg* = 1 or 3), the process used to reach the target state must be isothermal, and must proceed from a current state for which both v_f and v_g are known (`SUBCOOLED_LIQUID`, `SATURATED_LIQUID`, `WET_VAPOUR` or `SATURATED_VAPOUR`). The fluid properties may then be found simply by interpolating between the saturation states. That is,

$$v = v_f + x v_{fg} \qquad \textbf{(11.1.9a)}$$

$$h = h_f + x h_{fg} \qquad \textbf{(11.1.9b)}$$

$$s = s + x s_{fg} \qquad \textbf{(11.1.9c)}$$

If v is specified as an input parameter (`iarg` = 0 or 2), the properties of the fluid at the target state are found as follows:

1. The saturation temperature and pressure must be found. If an isothermal process is specified, the saturation pressure corresponding to the target temperature will already be known. Otherwise the saturation temperature or pressure must be set as specified, and the corresponding saturation pressure or temperature evaluated as required by invoking fluid-specific member function `Psatf` or `Tsatf`, as appropriate.

2. If the specified process is isothermal, and the current state of the fluid is `SUBCOOLED_LIQUID`, `SATURATED_LIQUID`, `WET_VAPOUR` or `SATURATED_VAPOUR`, the saturated liquid and vapour properties pertaining to the target state will already be known. Otherwise they must be calculated using equations (11.1.4-11.1.6).

3. The wet vapour properties may now be calculated. Quality (x) is found by inverting equation (11.1.9a), and enthalpy and entropy follow from equations (11.1.9b) and (11.1.9c) respectively.

11.1.1.4. Process Functions.

The internal state of an object of class fluid may also be altered by invoking one of a number of process functions. Those which have been implemented so far are described below.

11.1.1.4.1. Isentropic Compression.

The state of a fluid following an isentropic compression may be evaluated by a call to the following function:

```
boolean fluid::IsentropicCompression (double P);
```

This function handles the case of a process working between pressures P_1 and P_2, where $P_2 > P_1$, and P_2 is below the critical pressure. The latter restriction is of little significance in respect of air conditioning applications, although it might need to be relaxed if the software is to be used at a later date to simulate CO_2 cycles. Provided the above restrictions are satisfied, the state of the fluid following an isentropic compression can be calculated as follows:

1. Saturation entropy s_2' corresponding to pressure P_2 is calculated. If $s_1 > s_2'$, the vapour is superheated at the end of the process; otherwise it is in the wet vapour state. The latter condition could arise if the fluid is in the wet vapour state at the beginning of the compression process (wet compression). In principle at least it may also be encountered in the compression of superheated vapours of refrigerants such as r113 and r114. The saturation entropy s_2' is also used as a boundary point for the bracketing interval for wet vapour calculations (step 3 below).

2. If the fluid is a wet vapour at the end of the compression process, the saturated liquid and vapour properties corresponding to the target pressure P are calculated using equations (11.1.4) to (11.1.6). Quality (x) follows by inverting equation (11.1.9c), and entropy and enthalpy may be calculated from equations (11.1.9a) and (11.1.9b).

3. If the fluid is in the superheated state at the end of the compression process, then the state point will lie within an interval on the locus $P = P_2$ such that $s_2' < s_2 \leq s_2''$. To find the point we need to solve for the root of the function

$$f(\tilde{T}_2) \equiv \tilde{s}_2 - s_1 = 0 \qquad \textbf{(11.1.10)}$$

where $\tilde{T}_2$ is a trial value for the temperature, and $\tilde{s}_2$ is the corresponding entropy. The function may be evaluated at the trial point using the following steps:

i. Specific volume $\tilde{v}_2$ at the trial point is found by invoking member function `SpecificVolume` with P_2 and $\tilde{T}_2$ as arguments.
ii. Entropy $\tilde{s}_2$ is found by invoking member function `Entropy` with $\tilde{v}_2$ and $\tilde{T}$ as arguments.
iii. The value of $f(\tilde{T})$ follows.

Let T_2' be the temperature at point (P_2, s_2'), and T_2'' be the temperature at point (P_2, s_2''). T_2' is simply evaluated as the saturation temperature at pressure P_2. An initial estimate for T_2'' can be made using the ideal gas isentropic compression equation:

$$\tilde{T}_2'' = T_1\left(\frac{P_2}{P_1}\right)^{\frac{\gamma - 1}{\gamma}} \qquad \textbf{(11.1.11a)}$$

where

$$\gamma = \frac{C_p^0}{C_v^0} \qquad \textbf{(11.1.11b)}$$

and

$$C_p^0 - C_v^0 = R \quad . \qquad \textbf{(11.1.11c)}$$

C_p^0 is evaluated by invoking the fluid-specific member function `Cp0`. If $f(\tilde{T}_2'') \geq 0$, $\tilde{T}_2''$ establishes the upper boundary of the bracketing interval. Otherwise, $\tilde{T}_2''$ may be repeatedly multiplied by a factor of 1·1 (say) until the bracketing interval has been established, and the root of equation (11.1.10) subsequently found using function zero. The only thermodynamic property which remains to be evaluated is the enthalpy; this may be found by calling member function `Enthalpy`.

11.1.1.4.2. Isobaric Condensation.

This process is implemented using a function which supports two alternative calling sequences:

```
boolean fluid::IsobaricCondensation ();
boolean fluid::IsobaricCondensation (double Ts);
```

If the first form of the function is used, the condensation process will terminate at the saturated liquid line. This point is found by a call to `State` with the following arguments:

```
State (SATURATED_LIQUID, 0.0, 1);
```

The argument in the second form of the function specifies a certain amount of subcooling (°C). The desired end-point of the process may be reached by an isobaric process, terminating at temperature $T = T_{sat} - T_s$. This point is found by a call to `State` with the following arguments:

```
State (SUBCOOLED_LIQUID, T);
```

11.1.1.4.3. Isenthalpic Expansion.

The state of the fluid following an isenthalpic expansion may be evaluated by a call to the following function:

```
boolean fluid::IsenthalpicExpansion (double Pf);
```

The argument `Pf` specifies the pressure of the working fluid at completion of the expansion process, where this is less than the pressure at the commencement of the process. It is assumed that, at the beginning of the process, the fluid is a wet vapour, a saturated liquid, or a subcooled liquid. It is further assumed that, at the end of the process, the fluid is in the wet vapour state, in which state `Pf` is the saturation pressure. The saturation temperature may be found by invoking member function `SaturationTemperature`, and the remaining saturated fluid properties calculated using equations (11.1.4) to (11.1.6). Since the enthalpy at completion of the process is by definition, known, quality may be found using equation (11.1.9b), and specific volume and entropy follow from equations (11.1.9a) and (11.1.9c).

11.1.1.4.4. Isobaric Evaporation.

Isobaric evaporation is implemented using a function which supports two alternative calling sequences:

```
boolean fluid::IsobaricEvaporation ();
boolean fluid::IsobaricEvaporation (double Ts);
```

The first form is used to specify an evaporation process terminating at the saturated vapour line. This point is found by a call to `State` with the following arguments:

```
State (SATURATED_VAPOUR, 0.0, 1);
```

By analogy with the isobaric condensation function, the argument in the second form of the function specifies a certain amount of superheat (°C). The desired end-point of the process may be reached by an isobaric process terminating at temperature $T = T_{sat} + T_s$. This point is found by a call to `State` with the following arguments:

```
State (SUPERHEATED_VAPOUR, T);
```

Note that this function, as currently implemented, will not handle an evaporation process terminating within the wet vapour region, and is thus not suitable for simulating cycles involving wet vapour compression.

11.1.2. Class *refrigerant*.

Class *refrigerant* inherits class *fluid* as a public base class, and in its turn provides a base class for various refrigerants and families of refrigerants. Class *refrigerant* defines a set of functions providing the functionality required of the fluid-specific pure **virtual** functions listed in section 11.1.1.2 above. These provide a set of default functions for computing the properties of halocarbon refrigerants. The expressions evaluated by these functions will be described below. In most cases derived classes will need to allocate memory for and initialize a number of variables; details of these are also provided[135]. All functions are declared to be **protected virtual** functions, which means that they may be redefined by any derived classes, if so desired. Class *r134a*, which is derived from *refrigerant* will be described below in section 11.1.3. Section 11.1.4 describes class *reftype2*, which is a base class derived from *refrigerant*.

11.1.2.1. Function `Peqs`.

The function `Peqs` implements an equation of state of the Martin-Hou type which, in its most general form may be expressed in the form (Chan and Haselden, 1981a):

$$P = \frac{RT}{V-b} + \frac{A_2 + B_2T + C_2\left[\mu\exp\left(\frac{-kT}{T_c}\right) + \frac{\nu}{T^3}\right]}{(V-b)^2} + \frac{A_3 + B_3T + C_3\exp\left(\frac{-kT}{T^3}\right)}{(V-b)^3} + \frac{A_4 + B_4T + C_4\left[\mu\exp\left(\frac{-kT}{T_c}\right) + \frac{\nu}{T^3}\right]}{(V-b)^4} + \frac{A_5 + B_5T + C_5\exp\left(\frac{-kT}{T^3}\right)}{(V-b)^5} + \frac{A_6 + B_6T + C_6\exp\left(\frac{-kT}{T^3}\right)}{\exp(aV)[1 + C'\exp(aV)]} \qquad \textbf{(11.1.12)}$$

---

[135] Currently objects of type *refrigerant* can be declared, but attempting to use them will lead to run-time errors, since no memory will have been allocated for the fluid-specific constants. The policy of requiring the derived classes to allocate memory has been adopted since, in the case of some of the following expressions, and alternatives defined in derived classes, the order of the equation will depend on the working fluid. Objects of properly-configured subclasses of *refrigerant* only should be created. There are a number of possible methods by which security could be improved in later releases of the software, thus eliminating the possibility of run-time errors through inadvertent creation of objects of the base class.

in which the sixth term is negligible for many refrigerants. Derived classes which use this function must:

i. Initialize a boolean variable indicating whether the sixth term is to be evaluated for this refrigerant.

ii. Allocate memory for arrays ***A***, ***B*** and ***C***.

iii. Initialize arrays ***A***, ***B*** and ***C***, together with constants b, k, μ and v, and if appropriate, C' and a.

A critique of this form of the equation of state, together with a number of commonly-used alternative formulations, is presented by Gosney (1982).

11.1.2.2. Function `CpO`.

The ideal gas heat capacity equation is commonly expressed in the form of a modified polynomial, representing either the isobaric or the isochoric specific heat. The form used here as the default for halocarbon refrigerants is the ideal gas isobaric heat capacity equation given by Wilson and Basu (1988):

$$C_p^0 = C_{p1} + C_{p2}T + C_{p3}T^2 + C_{p4}T^3 + \frac{C_{p5}}{T} \quad . \tag{11.1.13}$$

Derived classes which use this function must allocate memory for and initialise array $\boldsymbol{C_p}$.

11.1.2.3. Function `heqn`.

The enthalpy equation is derived from equations (11.1.2) and (11.1.13) above using the thermodynamic relationship:

$$dh = C_v^0 + d(Pv) - \left[P - T\left(\frac{\partial P}{\partial T}\right)_v\right]dv \tag{11.1.14}$$

from which

$$h_{(T,P)} = \int_{T_{ref}}^{T}(C_P^0 - R)dT + \int_{P_{ref},v_{ref}}^{P,v} d(Pv) - \int_{v_{ref}}^{v}\left[P - T\left(\frac{\partial P}{\partial T}\right)_v\right]dv + h_{(T_{ref},P_{ref})} \tag{11.1.15}$$

in which T_{ref}, P_{ref} and V_{ref} define an arbitrary reference state. Substituting equations (11.1.12) and (11.1.13) and integrating gives

$$
\begin{aligned}
h = h_{ref} &+ (Pv - RT) \\
&+ \left(C_{p1}T + C_{p2}\frac{T^2}{2} + C_{p3}\frac{T^3}{3} + C_{p4}\frac{T^4}{4} + C_{p5}\ln T \right) \\
&+ \left(\frac{A_2}{(v-b)} + \frac{A_3}{2(v-b)^2} + \frac{A_4}{3(v-b)^3} + \frac{A_5}{4(v-b)^4} \right) \\
&+ e^{-kT_r}(1 + KT_r) \times \\
&\left(\frac{C_2}{(v-b)} + \frac{C_3}{2(v-b)^2} + \frac{C_4}{3(v-b)^3} + \frac{C_5}{4(v-b)^4} \right)
\end{aligned}
\qquad \textbf{(11.1.16)}
$$

where H_{ref} is the enthalpy at the arbitrary reference state (see above). Two conventions are in common use for establishing the datum from which enthalpy values are measured. ASHRAE defines the enthalpy of saturated liquid to be zero at a temperature of -40°C. The IIR defines the enthalpy of saturated liquid to be 200 kJ/kg at a reference temperature of 0°C. The choice of reference state to use can be set using member functions `SetReferenceStateIIR` and `SetReferenceStateASHRAE`. The IIR reference state is used as the default. In specifying a class for a specific refrigerant derived from class `refrigerant`, the user must initialise an array **h0[2]**, the elements of which are:

h0[0] : Enthalpy (kJ/kg) at the IIR reference state;
h0[1] : Enthalpy (kJ/kg) at the ASHRAE reference state.

11.1.2.4. Function `Seqn`.

The entropy equation is derived from equations (11.1.12) and (11.1.13) above using the thermodynamic relationship:

$$ds = \frac{C_v^0}{T}dT + \left(\frac{\partial P}{\partial T}\right)_v dv \qquad \textbf{(11.1.17)}$$

from which

$$s_{(T,P)} = \int_{T_{ref}}^{T} \frac{C_v^0}{T}dT + \int_{v_{ref}}^{v} \left(\frac{\partial P}{\partial T}\right)_v dv + s_{(T_{ref},P_{ref})} \qquad \textbf{(11.1.18)}$$

in which T_{ref}, P_{ref} and V_{ref} define an arbitrary reference state. Substituting equations (11.1.12) and (11.1.13) and integrating gives

$$
\begin{aligned}
s = s_{ref} &+ \left(C_{p1} \ln T + C_{p2} T + C_{p3} \frac{T^2}{2} + C_{p4} \frac{T^3}{3} - \frac{C_{p5}}{T} \right) \\
&+ R \ln \left(\frac{(v - b) P_0}{RT} \right) \\
&- \left(\frac{B_2}{(v - b)} + \frac{B_3}{2(v - b)^2} + \frac{B_4}{3(v - b)^3} + \frac{B_4}{4(v - b)^4} \right) \\
&+ \frac{K}{T_c} e^{-KT_r} \left(\frac{C_2}{(v - b)} + \frac{C_3}{2(v - b)^2} + \frac{C_4}{3(v - b)^3} + \frac{C_5}{4(v - b)^4} \right)
\end{aligned}
\qquad \textbf{(11.1.19)}
$$

where P_0 is standard atmospheric pressure (101·325 kPa), and s_{ref} is the entropy at the arbitrary reference state (see above). Two conventions are in common use for establishing the datum from which entropy values are measured. ASHRAE defines the entropy of saturated liquid to be zero at a temperature of -40°C. The IIR defines the entropy of saturated liquid to be 1 kJ/kg.K at a reference temperature of 0°C. The choice of reference state to use can be set using member functions `SetReferenceStateIIR` and `SetReferenceStateASHRAE`. The IIR reference state is used as the default. In specifying a class for a specific refrigerant derived from class *refrigerant*, the user must initialise an array **S0[2]**, the elements of which are:

S0[0**]** : Entropy (kJ/kg.K) at the IIR reference state;
S0[1**]** : Entropy (kJ/kg.K) at the ASHRAE reference state.

11.1.2.5. Function `Vliq`.

The specific volume of a liquid refrigerant is found using the standard liquid density correlation of Hou and Martin (1959):

$$\rho_L = \rho_C + \sum_{N=1}^{4} D_N (1 - T_r)^{\frac{N}{3}} \quad . \qquad \textbf{(11.1.20)}$$

Derived classes which use this function must allocate storage for the array **D[i]**, i=1,..,4, and initialise the elements appropriately.

11.1.2.6. Function `Psatf`.

The default expression provided to compute the saturation pressure of a refrigerant uses the extended form of the Antoine equation, as presented by Wilson and Basu (1988):

$$\ln P_{sat} = A + \frac{B}{T} + CT + DT^2 + \frac{E(F - T)}{T} \ln(F - T) \quad . \qquad \textbf{(11.1.21)}$$

Derived classes which use this function must allocate storage for an array of coefficients, and initialise the elements appropriately.

11.1.2.7. Function `Tsatf`.

The Antoine equation in the form presented by Cleland (1986) provides an estimate of the saturation temperature:

$$T_{sat} = \frac{a_2}{\ln(P) - a_1} - a_3 \quad . \tag{11.1.22}$$

Derived classes which use this function must allocate storage for the array **a[i]**, i=1,..,3, and initialise the elements accordingly.

11.1.2.8. Function `dPdT`.

dP/dT for a saturated vapour is evaluated by differentiating equation (11.1.21), thus:

$$\frac{dP}{dT} = \exp\left(A + \frac{B}{T} + CT + DT^2 + \frac{E(F-T)}{T}\ln(F-T) \right) \times \left(-\frac{B}{T^2} + C + 2DT - \frac{E}{T}\left[\frac{F\ln(F-T)}{T} + 1 \right] \right) \tag{11.1.23}$$

11.1.2.9. Function `Cpliq`.

The default expression offered for the evaluation of the specific heat for a liquid refrigerant is a polynomial expression of the form:

$$C_p = \sum_{i=1}^{N} A_i \left(1 - \frac{T}{T_c} \right)^{\frac{i}{3}} \quad . \tag{11.1.24}$$

Derived classes using this expression must specify the order of the polynomial, and allocate storage for and initialise the polynomial coefficients.

11.1.2.10. Function `dvdT`.

$(\partial v/\partial T)_p$ can be evaluated as

$$\left(\frac{\partial v}{\partial T} \right)_p = -\frac{(\partial \rho/\partial T)_p}{\rho^2} \tag{11.1.25}$$

where $(\partial \rho/\partial T)_p$ can be found by differentiating equation (11.1.20):

$$\left(\frac{\partial \rho}{\partial T} \right)_p = -\frac{1}{3T_c} \sum_{N=1}^{4} N D_N (1 - T_r)^{\frac{N}{3} - 1} \quad . \tag{11.1.26}$$

11.1.2.11. Function `muliq`.

To evaluate the dynamic viscosity of a halocarbon refrigerant, Jung and Radermacher (1991) recommend the use of the following equation, which was proposed originally as an experimental correlation by Phillips and Murphy (1970a, b):

$$\log_{10}\mu = A + \frac{B}{T} + CT + DT^2 \qquad \textbf{(11.1.27)}$$

where μ in the above equation is in units of centipoise (cp). Jung and Radermacher have published coefficients for some 16 refrigerants for use with this equation. However, they caution against possible errors which may result if the equation is applied uncritically, particularly in respect of refrigerants which were not included in the original experiments of Phillips and Murphy. Refrigerants r152a and r124 are mentioned as showing particularly large discrepancies between experimental data and the predictions of equation (11.1.27).

Derived classes using this function must allocate storage for and initialise the coefficients for the expression.

11.1.2.12. Function `kliq`.

The default function provided to evaluate the thermal conductivity of a halocarbon refrigerant is the polynomial fit of Yata et al. (1984):

$$k = A + BT + CT^2 \quad . \qquad \textbf{(11.1.28)}$$

Jung and Radamacher (1991) have published coefficients for some 14 halocarbon refrigerants for use with this equation.

Derived classes using this function must allocate storage for and initialise the coefficients in the expression.

11.1.2.13. Function `sigmaf`.

Jung and Radermacher (1991) have evaluated the predictive method of Brock and Bird (1955) for calculating the surface tension of halocarbon refrigerants, and found it to be accurate and consistent. This method is implemented using the following equations:

$$\sigma = P_c^{2/3} T_c^{2/3} Q (1 - T_r)^{11/9} \qquad \textbf{(11.1.29)}$$

where

$$Q = 0{\cdot}1196\left[1{\cdot}0 + T_{br}\frac{\ln\left(\frac{P_c}{1{\cdot}01325}\right)}{(1{\cdot}0 - T_{br})}\right] - 0{\cdot}279 \quad . \qquad \textbf{(11.1.30)}$$

Note that the above expressions rely only on the thermophysical constants for the refrigerants, and are thus applicable to a broad range of refrigerants, in the absence of experimental data. To enhance computational efficiency, the constructor for a derived class must initialize a multiplicative constant $cst \equiv P_c^{2/3}.T_c^{2/3}.Q$.

11.1.2.14. Function `muvap`.

Based on dimensional analysis, Stiehl and Thodos (1962) developed an expression for the viscosity of polar gases at normal pressures. This expression has been modified by Nagaoka et al. (1986) to predict the viscosity of gaseous fluorocarbon refrigerants at atmospheric pressure, and takes the form

$$\mu^* \xi = (0{\cdot}5124 T_r - 0{\cdot}0517)^{0{\cdot}82} z_c^{-0{\cdot}81} \qquad \textbf{(11.1.31a)}$$

$$\xi = T_c^{1/6} M_w^{-1/2} P_c^{-2/3} \qquad \textbf{(11.1.31b)}$$

where μ^* is the viscosity (Pa.s $\times 10^6$) at atmospheric pressure. For pressures other than atmospheric, Jung and Radermacher (1991) recommend that the empirical correlations of Stiehl and Thodos (1964b) be applied to the above expression.

For $\rho_r < 0{\cdot}1$:

$$(\mu - \mu^*)\xi = 0{\cdot}761931 \rho_r^{1{\cdot}111} \quad . \qquad \textbf{(11.1.32a)}$$

For $0{\cdot}1 < \rho_r \leq 0{\cdot}9$:

$$(\mu - \mu^*)\xi = 2{\cdot}79283(9{\cdot}045 \rho_r + 0{\cdot}63)^{1{\cdot}739} \quad . \qquad \textbf{(11.1.32b)}$$

For $0{\cdot}9 < \rho_r \leq 2{\cdot}6$:

$$(\mu - \mu^*)\xi = 4{\cdot}6 \times 10^{CI}$$
$$CI = 3{\cdot}0 - 10^{(0{\cdot}6439 - 0{\cdot}1005 \rho_r)} \quad . \qquad \textbf{(11.1.32c)}$$

For computational efficiency, the constructor for a derived class must initialize constants ξ and $zcp \equiv z_c^{-0{\cdot}81}$.

11.1.2.15. Function `kvap`.

The thermal conductivity of a vapour at atmospheric pressure may be found using an Eucken-type equation, which is given by Jung and Radermacher (1991) in the form

$$k^* = 0{\cdot}001 \mu^* C_v^0 \left(C_1 + C_2 \frac{R}{C_v^0} \right) \qquad \textbf{(11.1.33)}$$

in which k^* is the thermal conductivity of the vapour at atmospheric pressure (W/m.K) and μ^* is the vapour viscosity at atmospheric pressure (Pa.s $\times 10^6$). For refrigerants, Jung and Radermacher recommend that values of $C_1 = 1{\cdot}235$ and $C_2 = 1{\cdot}9$ be used. For

operation at other than atmospheric pressure, the following correction factors due to Stiehl and Thodos (1964a) can be used.

For $\rho_r < 0{\cdot}5$:

$$(k - k^*)\lambda z_c^5 = 14 \times 10^{-8}(e^{-0{\cdot}535\rho_r} - 1{\cdot}0) \quad . \qquad \textbf{(11.1.34a)}$$

For $0{\cdot}5 < \rho_r \leq 2{\cdot}0$:

$$(k - k^*)\lambda z_c^5 = 13{\cdot}1 \times 10^{-8}(e^{0{\cdot}67\rho_r} - 1{\cdot}069) \quad . \qquad \textbf{(11.1.34b)}$$

For $2{\cdot}0 < \rho_r \leq 2{\cdot}8$:

$$(k - k^*)\lambda z_c^5 = 2{\cdot}976 \times 10^{-8}(e^{1{\cdot}155\rho_r} + 1{\cdot}069) \quad . \qquad \textbf{(11.1.34c)}$$

In the above,

$$\lambda = T_c^{1/6} M_w^{1/2} P_c^{-2/3} \quad . \qquad \textbf{(11.1.34d)}$$

For computational efficiency, constructors for derived classes must initialize constants λ and $zc5 \equiv z_c^5$.

11.1.3. Class *r134a*.

As a direct replacement for r12, r134a is probably the most important of the new generation of ozone-friendly refrigerants. The thermodynamic properties of r134a are described in detail by Wilson and Basu (1988), McLinden et al. (1989), Piao et al. (1989) and ICI (1991). A class *r134a* has been implemented, derived from class *refrigerant*. The user is referred to Wilson and Basu (1988) for details of the constants used in the thermodynamic equations, and to Jung and Radermacher (1991) for details of the constants used in the transport equations, with the following exceptions:

a. Functions `heqn` and `Seqn`. The appropriate constants for use with the IIR reference state are as follows:

$h_0 = 209{\cdot}100043$ kJ/kg; $\qquad s_0 = 1{\cdot}094452$ kJ/kg.K

and for use with the ASHRAE reference state:

$h_0 = 59{\cdot}118225$ kJ/kg; $\qquad s_0 = 0{\cdot}291438$ kJ/kg.K.

b. Function `muliq`. The liquid viscosity equation given by ICI (1991) is used. This takes the form

$$\ln\mu = A + \frac{B}{T} + \frac{C}{T^2} + \frac{D}{T^3} \qquad \textbf{(11.1.35)}$$

where $A = -24{\cdot}4565$, $B = 19225{\cdot}29$, $C = -5{,}458{,}933$, $D = 5{\cdot}2714 \times 10^8$. Liquid viscosity, as given by equation (11.1.35), is in units of cP.

c. Function `kliq`. The liquid thermal conductivity equation given by ICI (1991) is used. This takes the form

$$k = k_0 + \sum_{n=1}^{4} (1 - T_r)^{\frac{n}{3}} \qquad \textbf{(11.1.36)}$$

where $k_0 = 0{\cdot}0460164$, $k_1 = 0{\cdot}0533949$, $k_2 = -0{\cdot}120467$, $k_3 = 0{\cdot}2424306$.

d. Function `sigmaf`. Chae et al. (1990) empirically derived the following equation for the surface tension of r134a:

$$\sigma = \sigma_0 T_{cr}^{1{\cdot}26} \qquad \textbf{(11.1.37a)}$$

where

$$T_{cr} \equiv \frac{T_c - T}{T_c} \qquad \textbf{(11.1.37b)}$$

and $\sigma_0 = 0{\cdot}0581$.

11.1.4. Class `reftype2`.

There exists a considerable body of data describing the properties of traditional refrigerants in Imperial units. Chan and Haselden (1981a, b, c) for instance have published a set of relationships and computer programmes to calculate the properties of refrigerants r11, r12, r13, r13B1, r14, r22, r113, r114 and r502, the coefficients for each of which is appropriate to the calculation of the refrigerant properties in Imperial units. Class *reftype2*, which is a subclass of class *refrigerant*, has been developed to handle such cases. The member functions to compute thermodynamic variables in this class take the input in SI units, convert it to Imperial units, perform the appropriate calculations, and convert the output back to SI units before returning. The member functions provided generally parallel those of class *refrigerant*. The expressions used by Chan and Haselden, and implemented in class *reftype2* do however show minor differences from those on which the base class is based. The reader is referred to Chan and Haselden (1981a) for details.

The functions for the transport properties are not redefined in class *reftype2*.

11.1.5. Class *r22*.

Class *R22* is a subclass of class *reftype2*. The thermodynamic properties of refrigerant r22 are calculated using member functions of class *reftype2*, while for the most part, the transport properties are calculated using the appropriate member functions of class *refrigerant*. Coefficients for the former set of equations may be found in Chan and Haselden (1981a), while coefficients for the latter set are provided by Jung and Radermacher (1991). The following exceptions are noted:

a. Functions `heqn` and `Seqn`. The appropriate constants for use with the IIR reference state are as follows:

h_0 = 300·648553 kJ/kg; s_0 = 0·635909 kJ/kg.K

and for use with the ASHRAE reference state:

h_0 = 233·21409 kJ/kg; s_0 = −0·188922 kJ/kg.K .

b. Function `muliq`. The following liquid viscosity equations given by ASHRAE (1973) are used. For $170 \le T < 310$:

$$\mu = 0{\cdot}001 \exp\left(A + \frac{B}{T}\right) \qquad \textbf{(11.1.38a)}$$

where A = −3·39554, B = 532·855. For $310 \le T < 360$:

$$\mu = 0{\cdot}001(A + BT + CT^2) \qquad \textbf{(11.1.38b)}$$

where A = −1·65108, B = 1·24147 × 10^{-2}, C = −2·09286 × 10^{-5}.

c. Function `Cpliq`. The following equation given by ASHRAE (1973) is used:

$$C_p = A + BT + CT^2 + DT^3 \qquad \textbf{(11.1.39)}$$

where, for $100 \le T < 590$,

A = 2·98190 × 10^{-1}, B = 1·00920 × 10^{-3}, C = 1·01607 × 10^{-6}, D = −1·67315 × 10^{-9},

and, for $590 \le T < 1500$,

A = 2·79649 × 10^{-1}, B = 1·59006 × 10^{-3}, C = −1·05288 × 10^{-6}, D = 2·54011 × 10^{-10}.

11.2. Modelling System Components.

There are considerable advantages to be gained by modelling the various components in a refrigearation cycle using a set of classes derived from a base class which represents the components as abstract machines which exchange energy with a working fluid by means of an as yet undefined thermodynamic process. An abstract base class *Component* has been developed for this purpose. This class declares the following member variables:

`Entering, Leaving` : Variables of struct *FluidState*, which store the fundamental thermodynamic properties (T, P, h, s, v and x, together with a variable of type *fluid::state*, describing the state of the fluid) of the fluid entering and leaving the component.

`superheat, subcooling` : Representing the degrees of subcooling or superheat (K) applied to the fluid entering or leaving the component, as appropriate; the user application will determine the interpretation to be placed on these variables.

`W`: Work (kW) done during the process. This will be positive if done by the component, negative if done on the component.

`Q` : Heat (kW) transferred during the process. This will be positive if transferred to the component, negative if transferred from the component.

`m` : Mass flow rate (kg/s) of fluid through the component.

`f` : A pointer to the working fluid; a variable of class *fluid*.

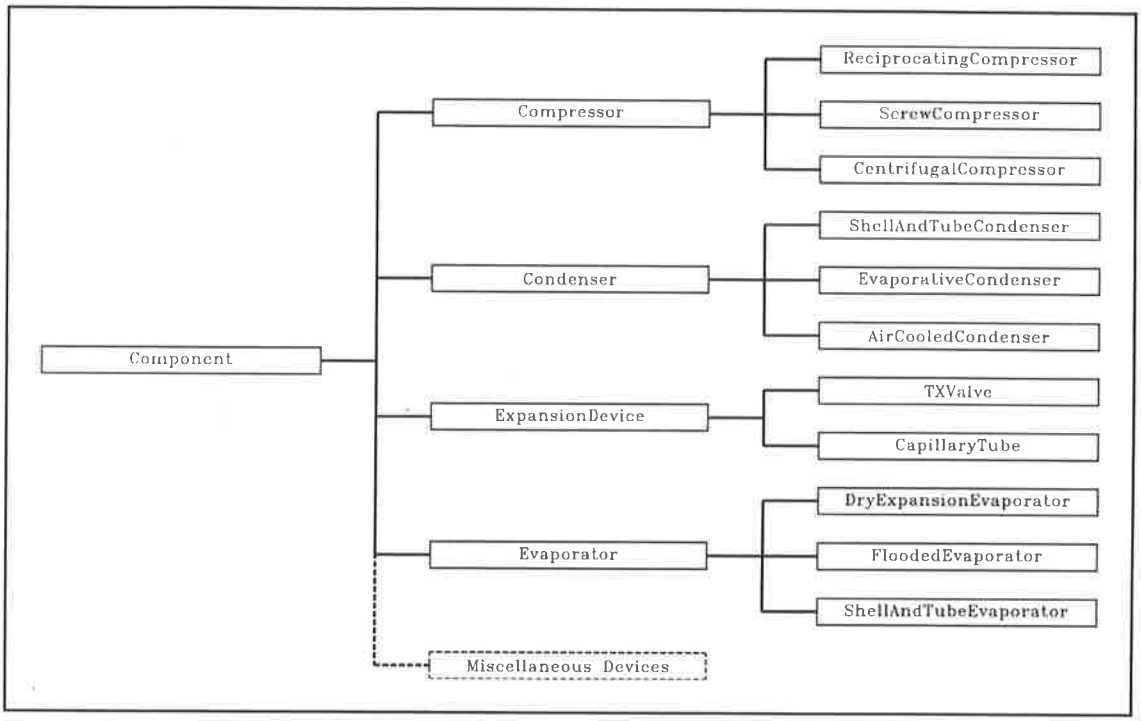


Figure 11.4. A hierarchy of classes representing components in a refrigeration cycle.

The class declares a normal complement of basic member functions, including constructors, destructors, and functions to access the member variables, including the various fields of the two variables of class *FluidState*. The following additional member functions are provided:

`UpdateEnteringState`, `UpdateLeavingState` : These update variables `Entering` and `Leaving` respectively to represent the current state of the variable of class *fluid* accessed by the pointer `f`. These are intended primarily for use by versions of `Process` (see below) defined in base classes.

`Process` : A pure **virtual** function which must be defined in a derived class. This function will take the variable of class *fluid* accessed by pointer `f` through some thermodynamic process, and update the member functions listed above accordingly. Note that there is no local copy of a variable of type *fluid*. All components in a cycle access a common copy of a variable of this type which belongs to the cycle, and which is modified by a sequence of calls of the form

```
X.Process ();
```

where x is an object belonging to some subclass of *Component*.

The class as defined is completely general, and may be used as a base for modelling the components in any thermodynamic cycle. Its use is not restricted to major plant items; it may just as well be used to represent the processes occurring in a length of pipe or a valve.

A possible hierarchy of classes derived from class *Component*, and suitable for representing the equipment items in a refrigeration cycle, are shown in figure 11.4. The set of subclasses derived directly from the base class are generic base classes representing equipment items of the various types. It is expected that each of these will implement a member function `Process` which will simulate the ideal thermodynamic process characteristic of the type. Thus, a function such as

```
virtual void Compressor::Process ();
```

will simulate isentropic compression between two working pressures, and will represent the default process for items of that generic type. By declaring these functions to be **virtual**, they may be overridden by functions of the same name in subclasses further down on the hierarchy, which may be expected to provide a more realistic simulation of the thermodynamic processes actually occurring. Note that the functionality provided by the base class is still available to the derived class, if it is explicitly addressed. Consider for instance a code fragment

```
virtual void CentrifugalCompressor::Process ()
{
  :
  Compressor::Process ();
  :
}
```

In this example the derived class (*CentrifugalCompressor*) explicitly calls a member function of the base class (*Compressor*) to make a first approximation of the behaviour of the working fluid during a compression, which may then be refined by the derived class.

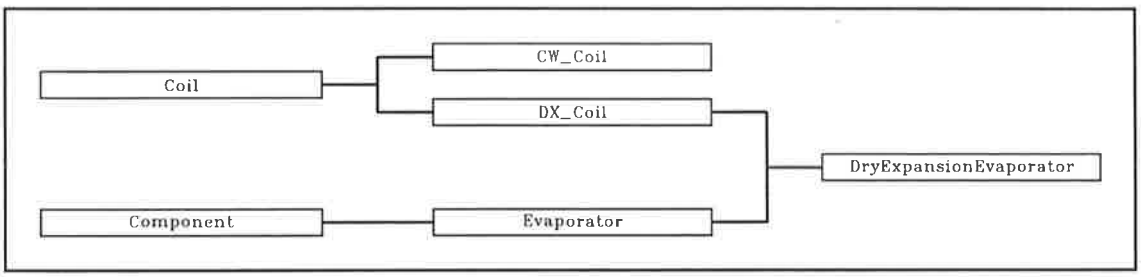


Figure 11.5. A proposed hierarchy for a class to model dry expansion evaporators.

In modelling the dry expansion evaporator, it is probably appropriate that the representative class should inherit ultimately from both classes *Coil* (described in chapters 6 and 7) and *Component*, using the C++ multiple inheritance mechanism. A proposed hierarchy is shown in figure 11.5. This provides a means for the evaporator to access the considerable facilities offered by the base classes. It also facilitates the use of this class as a link

between the air and refrigerant cycles in the air conditioning system model analogous to its function in an actual system.

Chapter 12. Operational Use of the Zebra Package.

The overall structure of the Zebra air conditioning simulation package was described in Chapter 2, while subsequent chapters have described in some detail the methodology used within the Zebra Kernel to model the various system components. The purpose of the present chapter is to describe a number of operational considerations associated with the use of the Zebra Kernel to simulate system performance for either an isolated load point, or a programmed sequence of loads. The specific aspects of the Zebra operational cycle to be described are:

i. The sequence of events involved in initiating a simulation session.
ii. The means by which the user, or a user-initiated task, interacts with a loaded Zebra process.
iii. The mechanisms provided for logging the results of a Zebra simulation run.

The Zebra Kernel has been developed for IBM-compatible computers running the DOS operating system, which is a simple single-tasking operating system. The following descriptions refer specifically to that version of the software. However, the code has been developed with the intention that it will subsequently be ported to the 32-bit Windows NT operating system, which provides facilities for concurrently running a number of cooperating tasks. This has significant consequences for the integration of the Zebra Kernel into a broader environment, and notes will be provided indicating the modifications which will need to be made to the Zebra software to enable it to communicate effectively with cooperating tasks.

12.1. Initiation of a Zebra Run.

When the Zebra Kernel is invoked, a sequence of steps is taken before the Zebra process can be regarded as loaded and able to accept input from a user, or from another user-initiated task. The major steps in the sequence are described below. The steps to be taken are determined by the command-line switches which are specified when the process is invoked.

12.1.1. Command-Line Switches.

A Zebra Kernel process is initiated from an interactive terminal session by typing a line of the following form in response to a keyboard prompt:

> `zk` [*options*]

in which *options* refers to a sequence of optional command-lines switches which control various aspects of programme operation. Alternatively, a Zebra Kernel process may be initiated by another process using for instance the `spawn` or `exec` function calls within a DOS environment, or the `WinExec` function call within a Windows environment. These functions also incorporate a mechanism for passing command-line switches to a process being initiated.

A Zebra command-line switch consists of a single-character code, prefixed by a hyphen, and possibly followed by sequence of characters comprising an argument to the switch the significance of which will be determined by the code. Thus, the Zebra Kernel could be invoked using the following command line

```
> zk -ishellbdg.zsp -rshellbdg.log
```

which specifies that the system specifications are to be read from a file `shellbdg.zsp`, with runtime logging directed to a second file `shellbdg.log`. The codes currently supported are[136]:

`-carg` Enables or disables logging of the separate coil components in a coil bank, when the Zebra Kernel is running in coil simulation mode. If *arg* is '+' or omitted, logging of the separate components will be enabled. If *arg* is '-', logging of the separate components will be disabled. Logging of separate components is disabled by default.

`-i`*filename* The argument *filename*, which is mandatory, specifies a source of textual data from which a set of system specifications will be read. Currently this must be a valid filename. However, the process of extracting a set of system specifications from a source is performed by an object of class *TextSource* (see section 2.4), and it will be recalled that the constructor for this class takes as an argument a reference to an object of the predefined class *istream*, which specifies the input stream from which data are to be extracted. In principle then, any suitably formatted source of textual data to which an object of class *istream* can be attached may be used as the input source with a minimal amount of extra support. In particular, the Windows Clipboard provides an efficient means of transferring data of this type between tasks running in the Windows environment. If this switch is omitted in an interactive environment, the user will be prompted to supply a file name; otherwise the switch must be specified.

`-l`*lines* The mandatory argument *lines* specifies the page length, in number of lines, for reports generated by the Zebra report generator[137]. If this switch is omitted, a default length of 60 lines is assumed.

`-m`*mode* The mandatory argument *mode* specifies the mode in which the Zebra Kernel is to run, and may take one of the following values:

---

136 The codes are case sensitive.

137 The Zebra report generator, as referred to in this and the following paragraphs, is an internally supported facility specific to the DOS environment. This will be described further in section 12.2 below, and is not to be confused with the external report generator shown in figure 2.1, which is projected for the Windows version of Zebra.

`c` : *Coil simulation* mode. A single coil bank is to be simulated.

`s` : *System simulation* mode. A complete system is to be simulated. This is the default mode.

`-o`*arg* The argument *arg* is a character string specifying the destination for reports generated by the Zebra report generator. If *"prn"* is specified reports are directed to the system printer. Otherwise, *arg* must specify a valid file name. The report destination specified by this argument may be overridden at run time, as will be described in section 12.2.

`-r`*arg* The argument *arg* is a character string which controls runtime logging. If "+" or "-" is specified, runtime logging will be *enabled* or *disabled* respectively. Otherwise, *arg* must specify a valid file name to which runtime log data may be directed. The issue of runtime logging will be taken up in further detail in section 12.3.

`-s` This switch takes no arguments, and suppresses interactive output to the screen.

`-t`*arg* Enables or disables logging of model time. If *arg* is "+" or omitted, logging of model time will be enabled. If *arg* is "-", logging of model time will be disabled. Logging of model time is enabled by default.

`-z`*filename* Argument *filename*, which is mandatory, specifies a metafile containing a sequence of commands which will in effect serve as a script to control operation of a loaded Zebra process. The metafile concept is discussed fully in section 12.2. If a metafile is specified via the command line its contents will be processed before the user is prompted to interact with the process. The metafile may in fact contain a record requesting termination of the Zebra process, thus denying the user further access to the process.

12.1.2. Coil Simulation Mode.

The purpose of the *coil simulation* mode is to provide a restricted environment in which the user may examine the response of a cooling coil bank to varying flow rates and entering conditions for the working fluids, in isolation from the rest of the system. Class *CoilTestEnvironment* provides a set of facilities to perform this function, which in essence may be looked upon as a subset of those available when Zebra is run in *system simulation* mode. An object of this class is created by invoking the constructor

```
CoilTestEnvironment::CoilTestEnvironment (istream& ifs,
                                          char* fn = "");
```

where the first argument specifies the input stream from which the coil specifications are to be read, and the second argument is a pointer to a string which may be used to store a description

of the source of the specifications (a file name, for instance). The constructor creates an object of class *TextSource* which it uses to access the records in the specification source, one at a time. In addition to a full specification of the coil bank configuration, the specification source must provide records which define the following:

a. The coil-on condition, specified by the dry-bulb temperature and either the wet-bulb temperature or the dew-point temperature.

b. The air flow rate, specified as a mass flow rate, a standard volume flow rate, or an actual volume flow rate.

c. The entering water temperature.

d. The water flow rate, specified as either a mass flow rate or a volume flow rate.

When the coil bank is constructed and the properties of the entering fluids set according to the supplied specifications, Zebra is loaded and ready to receive user input. Member function `CoilTestEnvironment::Process ()` invokes the Zebra command-line processor to receive and act on user input. The manner in which this is done is described in section 12.2.

12.1.3. Constructing the System Model.

The input specification file is opened within the `main` function of Zebra, and an object of class *SystemControl* (section 6.2.1) is created by invoking the appropriate constructor, which is defined as

```
SystemControl::SystemControl (istream& ifs,
                              char* fn = "");
```

Within the body of the constructor, an object of class *TextSource* is created to extract data on a line-by-line basis from the input stream referenced by argument `ifs`. The manner in which the system model will be created has already been described in some detail in section 2.4. The intention here is merely to provide a few additional notes to clarify this process within the context of the system model.

Having created an object of class *TextSource*, the constructor for class *SystemControl* initiates a loop to extract and process records until the specification source is exhausted. The input codes which are processed directly by this constructor are

`PROJECT_NAME` introduces a record specifying the project name (a null-terminated ASCII string). If no project name is specified, a null string is specified as the default.

`SYSTEM_ATMOSPHERIC_PRESSURE` introduces a record specifying the atmospheric pressure. If this record is omitted, standard atmospheric pressure (101·325 Pa) is assumed by default.

SYSTEM_OUTSIDE_AIR introduces a record specifying ambient air condition as a dry-bulb and wet-bulb temperature pair. By default, dbt = 32°C, wbt = 27°C.

SYSTEM_TIME introduces a record specifying the model time. This is an object of class *timec*, which stores model time to a resolution of one millisecond, and provides operators for writing time to, and reading it from a text stream in the format `hh:mm:ss.mmm`, where the second and millisecond fields may be omitted. A 24-hour clock is used. Model time defaults to 15:00 if this record is omitted.

DCONTROL_BEGIN introduces a record which initiates an input sequence for an object of class *DController* (see section 6.2.2). This record contains an identifier for the controller. This identifier must be unique. When this record is read, a new object of class *DController* is created, and control passed to the appropriate constructor, which takes as an argument a reference to the *TextSource* from which the specifications are being extracted. Control remains within this latter constructor, or within constructors or member functions which it invokes, until a matching record with the code CONTROL_END is read. Control then reverts to the calling function. The records contained between this record and the matching CONTROL_END record must be adequate to specify completely the air-handling unit associated with the controller, together with the associated duct systems, and the zones served by the unit. Sequences of records nested within this sequence may result in new objects being created, and control temporarily being transferred to the associated constructors, which in turn may invoke additional constructors. Thus, a sequence of records contained between a record having the code SA_CONNECTOR_BEGIN, and the matching record with code SA_CONNECTOR_END will be processed by the constructor for class *SA_Connector*, and may in turn invoke a constructor for a coil bank and its associated coils, a filter, a duct system, and several zones.

OCONTROL_BEGIN introduces a record which initiates an input sequence for an object of class *OController*. The above remarks concerning code DCONTROL_BEGIN are in general applicable here. The sequence of records contained between a record introduced by this code, and the matching CONTROL_END code must contain the specifications for all distributed units served by this outside-air unit.

The controller for class *SystemControl* may process multiple record sequences introduced by codes DCONTROL_BEGIN and OCONTROL_BEGIN. It is thus possible to configure a system which comprises several outside-air treatment units, and the distributed units they serve, together with a number of standalone distributed units.

Each of the major entities comprising a system model is identified by a unique integer code[138]. Class *SystemControl* maintains a set of associative arrays establishing a link between the

---

[138] Those entities involved are objects of classes *OController*, *DController*, *OA_Duct*, *RA_Duct*, *SA_Duct* and *Zone*.

indices and the entities to which they refer, together with member functions which provide client routines to access these arrays. These arrays are set up when the model is constructed. A similar method is used to access entities at the local level. Each section within a duct system is identified by a unique index which is stored as the key in an associative array, the corresponding section pointers comprising the associated values. The associative array itself is a member variable of class *Duct*.

12.2. User Interaction with Zebra.

Class *SystemControl* defines a member function

```
int SystemControl::CommandLineInterpreter (char* cmd,
                                           int (*cpr) (char, char**),
                                           char* cds);
```

This function implements the Zebra Command-Line Interpreter (**CLI**), which controls all user interaction with a *loaded* Zebra process. The arguments to the function have the following meanings:

`cmd`: A pointer to a null-terminated character string containing a command statement written in the Zebra Programming Language (**ZPL**). **ZPL** is an interpreted language, analogous in some respects to the Postscript language, but with far simpler syntax rules, and a much smaller vocabulary. The following description refers to Level 0 of the language and is of preliminary nature only. The range of functions supported by the language is expected to develop substantially as user experience with the package increases, and may well force a reappraisal of the syntactical rules[139].

A **ZPL** statement consists of one or more lexical tokens, each of which comprises one or more contiguous characters. Lexical tokens within a command are separated by white space. The first lexical token within a command is a *command introducer*, and consists of a single uppercase letter which determines the type of action to be initiated by the command. The remaining lexical tokens are of two types. If the lexical token begins with a hyphen, the token is interpreted as a *command modifier*[140], and qualifies

---

139 The commands supported have been implemented on a somewhat *ad hoc* basis to meet the needs of the present research project. This version of the software has recently been released internally for consulting work, and will in the near future be required to support projects involving the application of expert system principles and process integration to the design of air conditioning systems. These considerations will undoubtedly encourage a rationalization of the interactive functions supported by Zebra. For this reason, the discussion in this section should be regarded as indicative only of the type of interactive functions which may be supported by Zebra.

140 The leading hyphen is stripped from a command modifier before it is processed. In any references to command modifiers in the following, it will be assumed that this action has already been taken.

the action to be initiated by the command. All other lexical tokens are interpreted as *comments*, and ignored. **ZPL** commands are case sensitive. There is however no restriction on the length of a **ZPL** command. When Zebra is loaded in *system simulation mode*, the following command introducers are valid:

E : *(Edit).* A request to edit some aspect of the system model follows. The syntax of each command modifier conforms with the following format:

Type [Index] [Edit Strings]

where the entity to which the editing operation is to be applied is identified by a single letter code (*Type*) identifying the type of entity, followed if appropriate by an unsigned numeric code (*Index*) which identifies the particular instance of the type to which the operation is to be applied. The editing operations themselves are specified by an optional sequence of edit strings, as above. Thus, the **ZPL** command

```
E -Smd27.0w25.0t16:00 -Z5t24.0
```

will set the dry-bulb temperature of the outside air to 27°C, and its wet-bulb temperature to 25°C, while altering the model time to 16:00 and the thermostat setting for zone 5 to 24°C. *Type* may take one of the following values:

D : *(DController)* causes the editing operation(s) to be applied to a distributed unit controller, or one of the entities (fans, coils, etc) associated with it.

O : *(OController)* causes the editing operation(s) to be applied to an outside air unit controller, or one of the entities (fans, coils, etc) associated with it.

S : *(System)* causes the editing operation(s) to be applied to some property which applies to the entire system, such as time or outside air condition.

Z : *(Zone)* causes the editing operations to be applied to a specific zone, and may result for instance in thermostats and humidistats being activated or deactivated, or having their set-points changed, or in zone loads being altered.

It is not proposed to document here the edit strings which may be applied to each *Type* since these are being implemented on an *ad hoc* basis, as has been noted. In general, those aspects of the system model which may be altered by

transmitting edit commands to a *loaded* Zebra process will for the most part have a control function, and may be changed without altering the structure of the model. Other aspects of the model are perhaps better updated when the process is not loaded. An exception may be made where it is necessary to modify some aspect of the model structure to satisfy the requirements of a coordinating task, such as an expert system or an optimization code.

M: *(Metafile).* In the present context, a metafile is a formatted file, each record of which contains a command string which will perform some control action involving the loaded Zebra process. The record may contain a valid **ZPL** statement, in which case it will be passed to the Command-Line Interpreter for processing. Alternatively, it is possible to alter certain attributes of some of the entities of which the model is composed by respecifying data supplied originally by the specification input file, using the same codes and formats as used by the original source. In the case of system-wide attributes, the record specified will be identical to that in the original source. The codes supported are `SYSTEM_ATMOSPHERIC_PRESSURE`, `SYSTEM_OUTSIDE_AIR` and `SYSTEM_TIME` (see section 12.1.3). Thus, to alter the ambient dry-bulb temperature to 27°C, and the wet-bulb temperature to 25°C, the following record could be inserted into the metafile:

```
00003 27.0 25.0
```

To alter the attributes of other system entities in this manner, a suitable prefix must be added to the specification line, the format of which will be as follows:

x *Type Index Specification_Line*

The first character in the record will be a lowercase x, and *Specification_Line* contains the code for the attribute to be changed, and the new set of values, in the same format as used in the original source. Of the remaining fields, *Type* is a two-letter code indicating the type of entity for the which the attributes are to be updated, and *Index* is a numeric code identifying the particular instance of the entity type to be accessed. The code within the Type field may be one of the following:

Dc: *(DController).*
Oc: *(OController).*
Zo: *(Zone).*

A subset of the available specification codes are supported for use in this manner. As an example of this type of usage, the record

```
x Zo 5 16008 13.189 0.941
```

will alter the sensible load on zone 5 to 13·189 kW, and its sensible heat ratio to 0·941 (code 16008 prefixes an input record to specify the loads for a zone; see also the examples in section 2.4).

There is only one command modifier currently supported by this command introducer. The following ZPL statement

`M -i`*filename*

causes the specified file to be opened as a metafile, and its records read and processed. As in the case of the input specification file, it is envisaged that the metafile concept will in future be extended to handle input from media other than disc files. Note also that since a valid **ZPL** statement may be included as a record within a metafile, during the processing of one metafile, another metafile may be opened and its contents processed.

`Q:` *(Query).* A request to generate a report describing the current state of a particular subsystem follows. The following remarks apply specifically to a Zebra process running in a DOS environment, in which hard-copy and screen-based reports are generated on request using the Zebra report generator. As Zebra is ported to a Windows environment, it will be necessary to review and broaden the range of output options which are offered. Certainly, there will still be a need to generate hard-copy and screen-based reports in a manner analogous to that currently employed. It will also be desirable to transmit data to other processes, including in particular spreadsheet and database programmes, and graphics-based programmes, which may be used as intermediaries in the report generation process. It is envisaged that increased use will be made of the runtime logging feature (section 12.3) in future.

In current usage, the syntax of each command modifier conforms to the following format:

Entity_Type Index Report_Type [Destination][Filename]

In which *Entity_Type* specifies the type of entity for which the report is to be generated, and may take one of the following values:

> `D:` *(DController).* Generates a report for a subsystem based on a particular distributed controller, including the zones served by the corresponding unit.
>
> `O:` *(OController).* As for the preceding, but the report does not cover the distributed units served by the central outside-air unit.

Index specifies the particular entity for which the report is to be generated, and *Report_Type* specifies the type of report which will be generated. This latter

field must be present, and can take only one value, `f` *(full)*. The optional field *Destination* is a one-character code which specifies the destination for the report, and overrides the default destination[141]. This can take one of the following values:

`a`: *(Append)*. The file specified by the mandatory argument *Filename* is opened, and the current report appended to the existing contents of the file, if any.

`s`: *(Screen)* - *Filename* is not interpreted.

`p`: *(Print)* - *Filename* is not interpreted.

`w`: *(Write)*.This code operates in a similar manner to the *Append* code described above, the difference being that in this case any existing contents of the output file are truncated to the beginning of the file, before output of the current report commences.

Given an outside-air treatment plant, index 1, which serves five distributed units indexed 11 to 15, the following ZPL statement will generate and print reports on the current status of each of these units:

```
Q -O1fp -D11fp -D12fp -D13fp -D14fp -D15fp
```

`R`: *(Run)*. A request to initiate a simulation run follows. The syntax of the command modifier conforms to the following format:

Type Index

Type specifies the type of entity for which a simulation run is to be performed, and can take one of the following values:

`D`: *(DController)*. A moisture staircase iterative loop is performed for the distributed controller specified by *Index*.

`O`: *(OController)*. A simulation run is performed for the outside-air controller specified by *Index*, following which moisture staircase iterative loops are initiated for each distributed unit served by the outside-air unit.

---

[141] The default destination is the screen unless an alternative default destination has been specified using the command-line switch *-oarg* (see section 12.1.1). If an alternative destination is specified by the *[Destination][Filename]* arguments to the `Q` command, this new destination will be effective for this report only, the default destination being restored on completion.

`T` : *(Terminate).* This code terminates a Zebra run. No code modifiers are currently recognized for this code.

An equivalent set of command introducers is available when Zebra is loaded in *coil simulation mode.*

`cpr` : A pointer to a function (the *command processor*) which will process one **ZPL** command. Class ***SystemControl*** defines a member function

```
static int SystemControl::CommandProcessor (char code,
                                             char** strptr);
```

which performs this function for a Zebra process loaded in system simulation mode. The arguments to this function have the following meanings:

`code` is the command introducer.

`strptr` is a null-terminated array of pointers to the command modifiers, each of which is stored as a null-terminated string, stripped of its leading hyphen.

The values returned by this function are similar to those returned by the command-line interpreter (see below). The purpose of the command-line interpreter is to check the validity of the command introducer, and parse the command into its components, which are then passed to the command processor. The command-line interpreter generates an error return if the command introducer is not valid, or if the command processor signals an error. A command processor may choose to signal an error if it receives a syntactically incorrect command modifier, or a request to perform an operation on a non-existent entity. Alternatively, it may choose simply to ignore such command modifiers, and continue to process any additional command modifiers within the command.

The command processor for a Zebra process loaded in coil simulation mode is a member function of class ***CoilTestEnvironment.***

`cds` : A null-terminated character string containing the valid command introducers. Class ***SystemControl*** defines a member variable

```
static char* SystemControl::cds = {'E', 'M', 'Q', 'R', 'T', '\0'};
```

which fulfils this function for a Zebra process loaded in system simulation mode.

The value returned by the function may take one of the following values:

0 : Normal completion.
1 : A request has been received to a terminate the process.
-1 : An error has been detected. This indicates that an invalid command introducer has been specified, or that an error has been returned by the command processor.

Within the current DOS-hosted version of Zebra, once a process has been loaded, a loop is initiated to accept lines of input from the keyboard, and pass them to the command-line processor for further action. The loop terminates when the `T` command introducer is received. The Windows version of the programme is expected to operate in a similar manner, except that rather than receiving command-line input from the keyboard, it will receive it from one or more cooperating processes using the Dynamic Data Exchange (DDE) mechanism.

12.3. Logging of Runtime Data.

Class *LogFile* provides a facility which is used by Zebra for logging system specifications and runtime data. An object of class *LogFile* is created by invoking the constructor

```
LogFile::LogFile (char* fn = "\0",
                  boolean ts = false,
                  char* extn = 0);
```

where `fn` specifies the name of the file to which logged data will be written, and `ts` determines whether model time is to be logged. The final argument (`extn`) specifies an optional filename extension, which will override one supplied with the file name. If the file name is omitted, a temporary file is created, and deleted when the object is deleted. This default behaviour for the class was selected with future Windows applications in mind, in which it is envisaged that coordinating classes will interact extensively with the log file for a *loaded* Zebra process. In that case, the log file would become largely an intermediary between the model, and a target data store, such as a data base or a spreadsheet programme. In current usage, it is customary for the user to log data to a named file, which can be perused after the run. If the file name specified refers to an existing file, any new records are appended at the end of that file.

Data are written to the log file as formatted records, terminated with a newline character, and structured as follows:

S|T|N Spec_No|Run_No Time [Model_Time] [Data]

where the fields have the following significance:

1. The record is prefixed by an uppercase letter indicating the type of record to be written. Prefix *S* indicates that the record is a *specification record*, and contains details of the system configuration. Prefixes *N* and *T* indicate that it is a *data record*, and contains data describing the outcome of a simulation run. Prefix *T* indicates that logging of model time is enabled, and *N* that it is disabled. It is only meaningful to record model times when performing a sequence of simulations with load varying over time.

2. For specification records, *Spec_No* is the specification number, which is normally incremented when a new system configuration is loaded, or when the system configuration is altered. This may happen for instance, when an expert system is

searching for an optimal coil configuration. For data records, *Run_No* is the run number, which is normally incremented when the loads are changed, or some aspect of the system configuration is altered. For time-based simulations, an increment in run number normally corresponds to a change in model time. Both of these indices begin at 1, and contiguity is maintained when records are appended to an existing log file.

3. *Time* is the time the record is written, in MS-DOS internal format.

4. The model time is written only to records prefixed with *T*.

5. The preceding fields are all *prefix* fields. The process using the log file (Zebra in this case) may write runtime data, as appropriate, into the remaining fields of the record. No restriction is placed on the length of the record.

Three member functions are provided to facilitate write access to the log file:

```
ostream& LogFile::SpecificationRecord ();
ostream& LogFile::DataRecord ();
void LogFile::Write ();
```

The first two functions write an appropriate set of values to the prefix fields of the record, and return a reference to the output stream. Any further output will follow the prefix fields. The final function will write the current record to file. Note that since both types of record are written to the same file, one record should be written to its final destination before output to another is begun.

The current version of Zebra does not perform any read operations on the log file. However, a class *ReadLog* has been developed for this purpose. This class is similar in concept to class *TextSource*, which was described in section 2.4.

Chapter 13. Case Studies and Examples.

The preceding chapters have described a methodology for modelling the steady-state performance of all-air systems employing chilled water as the cooling medium. The ZEBRA software package has now been in use for several years, both in its present form and in the form of earlier versions based on essentially the same principles. During that time it has been used as a design and research tool for a number of conceptual and actual design studies. The results of a number of these will be used in the following to demonstrate some of the capabilities of the modelling techniques developed, to explain the basis of some of the successful design concepts which have emerged through experience, to expose some of the common misconceptions of air conditioning designers, and to provide some indication as to how the software may be applied within the context of a design methodology. The emphasis will however be very much on the first of these objectives.

A considerable body of experience has now been gained in the use of ZEBRA as a design tool. A number of studies are now being undertaken to condense this experience into a formal design methodology. The approaches taken in these studies are various, although broadly speaking all have a common objective. In addition, the modelling methodology described in the present thesis is a fundamental component of each of these latter studies. The examples presented in this chapter have been selected primarily to demonstrate the use of the package in a number of typical working situations.

As noted in the previous chapter, ZEBRA may be run in one of two modes. *Coil simulation mode* provides the user with an opportunity to examine the response of a coil bank to variations in the quantities and properties of the working fluids entering the coil. The first part of the chapter demonstrates the manner in which ZEBRA is able to model the thermodynamic processes within a composite coil bank. This both illustrates some of the concepts developed in chapter 8 and clarifies and quantifies the response of a simple coil to variations in the flow rates of the working fluids.

The examples and case studies presented in the remainder of the chapter demonstrate the use of ZEBRA in *system simulation mode*. Three examples are considered:

i. A lecture theatre at the National University of Singapore. This example is based on a retrofit for a lecture theatre in a tropical climate. The direct solar load on the conditioned space is negligible, and occupancy levels vary over a particularly wide range. These factors, together with the high outdoor air humidity levels characteristic of the climatic region in which the building is located, make it a particularly demanding application. The study was completed some years ago (1991), and the design solution selected for that project is not indicative of our current understanding of how the methodology may best be applied in practice. The example is, however, of more than historical interest. The absence of direct solar load, together with the use of the load locus, meant that the performance of the system over its entire operating range could be represented graphically as a function of outdoor air conditions and

occupancy. A custom-built version of the ZEBRA programme was devised to facilitate the simulation process.

ii. A multistorey office block. The example selected is based on a conceptual study (Marshallsay et al., 1993) which compared the energy consumption of various types of air conditioning system serving a multistorey office building when the building is located consecutively in four different geographic locations which are representative of four distinct climate types. Twodifferent ventilation rates are considered in each location. In the present study, the building which formed the basis for the earlier study is used as a test-bed for an in-depth examination of a number of solutions for a common design problem. While the range of climates considered here is less than in the original study, the opportunity will be taken to broaden the range of operating parameters considered.

iii. An apartment block in Bangkok. This example is of interest for a number of reasons. It illustrates the most recent developments in the HDP concept and its ability to achieve exceptional efficiency and psychrometric performance while at the same time satisfying severe commercial constraints. Many of these developments have been made possible by the simulation capability provided by the ZEBRA programme. Unlike the preceding cases, the plant considered in this example is required to operate for 24 hours each day in a climate which offers considerably more diversity than do locations closer to the Equator. In addition, the operational requirements peculiar to specialized areas such as "party" apartments and a lounge bar pose particularly challenging design problems.

13.1. Use of Zebra in Coil Simulation Mode.

In Chapter 7 were described a set of algorithms which may be used to simulate the behaviour of a simple coil given the flow rate and thermodynamic state of the working fluids entering the coil. The methodology was extended in chapter 8 to enable composite coils to be simulated using a suitably structured set of simple coils as the component elements. In the present section a number of examples are presented which will demonstrate the concepts developed in those chapters.

It is instructive in the first instance to consider the response of a simple coil to variations in the flow rates of the working fluids. The test results are based on a half-circuited four-row coil with a fin density of 6 fins per inch which presents a face area 914·4 mm high (24 tubes) by 1000 mm wide. The chilled water temperature at entry to the coil in each case is 6·5 °C.

In figure 13.1 the coil-off condition has been plotted on a psychrometric chart as a function of chilled water velocity. For the upper curve the air entering the coil has a dry-bulb temperature of 36 °C and a dew-point temperature of 25 °C; for the lower curve, the entering condition is 36 °C dry-bulb and 20 °C dew-point. In both cases, the actual air face velocity is 1·5 m/s. It is noteworthy that at very low coolant velocities the cooling is almost entirely sensible. Only when the coolant velocity becomes high enough to cool the air almost to its dew-point

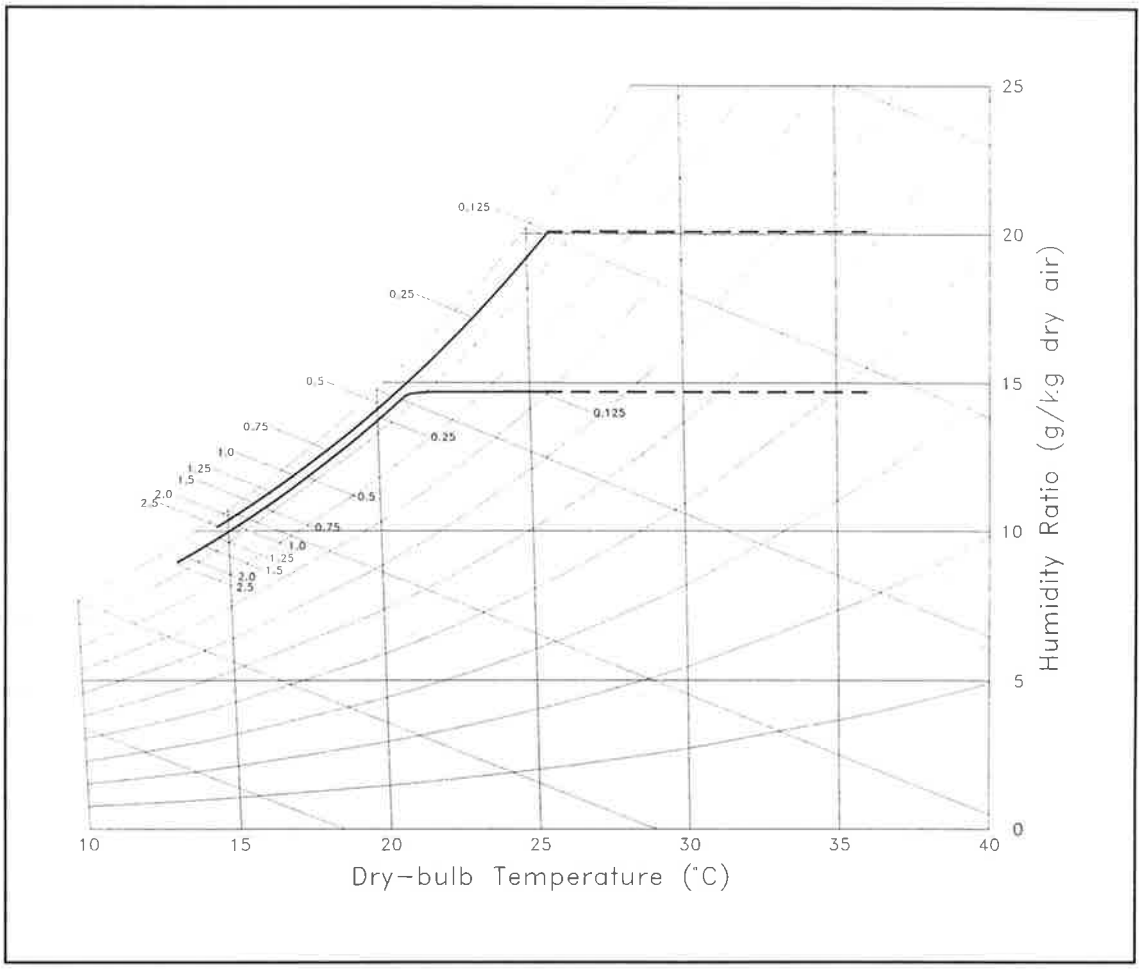


Figure 13.1. Coil-off condition for a simple coil (refer text) as a function of chilled water velocity.

temperature does any significant removal of latent heat occur. At higher coolant velocities, the coil-off point describes a locus which closely approximates a line of constant relative humidity. On the basis of the two cases shown, it appears that the exact position of this locus is determined by the humidity ratio of the air entering the coil, and can be expected to more closely approach the saturation line for more humid entering conditions. Further simulations need to be performed before this relationship can be quantified.

Further details of coil performance are contained in Appendix A, plots (A.1.a)-(A.1.h). In all cases, the entering air condition for the coil is 36°C dry-bulb, 25°C dew-point. In plots (A.1.a)-(A.1.g), various operating parameters are plotted as a function of chilled water velocity over the range from 0·125 to 2·5 m/s. The parameters plotted are:

a. Latent, sensible and total cooling capacity (kW) for an air face velocity of 1·5 m/s.
b. Sensible heat ratio.
c. Total capacity (kW).
d. Leaving air dry-bulb temperature (°C).
e. Water temperature rise (°C).

f. Surface wetness fraction, defined as the ratio of wet surface area to total surface area.
g. Water pressure drop (kPa) for an air face velocity of 1·5 m/s.

In plots (A.1.b)-(A.1.f), the relevant operating parameter is plotted for air face velocities of 0·5, 1·0, 1·5, 2·0 and 2·5 m/s. The water pressure drop[142] is virtually independent of air face velocity. Note that the information on coil capacity shown in plot (A.1.a) is also contained, in equivalent form, in plots (A.1.b) and (A.1.c). The remaining plot (A.1.h) shows air pressure drop as a function of air face velocity over the range from 0·25 to 2·5 m/s for a water velocity of 1·5 m/s; this parameter is virtually independent of water velocity.

The plots shown provide a fairly complete description of the coil performance *for one set* of entry conditions for the working fluids. In spite of this restriction, it is possible to draw some comparisons of fairly general relevance from a comparison of the various plots. These plots, and in particular (A.1.b) and (A.1.f), clearly illustrate the physical basis of the LFV/HCV method and enable general guidelines for its application to be derived. As shown in plot (A.1.b), dehumidification performance improves with decreasing face velocity, and at the lowest face velocity shown (0·5 m/s), the coil will be fully saturated for all but the very lowest chilled water velocities (plot (A.1.f)). However, from the point of view of dehumidification performance, combining high coolant velocity with an extremely low face velocity results in a redundancy. Referring again to the lowest face velocity shown (0·5 m/s, plot (A.1.b)), it will be seen that sensible heat ratio of the coil becomes virtually independent of chilled water velocity if this latter parameter exceeds approximately 0·5 m/s. The benefits of high coolant velocity become more apparent as the air face velocity is increased, but even at the highest face velocity considered, 2·5 m/s which is typical of conventional practice, use of coolant velocities in excess of about 1·5 m/s confers little benefit in terms of dehumidification. There are other reasons for limiting coolant velocities to moderate values. As shown by figure 13.1, and plots (A.1.c) and (A.1.d), coil capacity (both sensible and latent) increases monotonically with coolant velocity, but the rate of increase falls with increasing water velocity. Meanwhile, water pressure drop increases as the square of water velocity (plot A.1.g). Thus, increasing the coolant velocity from 1·5 to 2·5 m/s for an air face velocity of 2·5 m/s results in an increase in total capacity of about 10%, but requires a five-fold increase in pumping power, while substantially increasing wear and tear inside the coil tubes. At the same time, water temperature rise will decrease, possibly to a value which is insufficient to ensure efficient chiller operation[143].

A more extensive and systematic study is required to establish definitive design guidelines. However, on the basis of the author's experience, a chilled water velocity of about 1·5 m/s at peak, coupled with an air face velocity of about the same value probably represents a reasonable design target.

---

142 The near-discontinuity in this curve is attributable to reverse transition of the flow in the tube bends at low Reynolds number, a phenomenon which is described in section 7.6.3.

143 Bearing in mind that water temperature rise will increase at part load, a target peak load design value of about 5 °C is perhaps appropriate.

An additional series of simulations has been undertaken to demonstrate the performance of four composite coil banks, circuited according to the methods described in Chapter 8. The results of these simulations are reproduced in a further series of plots in Appendix A. The test data are presented for a single air face velocity of 1·5 m/s in each case, and in each case, the air entering the coil bank has a dry-bulb temperature of 36°C and a dew-point temperature of 25°C. The cases considered are described below. Of special interest is the distribution of cooling capacity (total, sensible and latent) between the coil components in the various cases.

1. **Case 1.** (Appendix A.2). The arrangement in this case comprises two coils with the chilled water flow to each circuited in parallel, as shown in figure 8.1. The coil dimensions are as follows[144]:

Coil height (L_f) : 914·4 mm (N_t = 24 tubes/row)
Coil width (L_t) : 1000 mm
Fin density (N_f) : 6·18 fins/inch

Coil[145]	N_r	N_{pc}	N_c	A_o
1	2	12	4	28·682 m²
2	4	12	8	57·365 m²

The functional relationship between sensible heat ratio and chilled water velocity (plot (A.2.d)) closely follows that for the single component coil considered previously. This is also the case for the following two cases (2 and 3). A comparison of the distributions of the sensible and latent cooling components between the various coil components in each of the different circuiting arrangements however reveals the differences in the mechanisms operating. Coil 2 (the downstream component) makes the major contribution to the *total* cooling capacity over the entire chilled water velocity range simulated, although the difference is only marginal at the upper end of the range (2·5 m/s). This difference is at least partially attributable to the fact that the cooling surface of coil 2 is double that of coil 1, a point which should be borne in mind in interpreting both this and the next example. For the *sensible* cooling however, it is coil 1 which predominates at all but the very lowest chilled water velocities (~0·125 m/s). The contribution of coil 2 to the *latent* cooling dominates over the entire velocity range, in both an absolute sense (plot (A.2.c)) and a relative sense (plot (A.2.d)). This occurs in spite of the fact that in the circuiting arrangement shown, both coil components are fed with chilled water at the same temperature, and hence the dehumidification potential of coil 1 should exceed that of coil 2. This mode of behaviour is in line with that shown in figure 13.1, and together they demonstrate

---

144 The nomenclature follows that of section 7.1.

145 Coils are numbered in order from the front of the coil bank with respect to the direction of air flow.

clearly the fundamental physical fact that the air entering the coil will undergo sensible cooling until it is almost saturated before any latent cooling occurs[146].

2. **Case 2.** (Appendix A.3). The arrangement simulated in this case is that shown in figure 8.2, in which two coil components are circuited in a counterflow arrangement with respect to the chilled water flow. The coil dimensions are as follows:

Coil height (L_f)	:	762·0 mm (N_t = 20 tubes/row)
Coil width (L_t)	:	1000 mm
Fin density (N_f)	:	6·18 fins/inch

Coil	N_r	N_{pc}	N_c	A_o
1	2	10	4	23·902 m²
2	4	10	8	43·908 m²

The solution for the thermodynamic performance of the coil bank in this example has been found by solving equation (8.4.1) for the single intermediate manifold. While the ratio of the surface areas of the two coils is the same as in the preceding case, in this case coil 1 recieves water which has already passed through coil 2. The water entering coil 1 is thus degraded in terms of its potential for both dehumidification and sensible cooling when compared with the preceding case. Consequently, the dominance of coil 2 in its contribution to total and latent cooling is even more marked than in the preceding case, although the contributions of the two coils to the sensible cooling are fairly closely matched over much of the range simulated.

3. **Case 3.** (Appendix A.4). This example is an extension of that considered above, in that the coil bank is composed of *three* identical coil components circuited in series in a counterflow arrangement. The dimensions of *each* coil component are as follows:

Coil height (L_f)	:	914·4 mm (N_t = 24 tubes/row)
Coil width (L_t)	:	1000 mm
Fin density (N_f)	:	6·18 fins/inch
Number of rows (N_r)	:	2
Passes/circuit (N_{pc})	:	4
Circuits (N_c)	:	12
Surface Area (A_o)	:	26·345 m²

---

146 Note that the analysis and the argument presented here assume one-dimensional flow of the air in the coil passages. As demonstrated in Chapter 7 the flow in the region of the tube/fin intersections is strongly three-dimensional. Physical evidence derived from observations of many coils reveals that some condensation does in fact occur in the leading rows of the coil and can, if the air flow velocity is sufficiently low, occur even at the leading edges of the fins. Nevertheless the one-dimensional air flow model is valid in that the predicted mixed state of the air leaving the first two rows of a six row coil, which is modelled as a combination of a two row and a four row coil in series, agrees with the experimentally determined state of the air leaving a two row coil in isolation (Sekhar, 1990).

The coil bank in this example has two intermediate manifolds, and the thermodynamic performance has consequently been found by solving equation (8.4.2). The general observations made in relation to case 2 apply here also, although since in the present case the coil components have equal surface areas, a more equitable assessment of the contribution of the various coil components to the sensible, latent and total cooling capacity of the coil bank is possible. The configuration, it should be noted, is that of a single six-row half-circuited coil. The results of the simulation show a steady decrease in the contribution of the coil components to the total and sensible cooling capacity of the coil. The trend is the same for the latent cooling capacity at high coolant velocities, although it is partially reversed at lower coolant velocities. The relative contribution of the various coil components to the cooling capacity is as follows:

Chilled Water Velocity (m/s)	**0·5**	**1·0**	**1·5**	**2·0**	**2·5**
Total Capacity					
Coil 1 (rows 1 and 2)	0·385	0·456	0·488	0·506	0·513
Coil 2 (rows 3 and 4)	0·332	0·321	0·314	0·309	0·312
Coil 3 (rows 5 and 6)	0·283	0·223	0·198	0·185	0·175
Sensible Capacity					
Coil 1 (rows 1 and 2)	0·514	0·529	0·542	0·551	0·554
Coil 2 (rows 3 and 4)	0·276	0·283	0·284	0·283	0·288
Coil 3 (rows 5 and 6)	0·210	0·188	0·174	0·166	0·159
Latent Capacity					
Coil 1 (rows 1 and 2)	0·261	0·393	0·441	0·468	0·478
Coil 2 (rows 3 and 4)	0·386	0·355	0·340	0·332	0·334
Coil 3 (rows 5 and 6)	0·353	0·252	0·219	0·201	0·188

4. **Case 4.** (Appendix A.5). This case provides an example of a coil bank subject to a changeover involving deactivation of circuits. The example chosen is that shown in figures 8.3 and 8.4; a three-row coil 24 tubes high, in which four circuits have been deactivated from the upstream two rows of the coil. The nomenclature used to label the various coil components is as shown in figure 13.2, and the dimensions of the coil are as follows:

Coil width (L_t) : 1000 mm
Fin density (N_f) : 6·18 fins/inch

Coil	L_f	N_t	N_r	N_{pc}	N_c	A_o
Full	914·4 mm	24	3	4	18	39·517 m²
1A	609·6 mm	16	2	4	8	17·563 m²
2A	609·6 mm	16	1	4	4	8·781 m²
2B	304·8 mm	8	1	4	2	4·391 m²

The results of the simulation are shown in Appendix A.5 for both the full coil and the reduced coil, and the plots contained therein clearly show the manner in which the changeover mechanism acts to improve the dehumidification performance of a coil under part load. Consider the case of a coil required to provide 20 kW of sensible cooling. From plot (A.5.b) it is seen that the chilled water velocity required to provide this amount of sensible cooling using the entire coil is 0·32 m/s. After deactivating four circuits as shown, the chilled water velocity in the remaining active circuits increases to 1·6 m/s. Simultaneously, the sensible heat ratio (plot (A.5.d)) decreases from 0·6 to 0·52.

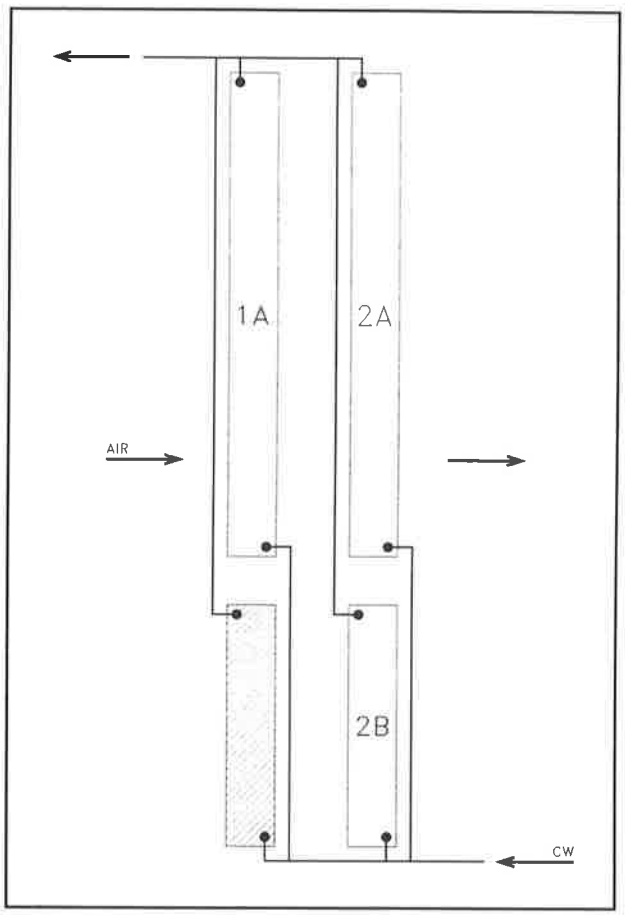


Figure 13.2. Nomenclature for coil components used in case 4.

The examples shown in the foregoing have been selected to illustrate the use of ZEBRA in coil simulation mode to explore the manner in which the performance of a coil will vary across the range of loads which it will be required to offset. In addition to providing an overall picture of the coil performance it is possible, by judicious segmenting of the coil into a number of component parts, to investigate the contribution which various rows or groups of rows make towards offsetting the latent and sensible loads inposed on the coil. The insights provided by case 4 (Appendix A.5) are especially useful. A vital step in designing a staging strategy (see section 6.5.4) is that of selecting the water velocities at which changeover will occur with a sufficient degree of overlap on rising and falling load. Failure to provide the necessary overlap will result in unstable operation. Use of plots such as these are an invaluable aid in selecting the required changeover points.

13.2. Case 1 - A Lecture Theatre in Singapore.

The brief in this case was to design and commission a replacement air handling unit for a 150 seat lecture theatre located in monsoonal Asia. The occupancy of the lecture theatre varies significantly from hour to hour and ranges from maximum capacity down to a hand-full of students studying individually or in small groups. The target zone temperature was 24°C, and the aim of the design exercise was to produce a system which would maintain the relative humidity within the conditioned space at or below 60% over its entire operating range, without the need for overcooling and reheat.

The system was designed strictly in accordance with LFV/HCV principles. The design also represents the earliest implementation of an HDP system. At the time the design was prepared (1991), the full potential and flexibility of the HDP system had not been fully realized and therefore this case study may be seen as having largely historical significance. Research undertaken since 1991 has greatly augmented our understanding of how HDP concepts may most effectively be applied in a range of situations. A number of HDP applications will be described in the case studies which follow this one.

The real interest of this example lies in the fact that, due to a combination of circumstances, it was possible to explore the functional dependence between system performance and its operating parameters within a far simpler framework than is normally possible. This permits the major indices of system performance across the entire operating range to be displayed using a unique graphical representation analogous to an "engine map", from which valuable insights into the behaviour of the system can be derived.

In general, the cooling loads imposed upon a conditioned space are determined by the nonlinear interaction of a large number of variables. Two key simplifications are made in the analysis associated with the present study:

i. The conditioned space has no glazed external surfaces, and solar load can consequently be ignored.

ii. Historical effects associated with the thermal inertia of the building fabric can be neglected. The validity of this assumption is open to question, and in general the ignoring of thermal inertia can significantly reduce the reliability of load calculations (von Thun and Witte, 1991). The practice is acceptable in the present instance since the diurnal swing characteristic of tropical climates is small, and the ambient temperature may be expected to vary smoothly in the course of the day (Sham, 1980). In addition, for this particular application the plant does not operate at night, and night-time ambient temperatures are close to the required zone operating temperature so the morning transient "pull-down" load is small.

If it is further assumed that the lighting and equipment loads remain constant, the zone loads become a function of two parameters:

i. Number of occupants within the zone.

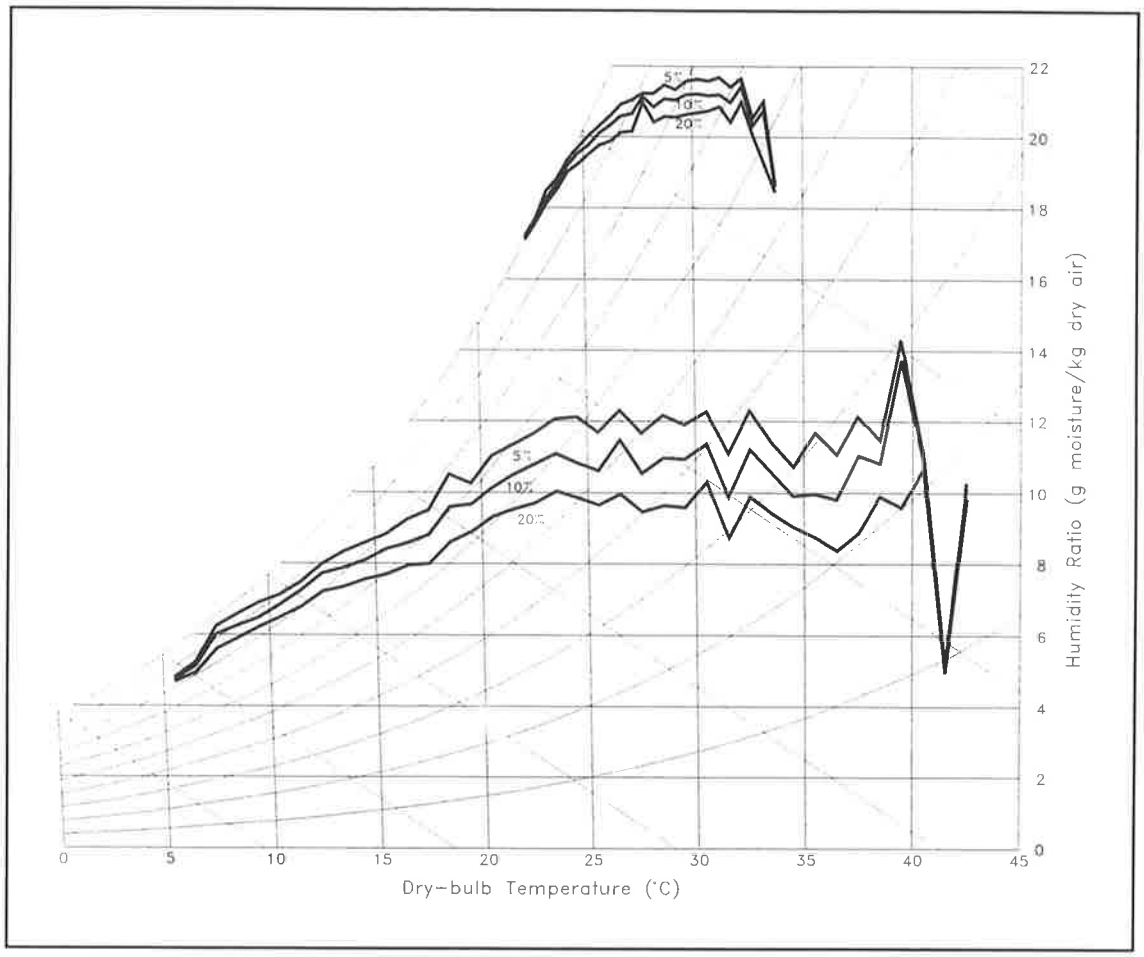


Figure 13.3. Design loci for 5%, 10% and 20% levels for Adelaide (lower set of curves) and Singapore (upper set of curves).

ii. Ambient dry- and wet-bulb temperatures.

An appropriate set of design ambient conditions may be derived by reference to the *design locus* for Singapore (Luxton et al., 1989). A point on the design locus is located by the dry-bulb temperature, and by the simultaneous value of wet-bulb temperature which will not be exceeded with an expectancy of E. In the present case, the design locus for $E = 10\%$ has been used. Daytime design loci for Adelaide and Singapore[147] are shown in figure 13.3. The locus for Adelaide was constructed using statistics covering the period from 0900 to 1800, that for Singapore using statistics covering the period from 0800 to 1700.

It needs to be emphasized that the design locus, and hence the performance maps to be derived from it later in this section, are *design* tools. The locus permits the designer to locate the peak design conditions and those part-load conditions which will be critical to the satisfactory

---

147 The design locus for Adelaide was constructed using statistics provided by the CSIRO based on readings obtained by the Bureau of Meteorology. The statistics upon which the Singapore locus is based were supplied by Assoc. Prof. T.Y. Bong of the National University of Singapore.

dehumidification performance of the plant. The design locus does not completely define the operating range of the plant; by definition, a system will operate at points below the 10% design locus for 90% of its working life.

13.2.1. System Configuration.

The system installed is a Variable Air Volume (VAV) system having separate outdoor-air and return-air coils circuited in series, with the chilled water fed to the outdoor-air coil first. The water entering the outdoor-air coil thus has a higher dehumidification potential than that entering the return-air coil. This is an appropriate design decision for a humid tropical climate. The specifications for the coils used are as follows:

		Outdoor Air	Return Air
Coil Width	L_t	772·0	1397·0
Coil Height	L_f	914·4	914·4
Tubes/Row	N_t	24	24
Rows	N_r	3	3
Passes/Circuit	N_{pc}	4	4
Circuits	N_c	18	18
Surface Area (m²)	A_o	30·507	55·205
Fin Density (fins/inch)	N_f	6·18	6·18

The chilled water flow to the two coils is modulated using a single valve, so that the total water flow through both coils remains the same. The circuiting arrangement is as shown in figure 13.4. Falling loads are accommodated by selectively deactivating circuits from the first two rows of the *return air coil only*. This is achieved using simple on/off valves in conjunction with a split outlet manifold for the return air coil. Water pressure drop is used as a surrogate for water velocity to trigger the changeovers although in the following, all references are to the corresponding water velocity. Details of the staging employed are as follows:

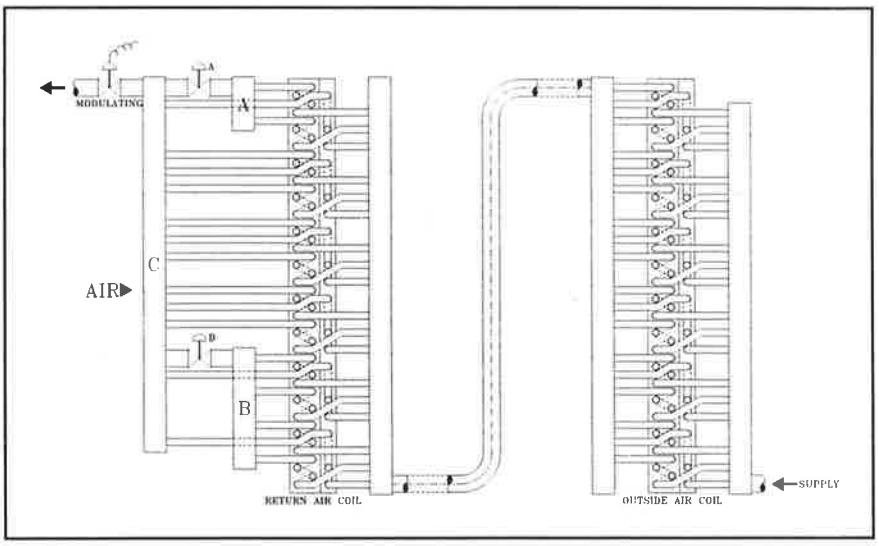


Figure 13.4. Coil circuiting diagram for lecture theatre.

Stage	1	2	3
Circuits Active - O/A Coil	18	18	18
Circuits Active - R/A Coil	18	14	12
Upper Breakpoint (m/s)	-	2·1	2·1
Lower Breakpoint (m/s)	0·8	1·0	-
Valve A	Open	Closed	Closed
Valve B	Open	Open	Closed

Outdoor air is 600 L/s (4 L/s per person) at peak load, when the lecture theatre has 150 occupants, and is modulated in proportion to the total air supplied to the space. This is acceptable practice in the current application, since it is guaranteed that adequate ventilation (greater than the Singapore standard of 2·5 L/s per person) will be supplied to the space at all times. The supply air temperature was 14·5°C.

13.2.2. System Performance.

System performance was evaluated over a load matrix defined by the design locus and the occupancy of the space, the nodes being defined by the following coordinates:

Occupants: 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 125, 150.

Ambient dbt (°C)	25	26	27	28	29	30	31	32
Ambient wbt (°C)	24·5	25·2	25·5	26	26·3	26·5	26·7	27

Room sensible heat ratio varied across the load matrix from a value of 0·624 with the space occupied by 150 students under ambient conditions of 25°db, 24·5°wb, up to 0·973 with the space occupied by 5 students under ambient conditions of 32°db, 27°wb.

The simulations were performed using an earlier version of the ZEBRA code described in the preceding chapters. While lacking the generality of the current version of the code, this version did possess specific features which enabled it to process a sequence of load conditions covering the load matrix described above, and also to locate the intermediate points in the matrix at which a changeover would occur[148]. For each ambient condition in the above matrix, the following steps were executed:

---

148 While the current version of the Zebra Kernel does implement the ability to process an arbitrary sequence of load conditions, and also provides a fundamental set of operations to support the staging mechanism (see section 6.5.4), the ability to locate the changeover points iteratively has been omitted from the Zebra Kernel. In keeping with the philosophy underlying the current version of the software such functionality should, if required, be implemented as a component of a coordinating process, rather than as an integral component of the Zebra Kernel itself. It should be noted that the staging mechanism currently supported is somewhat broader in scope than that envisaged when the design study considered here was performed.

1. A sequence of records was read from from a file, each record specifying the occupancy (p) of the lecture theatre, and a load vector ($\overline{L}$) containing the following components:

   a. Sensible load (kW),
   b. Sensible heat ratio,
   c. Supply air duct heat gain (°C),
   d. Return air duct heat gain (°C),
   e. Ventilation rate (L/s),

   each calculated on the basis of the operative ambient conditions and occupancy level[149]. The records were read in *ascending* order of occupancy level for simulation sequences describing system behaviour on *rising* loads, and in *descending* order where it was desired to simulate *falling* loads.

2. For the load sequence, a functional relationship was established between occupancy and load by fitting a cubic spline (Press et al., 1988) to the various components of the load vector $\overline{L}$, with occupancy level as the independent variable.

3. The nodes specified by the sequence of occupancy levels were traversed. For the first node in the sequence, the appropriate stage was determined using methods broadly similar to those described in section 6.5.4. For each succeeding node i, system performance was calculated, and chilled water velocity checked to determine whether it lies within the bounds of the current stage; in other words, whether it lies below the upper bound on a rising load, and above the lower bound on a falling load. If the water velocity lay beyond the appropriate bound, then clearly a changeover point was located between node $i-1$ and node i. Let V_w be the water velocity at an arbitrary point, and V_{wb} be the water velocity at the operative changeover point. Now, V_w is a function of the load vector,

$$V_w = f(\overline{L}) \qquad \textbf{(13.2.1)}$$

   where a smooth functional relationship has been established between occupancy level and the load vector in step (2) above. It was thus possible to evaluate V_w as a function of occupancy level using ZEBRA and hence, knowing that the search interval was bounded by nodes $i-1$ and i, to find the occupancy level at which the changeover occurred by solving

$$f'(p) = V_w(p) - V_{wb} = 0 \qquad \textbf{(13.2.2)}$$

   using algorithm zero. Let stage j be the stage operative before the changeover occurred, and j' be the stage operative after changeover. Then, having located the occupancy level at which changeover occurred, system performance was evaluated at that occupancy level for both stages j and j'.

---

[149] Fan temperature gain was calculated internally using equation (10.1.3).

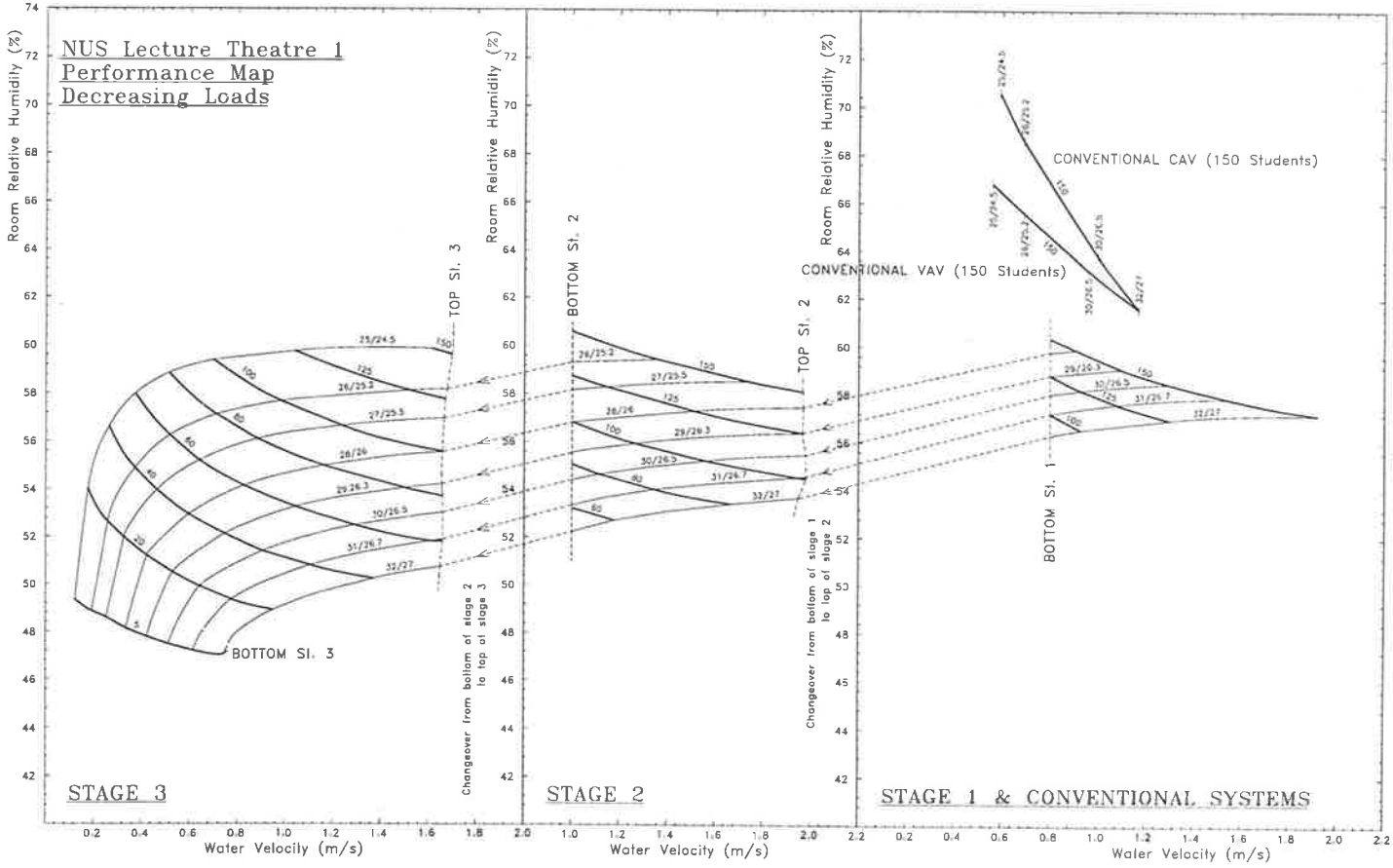


Figure 13.5a. Peformance map for lecture theatre air conditioning system under conditions of *decreasing* load. Lines of constant population slope up to the left. Lines of constant outside air condition slope down to the left. Performance curves for conventional VAV and CAV systems under conditions of full occupancy are shown for comparison.

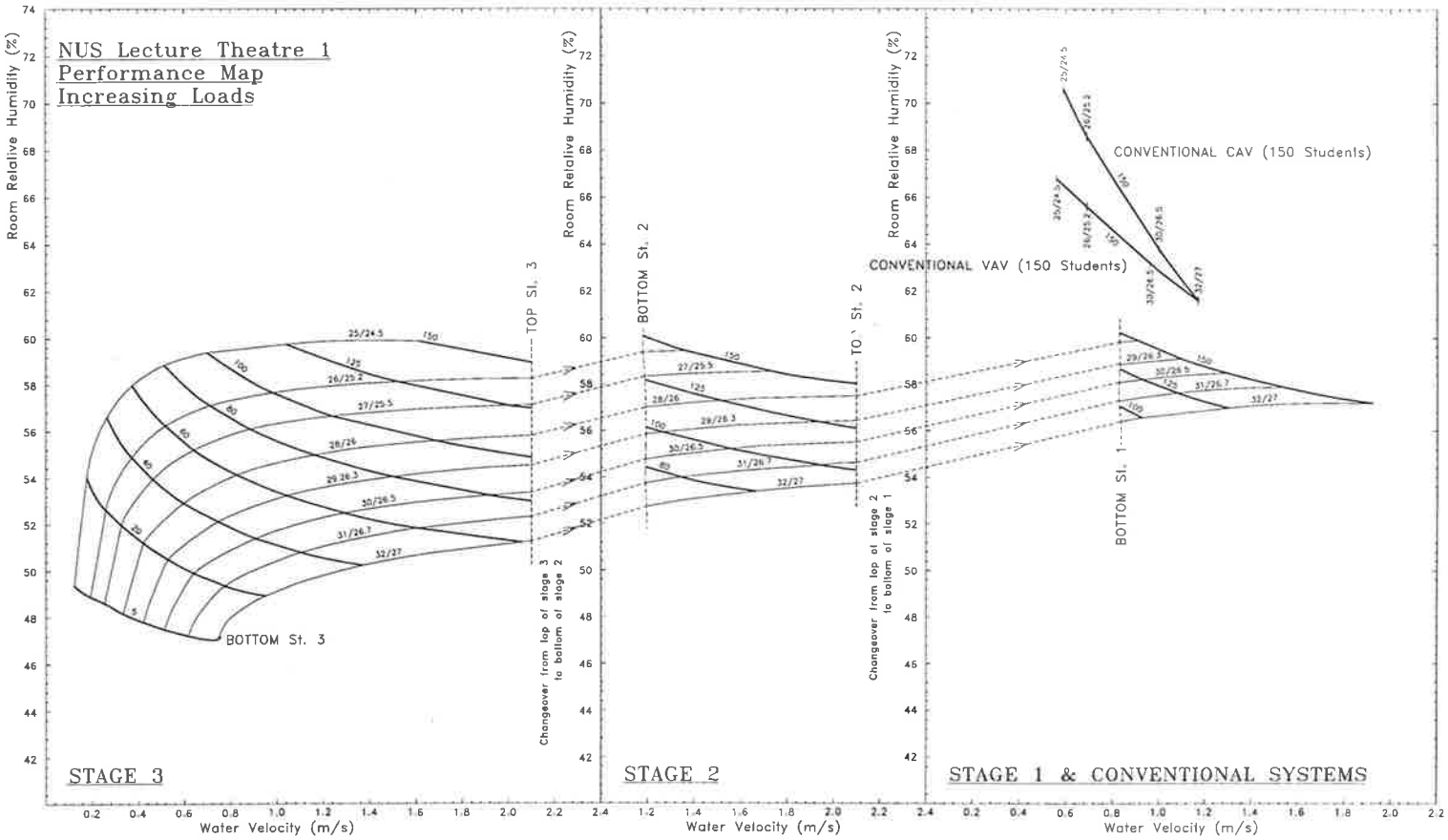


Figure 13.5b. Performance map for lecture theatre air conditioning system under conditions of *increasing* load. Lines of constant population slope up to the left. Lines of constant outdoor air condition slope down to the left. Performance curves for conventional VAV and CAV systems under conditions of full occupancy are shown for comparison.

4. Each load sequence was traversed in both ascending and descending orders to locate the changeover points on rising and falling loads.

A number of alternative graphical representations of the data are possible. A performance map for the system is shown in figure 13.5, in which room relative humidity is plotted as a function of chilled water velocity over a set of loci specifying conditions of constant occupancy, and a set of loci specifying constant ambient conditions. The abscissa has been chosen to show the control action to advantage. Figure 13.5a shows system performance under conditions of decreasing load; the rising load condition is shown in figure 13.5b. Comparing the two diagrams, it is seen that the chilled water velocity immediately following a changeover on decreasing load is lower than the upper breakpoint for the stage following the changeover under all conditions. An analogous observation holds in respect of a changeover on increasing load. Stability of operation in the vicinity of the changeover points is thus ensured *at all times*. It is noted that the relative humidity limit of 60% stipulated in the design brief is satisfied for all but a very small portion of the operational cycle of the system. This is in contrast to conventional VAV and CAV systems. Peformance curves for these latter systems under conditions of full occupancy are also shown in figure 13.5, and in figure 5.10, which plots relative humidity as a function of room sensible load[150]. The part-load performance of the CAV system is especially poor, for reasons which have been described in section 5.1.1. It should be noted that the supply air temperature for each system has been chosen to minimize the risk of condensate forming on the supply air diffusers in the lecture theatre. The superior dehumidification performance of the LFV/HCV system means that a lower dew-point

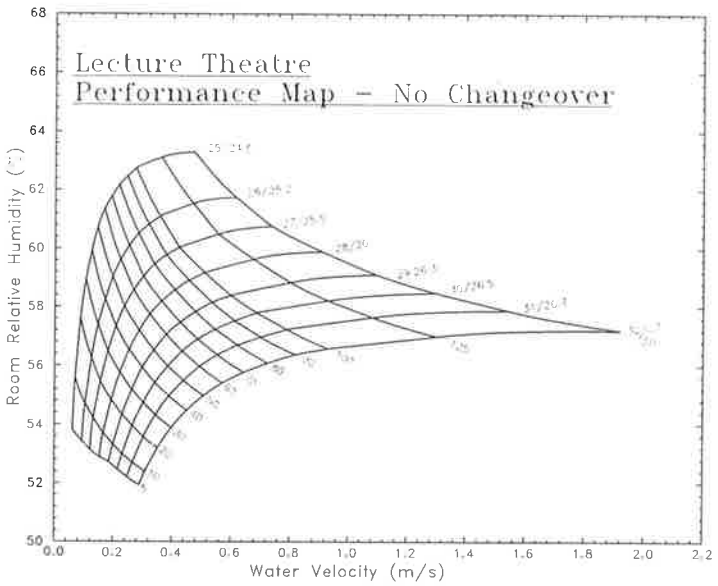


Figure 13.6. Performance map for the lecture theatre air conditioning system without changeovers. Lines of constant population slope up to the left. Lines of constant outdoor air condition slope down to the left.

150 The conventional systems used in these comparisons featured a half-circuited four-row coil with a fin density of 12 fpi, and a face area 914·4 mm (24 tubes) high, and 876 mm wide. It should be noted that the inability of the conventional systems to maintain a sufficiently low dew point in the room required that the supply air temperature be raised to 16°C and the supply air volume be increased to avoid condensation in the supply air diffusers. This incurred a direct energy penalty in terms of the mass of the return air to be treated and a parasitic energy penalty in terms of the fan energy. In part it was this factor which led to the formulation of the HDP concept.

temperature can be maintained using this system than is possible with a conventional system. This in turn makes it possible to use a lower supply air temperature.

For comparison with the performance map of figure 13.5, a performance map for an identical system, but featuring no changeovers, is shown in figure 13.6. Clearly, the LFV/HCV system offers two major advantages over its conventional counterparts. The basic LFV/HCV configuration increases the dehumidification potential of the system, and hence reduces humidity within the conditioned space, under all conditions. It should be borne in mind that the LFV/HCV system used in the present case is a Variable Air Volume system, and shows the same trends as a conventional VAV system at part-load, while of course maintaining approximately the same improvement in dehumidification performance across the load envelope. This behaviour is clearly illustrated in figure 5.10. If the system in its basic configuration proves incapable of satisfying air quality criteria under all part-load conditions, the use of a staging strategy provides a means of restoring the dehumidification potential of the active portions of the coil, and hence of improving dehumidification performance at part-load.

The current example was described in an earlier paper by Shaw et al. (1992), in which an analogy was drawn between the performance map and the "engine map" used by the designers of motor vehicles to determine the best transmission ratios and gear change points, and to

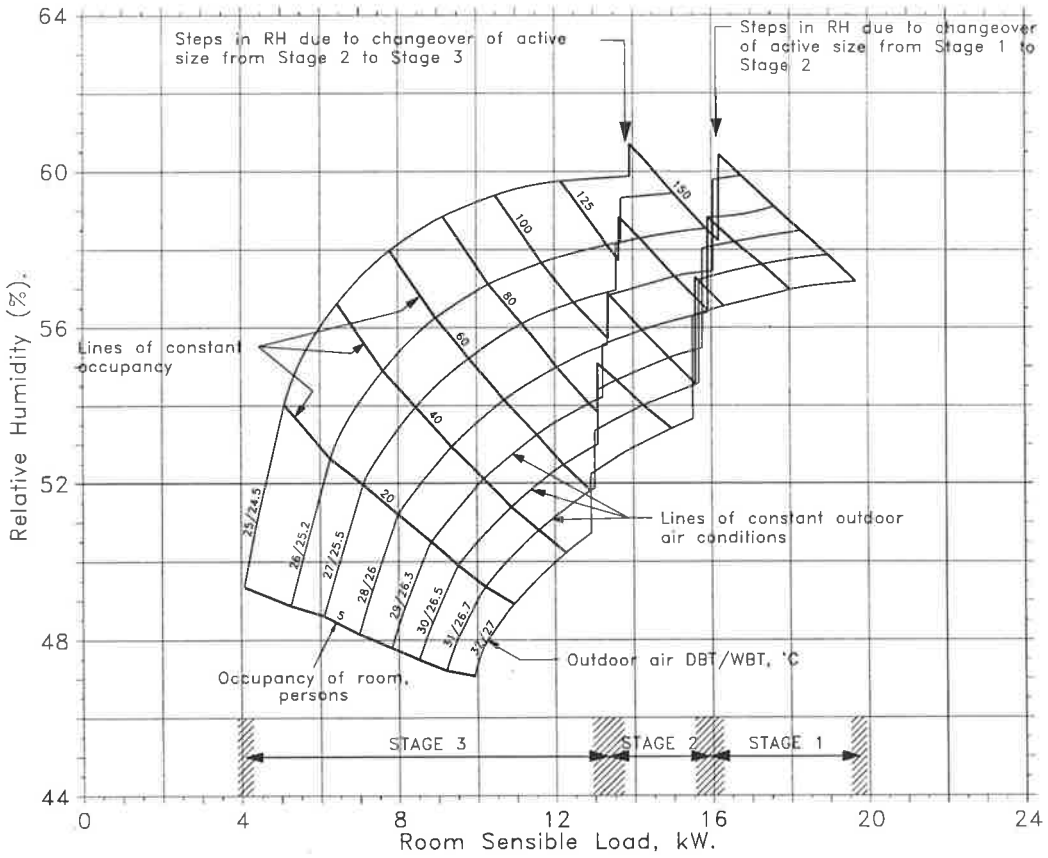


Figure 13.7. Room relative humidity as a function of room sensible load for the lecture theatre air conditioning system under conditions of decreasing load.

predict vehicle performance and fuel consumption. The choice of chilled water velocity as the abscissa for the performance map in this instance admirably fulfils the above objectives, since the water velocity is used as a control parameter for the system under consideration. The use of room sensible load as the abscissa for the performance map arguably provides a more convenient basis for comparing the performance of competing design solutions, since the correspondence between equivalent operating points for different systems is now considerably simplified. Equivalent operating points have the same room sensible load for all systems. The variation in water flow rate across the operating range of the system is, on the other hand, a function of the system design, and the correspondence between equivalent operating points is less obvious. This latter type of representation has been used to compare the performance of the LFV/HCV system with conventional VAV and CAV systems in figure 5.10. In figure 13.7, this representation is extended to present the performance map of an LFV/HCV system as a plot of room relative humidity versus room sensible load.

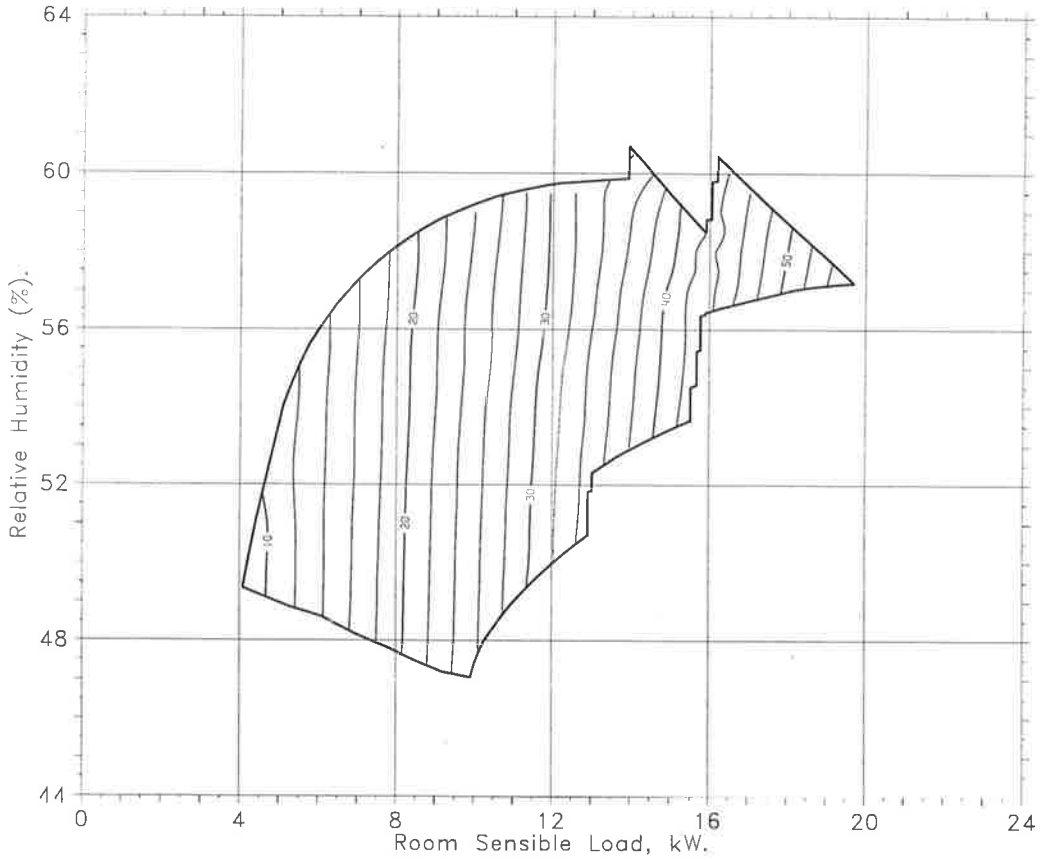


Figure 13.8. Contour map of system refrigeration capacity for the LFV/HCV system. This plot is designed to be used as an overlay for the performance map of figure 13.7.

The most suitable representation, and the choice is not limited to those shown above, will be determined by the needs of the user. In general, it might be expected that the presentation of figure 13.5 would be more useful for the system designer, while that of figure 13.7 provides a

Operating Conditions		LFV/HCV	Conventional VAV	Conventional CAV
Outdoor Air: 32°C db; 27°C wb 150 Students (Peak design condition)	*Total Ref. Cap. (kW)*	56·38	61·20*	61·20*
	Overcooling (kW)	0·00	4·82	4·82
	Ref. Power Consum. (kW)	17·62	19·13	19·13
	Additional Reheat (kW)	0·00	3·47	3·47
	Pump Power (kW)	0·26	0·05	0·05
	Fan Power (kW)	0·04	0·55	0·55
	Total Power (kW)	17·92	23·19	23·19
	Total Add Power (kW)		5·27	5·27
	% Add Power Req'd		**29·44%**	**29·44%**
Outdoor Air: 30°C db; 26·5°C wb 100 Students	*Total Ref. Cap. (kW)*	40·53	41·40	52·50
	Overcooling (kW)	0·00	0·87	11·97
	Ref. Power Consum. (kW)	12·67	12·94	16·41
	Additional Reheat (kW)	0·00	3·91	8·00
	Pump Power (kW)	0·10	0·01	0·03
	Fan Power (kW)	0·02	0·27	0·55
	Total Power (kW)	12·78	17·13	24·98
	Total Add Power (kW)		4·35	12·20
	% Add. Power Req'd		**34·00%**	**95·46%**
Outdoor Air: 26°C db; 25·2°C wb 100 Students	*Total Ref. Cap. (kW)*	29·33	33·00	49·80
	Overcooling (kW)	0·00	3·67	20·47
	Ref. Power Consum. (kW)	9·17	10·31	15·56
	Additional Reheat (kW)	0·00	3·41	9·78
	Pump Power (kW)	0·02	0·00	0·02
	Fan Power (kW)	0·01	0·15	0·55
	Total Power (kW)	9·20	13·87	25·90
	Total Add. Power (kW)		4·68	16·71
	% Add Power Req'd		**50·85%**	**181·65%**

* See footnote 150 on page 372.

Table 13.1. Comparison of power consumption between an LFV/HCV system and conventional VAV and CAV systems constrained to deliver the same dehumidification performance.

clearer means of comparing the performance of various systems. Regardless of the axes selected however, it is possible to build up a complete picture of system performance by plotting contour maps for the entire range of parameters of interest as overlays on the basic performance map. To illustrate this idea, a map of refrigeration capacity is shown in figure 13.8 as an overlay for the performance map of figure 13.7.

The unique set of conditions which permitted the zone loads to be expressed as a function of just two parameters was described earlier in this section. In the general case, the factors affecting the zone loads are both more numerous, and more complex in the interactions which exist between them. While the concept of the performance map is certainly attractive, it is not immediately clear how it can meaningfully be extended to situations in which it is not possible to make simplifying assumptions comparable to those operating in this case.

In the foregoing, a comparison has been drawn between the dehumidification performance of an LFV/HCV system, and that of conventional systems designed to offset the same *sensible* load. It has been noted that the conventional VAV and CAV systems considered in the design

study are unable to meet the design brief over much, if not all of the operating range. It is of course possible, by the use of overcooling and reheat, to *control* these latter systems to meet the desired humidity specifications. This factor must be accounted for in comparing the performance of systems in terms of energy consumption. In other words, if meaningful energy comparisons are to be drawn between different systems, the systems under consideration must be capable of meeting the *same* specifications, in terms of offsetting both sensible *and* latent loads.

Algorithms for solving the moisture staircase problem for a system subject to humidity constraints are described in section 6.3.4.2. With the conventional systems thus constrained to maintain the same relative humidity as the LFV/HCV system[151], the power consumption for the three systems was evaluated at three representative load conditions. The results are tabulated in Table 13.1. In interpreting these figures, note that

i. Refrigeration power consumption was derived from refrigeration capacity using a coefficient of performance of 4·0, and a compressor efficiency of 0·8.

ii. The additional reheat is that which must be added in addition to duct and fan heat gains to obtain the desired performance.

iii. The extremely low fan power consumption for the LFV/HCV system is a result of the low face velocity associated with this system. Power consumption for the spill air fan (which is the same for all systems) has been neglected in Table 13.1.

The figures presented provide an indication of the upper bound of the energy savings which are obtainable using the LFV/HCV methodology.

13.3. Case 2 - A Multistorey Office Building.

The two other case studies documented in this chapter, the National University of Singapore lecture theatre, and the Attakarn Prasit Staff Apartments in Bangkok, deal with actual design briefs. In the case of the first-mentioned, the plant has been in operation since January 1993, and in the second case commissioning was completed in November 1995. The study described in the following is, by contrast, of a conceptual nature. The idea for this study originated from a number of design studies for high-rise buildings in Singapore which were undertaken in early 1992. While the ideas developed in those studies were not adopted in practice at that time, the studies played an important rôle in the evolution of the HDP concept. These design studies formed the basis for a later conceptual study (Marshallsay et al., 1993)

---

[151] The comparison criterion used in this early study (i.e. equal dehumidification performance) over-penalises the conventional systems in that it is strictly only necessary that the conventional systems just satisfy the RH specifications. Any dehumidification which the LFV/HCV systems offer in excess of the design value is an unquantifiable bonus. Nevertheless, overcooling and reheating is required to achieve 60% RH with the conventional systems at all times and the energy penalty remains substantial.

which compared the performance of a typical office building when it is located in each of four capital cities in loacations which span a wide range of climates; Adelaide, London, Singapore and Washington. In that study, simulations were performed for each location at peak sensible load and at a representative part-load condition, each for two outdoor air flow rates, and for a number of system types of both conventional and novel configuration.

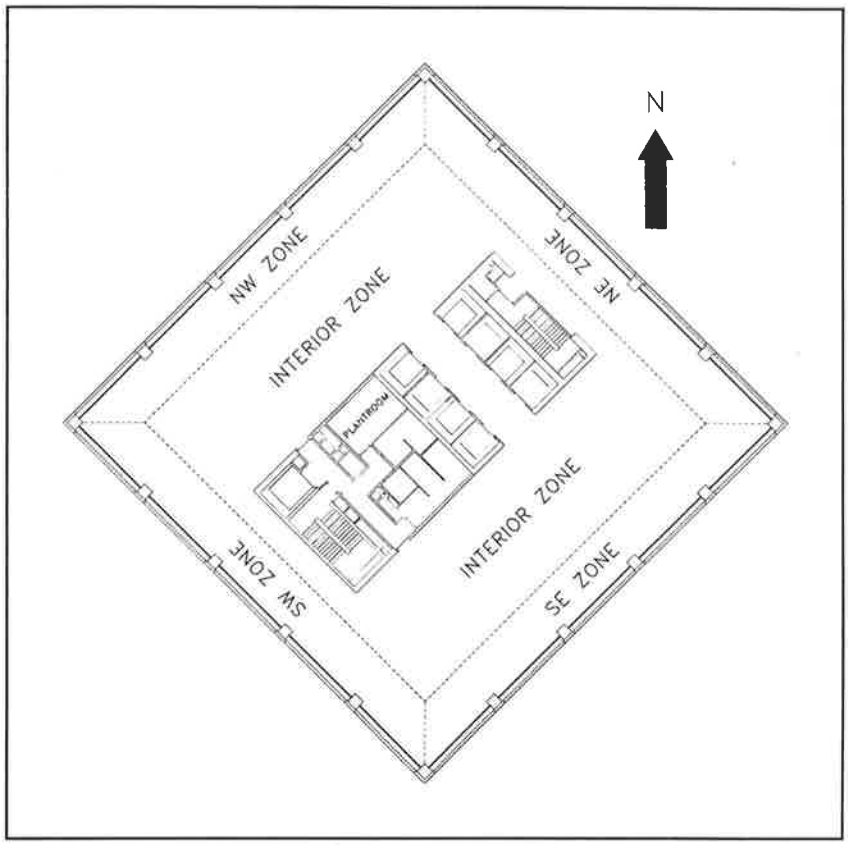


Figure 13.9. Plan view of a floor of the "typical" office building.

The earlier study is revisited in the following. The number of climates considered is reduced to two; Kuala Lumpur and Adelaide. Kuala Lumpur has a typical rainy tropical climate, while that of Adelaide is dry subtropical. The configuration of the system types to be considered has been revised as a result of a reassessment of the earlier study and in the light of our more recent design experience. Finally, the opportunity has been taken to examine a wider range of design and operating parameters, and to make a more detailed presentation than was possible in the earlier study.

The present study is timely in view of recent work undertaken by the group (Luxton and Petrovic, 1995) to identify the parameters which will define a "design matrix" which may be used by the HDP system designer both as an estimating tool, and as a means of developing a sound initial approximation to the design which can subsequently be refined. While the study described below is incomplete in itself, it does provide an indication of the manner in which a simulation package such as ZEBRA may be used to develop, verify and supplement the design matrix.

13.3.1. The Building.

A central preliminary task in this study was that of selecting a "standard" building which would provide a reference for evaluating the comparative characteristics of contending air conditioning design solutions. It goes without saying that, for a given situation, different buildings will exhibit significant variations in thermal behaviour, according to their purpose

and construction. However, one may envisage a "standard" office building, for which the construction, lighting and occupancy schedules may be regarded as being in a sense typical. Orientation, together with climatic regime and location, are then the major factors determining the thermal behaviour of the building. By making the building symmetrical, and setting it with a diagonal facing North, the effect of orientation can be minimized. The building then forms an ideal test-bed for comparing the performance of various system types.

The construction of the "standard" building used for the study is shown in plan form in figure 13.9, and in section in figure 13.10. The construction is based on that of an actual building, and represents a "typical" small to medium size office block of up to perhaps, 20 stories. Each floor is divided into four perimeter zones, and an interior zone. The interior zone has a floor area of 591·4 m², that of each of the perimeter zones being 110·4 m². A floor plan has been chosen in which the service area is completely enclosed within the interior zone, thus making the building almost symmetrical, to reduce the number of variables in the load calculations. The lift lobby (area 33·4 m²) is air conditioned, and for load calculation purposes is regarded as part of the interior zone. The remaining portions of the service areas are not air conditioned. In this exploratory study we have taken the building to comprise 20 identical stories. For the purpose of the load calculations, thermal aspects specific to the ground and top floors have been neglected.

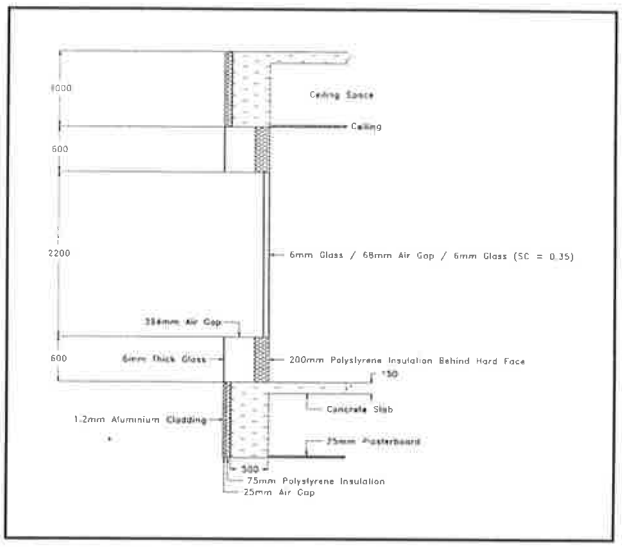


Figure 13.10. Section through the outdoor wall of the "typical" office block.

Return air is ducted through the ceiling plenum. For load calculation purposes, the plenum is divided into five zones, each corresponding exactly in area to one of the conditioned zones. Building construction is assumed to be 'tight' and, on this basis and using information published by ASHRAE (1993; chapter 23), the perimeter zones may be taken to be subject to leakage through the external walls at a steady rate of 60 L/s/zone, while the leakage rate from the interior zone through elevator shafts may be taken to be 13 L/s/floor. These rates are commensurate with a building envelope pressurized to 75 Pa above atmospheric. Unfortunately, it has not been possible to account for the effects of leakage in the present study, since the TEMPER programme which was used to perform the load calculations does not provide for the estimation of *exfiltration*, although *infiltration* may be accounted for[152]. It was however thus possible to simulate the effect on the load of infiltration through the walls of each of the perimeter zones during the period when the plant is not operating, and the building is not pressurized. This infiltration rate was taken to be 100 L/s per zone.

---

[152] Exfiltration cannot be simulated by specifying a negative value for infiltration, since TEMPER will flag such values as 'out of range'.

13.3.2. Climatic Considerations.

Air conditioning system design and energy analysis requires a knowledge of the cooling loads in the conditioned space at appropriate points in the operating cycle. Conventional practice concentrates on design for peak load conditions. For this purpose, design aids are available which tabulate recommended design outdoor-air conditions, in terms of dry- and wet-bulb temperature, for various locations (Australian Construction Services, 1988a; ASHRAE, 1993, Chapter 24). The design methodology being developed at the University of Adelaide (Luxton et al., 1989; Sekhar et al., 1989), while recognizing the importance of peak load conditions for sizing plant, places equal emphasis on design to achieve satisfactory performance under part-load conditions. To support this design philosophy, the *design locus* was introduced (see section 13.2). The design locus enables the designer to identify the outdoor-air conditions which are associated not only with peak conditions, but also with a series of critical part-load conditions. In the general situation however, such information is not in itself sufficient to calculate the associated cooling loads, since no unique relationship exists between the outdoor-air condition on the one hand, and the solar load and thermal history of the building on the other hand. Only in certain circumstances, such as those pertaining in case 1 above, is it permissible to ignore solar load and dynamic effects, and to calculate cooling loads on the basis of outdoor-air conditions alone.

Computer simulation may be used to evaluate the thermal response of a building to arbitrary variations in outdoor-air conditions. A number of computer codes are available which provide such simulations on an hour-by-hour basis, given suitable meteorological input. The input required varies significantly from code to code. The TEMPER (Australian Construction Services, 1987) and BUNYIP (Moller and Wooldridge, 1985) codes, the former of which was used in the present study, synthesize the variations in dry- and wet-bulb temperatures over a 24-hour cycle on the basis of values specified for a maximum at 1400 hours, and the diurnal swing in dry-bulb temperature. This model will be considered further below. At the other end of the spectrum, the energy simulation code ESP-II requires the user to supply actual meteorological records for an entire year[153], covering a wide range of variables. Variations on this latter approach have been proposed in the literature. Yoshida and Terai (1992) advocate the use of weather records synthesized by time-series analysis of existing data records, as a means of generating data records which are more representative than those pertaining to any particular year. Additional advantages claimed for this approach are that it provides a means of accounting for climate trends, particularly those arising from urbanization, of synthesizing records for sites for which records are incomplete or nonexistent, and of compacting the data. Von Thun and Witte (1991) also describe the synthesis from actual records of a "design weather year", incorporating a number of sequences representing extreme weather conditions.

In air conditioning applications simulation may be used as a tool for either design or analysis. The use of a load calculation programme to determine peak load conditions, and hence to size the plant, is obvious. A heat transfer model suitable for calculating the loads in a conditioned space when combined with a plant model, albeit rather basic in parts, allows an estimate to be

---

[153] A selected year of meteorological data, known as the "Test Reference Year", is commonly used as a standard for performing energy analyses for a given location. See Clarke (1985).

made of the energy consumption of the plant. This then allows comparison of the effects of architecture and zoning changes on energy consumption and is the forte of codes such as BUNYIP and ESP-II. It must be stressed that these codes do not claim to be definitive in their definition of an actual plant and hence they must not be used for absolute estimates of energy consumption. Nor should they be used to compare different types of plant. In a recent paper, von Thun and Witte (1991) demonstrate what is perhaps the limit of validity of the use of a programme of this type, namely the optimization of the operating schedules of a notional HVAC system to minimize required capacity by taking advantage of the thermal mass of the building. Unlike codes such as Bunyip and ESP-II, the Zebra Kernel has been designed to provide a reliable estimate not only the energy consumption of a specified air conditioning system, but also to determine the zone conditions which will result when the system is applied to offset a given set of cooling loads. This has been a central aim and an essential element of our design procedures since the beginning of the work described in the present thesis. It is not immediately obvious how the Zebra Kernel can be applied in a *systematic* manner to arrive at a near-optimal design solution for a given situation, and although this is currently an active area of research, our design procedures retain an element of trial and error. Nevertheless, the Zebra Kernel can be used in conjunction with a suitable load calculation programme to compare the performance of candidate plant, zoning and architectural design solutions across a range of load conditions, and to examine the sensitivity of the various designs to changes in the operating parameters.

In analysing system performance, it is necessary to specify a weather data set and sets of occupancy, lighting and equipment schedules which will adequately model the load envelope within which the system is expected to operate. The choice of operating schedules will be described in the following section. In the current study, the TEMPER programme has been used to calculate the loads. Within a TEMPER run, the building loads are calculated over one twenty-four hour period. The daily temperature profile model used by TEMPER is straightforward, the user being required to specify the following parameters:

i. The maximum ambient dry-bulb temperature which occurs within the twenty-four hour period.

ii. The wet-bulb temperature coincident with the maximum ambient dry-bulb temperature.

iii. The diurnal swing in dry-bulb temperature.

It is assumed that the daily maximum is achieved at 1400 hours, the dry-bulb temperature falling to its minimum value, as determined by the diurnal swing, at 0600 hours[154]. A double-cosine curve is fitted to these two points. The ambient moisture level is assumed to remain constant over the twenty-four hour period at the level determined by the humidity at the peak dry-bulb temperature.

---

[154] For Kuala Lumpur at least, the daily maximum and minimum consistently occur close to the times assumed in the TEMPER model (Sham, 1980).

In addition, the user may specify a multiplicative factor [0...1] which will be used to reduce the solar radiation from its clear day value. This factor may be evaluated simply, if not with a high degree of precision, by observing (Haltiner and Martin, 1957) that solar radiation on a fully overcast day is reduced to 46% of its value under clear sky conditions. Then, if C is the fraction of the sky covered by clouds, the multiplicative factor f_s is given by

$$f_s = 1 - 0{\cdot}54\,C \quad . \qquad \textbf{(13.3.1)}$$

It should be noted that certain cloud conditions can augment the solar radiation above the clear sky conditions at some sun angles (Liu and Jordan, 1960). This effect is not consistent in the TEMPER formulation. It remains then to specify the parameters which determine the climate model for the desired months, namely items (i) to (iii) above, together with the cloud cover.

13.3.2.1. Adelaide *(Lat. 34 ° 56 'S; Long. 138 ° 35 'E).*

The analysis is based on a cooling period extending from November through to April[155]. It is noted that the various zones will peak at different times of the year, and this factor is important in sizing equipment, especially for CAV systems. For design purposes, simulations are run for maximum monthly ambient dry- and wet-bulb conditions, as tabulated by Australian Construction Services (1987). The sequence used is:

	Nov	Dec	Jan	Feb	Mar	Apr
dbt (°C)	33·0	36·0	36·0	36·0	35·0	34·0
wbt (°C)	21·0	21·0	21·0	21·0	21·0	18·5
Δt (°C)[156]	9·4	8·7	9·6	9·2	8·7	8·4
f_s	1	1	1	1	1	1

A sequence covering the same months for analysis purposes is constructed as follows:

i. The dry-bulb temperature is based on the highest mean daily maximum, as provided by the National Climate Centre for the Adelaide Regional Office[157] of the Bureau of Meteorology.

ii. The corresponding wet-bulb temperatures are determined using the 5% design locus (figure 13.3).

iii. The values for the diurnal swing are as above.

---

155 The cooling period may in fact extend a month or so on either end of this range, depending on the weather.

156 Figures for the diurnal range Δt were kindly supplied by Dr. Angelo Delsante of CSIRO Division of Building, Construction and Engineering.

157 Recorded over a seventeen year period from 1977, when the Regional Office was relocated to Kent Town.

iv. The factor f_s (equation 13.3.1) is calculated using monthly average 3 p.m. cloud cover statistics extracted from the "Climatic Survey: Adelaide" (Bureau of Meteorology, 1971).

The sequence thus derived is as follows:

	Nov	Dec	Jan	Feb	Mar	Apr
dbt (°C)	28·0	28·8	32·1	32·0	28·9	24·8
wbt (°C)	20·4	20·7	21·5	21·4	20·7	19·4
Δt (°C)	9·4	8·7	9·6	9·2	8·7	8·4
f_s	0·71	0·74	0·80	0·74	0·74	0·66

The above sequence is perhaps not entirely representative of Adelaide, in that the daily mean figures used are based on the highest monthly means recorded during a seventeen year period, rather than on the average monthly means. However, use of these values in conjunction with the corresponding wet-bulb temperatures extracted from the 5% design locus does provide a reasonable basis for comparing the performance of systems in the Adelaide climate.

13.3.2.2. Kuala Lumpur *(Lat. 3 °7 N; Long. 101 °42 E).*

The most comprehensive analysis of the climatology of Kuala Lumpur is undoubtedly that due to Sham (1979, 1980), which forms the basis for the following. The design and analysis data are based on the Subang site, for which the most complete records are available. The need, as before, is to construct separate sequences for design and analysis of air conditioning systems. In this case, cooling will be required for twelve months of the year. For design purposes, it is proposed to use a daily maximum condition of 33°C dbt, 27°C wbt throughout the year. The diurnal swing is as tabulated below. Clear skies are postulated to maximize solar load for sizing of plant.

The sequence used for analysis is constructed as follows:

i. Dry-bulb temperature is based on the mean daily maximum temperatures tabulated by Sham (1980). As shown by Sham, the daily maximum is attained at about 1400 hours throughout the year with a high degree of consistency.

ii. Sham tabulates mean daily average relative humidities, together with hourly correction factors. The values for 1400 hours are used in conjunction with the dry-bulb temperature values to derive the corresponding wet-bulb temperatures.

iii. The values for the mean diurnal swing tabulated by Sham are used.

iv. Cloud cover values tabulated by Sham are used to derive the factor f_s.

The sequence thus derived is as follows:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
dbt (°C)	31·3	32·4	32·7	32·0	32·2	31·2	31·3	31·4	31·4	31·6	31·4	30·3
wbt (°C)	24·6	25·2	25·6	26·2	26·6	25·8	25·7	25·6	26·1	26·1	26·5	25·3
Δt (°C)	9·0	8·9	9·3	7·9	7·8	7·7	8·0	7·8	7·3	7·9	7·2	7·2
f_s	0·58	0·55	0·53	0·54	0·55	0·55	0·55	0·55	0·53	0·54	0·53	0·53

13.3.3. Operating Schedules.

It remains to specify schedules for (i) plant operation, (ii) occupancy, (iii) lighting, and (iv) equipment operation. These will be considered below.

13.3.3.1. Plant Operation Schedule.

In the model used in this study, the plant operation schedule is as follows:

1. For Adelaide, the plant operates to provide daytime cooling to 24°C in all zones from 0600 to 2000 hours. This allows three hours for the building to reach its design temperature before full occupancy is achieved. It would be desirable to use an economy cycle for night time operation during much of the cooling season in Adelaide. It is not however immediately obvious how this may best be simulated using TEMPER.

2. For Kuala Lumpur, the plant operates to provide daytime cooling to 24°C in all zones from 0600 to 1900 hours. In view of the high temperature and humidity content of the outdoor air, it is also desirable to provide cooling at night, both to minimize the startup load on the plant as above, and also to inhibit the buildup of moisture in the conditioned space.

13.3.3.2. Occupancy Schedule.

An occupancy level of 1 person per 10 m² has been used for the perimeter zones. A higher occupancy level is assumed for the interior zone, but allowing for the fact that a sizeable portion of this zone is occupied by the lift lobby and corridors, an *average* occupancy level of 1 person per 10 m² has been used for this zone also.

The basic working day is taken to be 0900 to 1700 hours for Adelaide, and 0800 to 1600 hours for Kuala Lumpur. The occupancy schedules for the two sites are as shown in Appendix G.

13.3.3.3. Lighting Schedule.

High efficiency lighting is used, with a total power consumption of 15 W/m². This is dissipated as follows (Kimura, 1977):

i. 75% (11·25 W/m²) is removed in the ceiling plenum. Of this, 67·5% is convective.

ii. The remaining 25% (3·75 W/m²) enters the room. None of this is convective.

The lighting schedule is as follows:

i. Full lighting is used from 0800 to 1800 hours.

ii. Between 0700 and 0800 hours, and between 1800 and 2000 hours, the lighting is reduced to 50%.

iii. Between 0600 and 0700 hours, and between 2000 and 2200 hours, the lighting is reduced to 10%.

13.3.3.4. Equipment Schedule.

The equipment schedule follows the occupancy schedule, except that there is no reduction in equipment power consumption during the lunch period. The power consumption due to equipment usage is assumed to be 18 W/m², giving a total equipment load of 1·99 kW for each of the perimeter zones, and 10·05 kW for the interior zone.

13.3.4. Equipment Selection and Operation.

Three system configurations have been selected for analysis in the current study:

i. A conventional multizone variable air volume (VAV) system configured as shown in figure 5.7, using a single cooling coil in a draw-through configuration. One air-handling unit is allocated to each floor. For this system, the fan and coil must be sized to handle the air flow and cooling requirements associated with the *simultaneous* peak sensible load imposed on the system.

ii. A conventional multizone constant air volume (CAV) system configured as shown in figure 5.5. Each zone is served by an individual cooling coil. Again, one air-handling unit is allocated to each floor. The fan for this system must be selected to handle the *sum* of the air flow rates required to offset the individual peak sensible loads for the various zones. Similarly, the coil serving each zone must be sized according to the individual peak sensible load for that zone.

iii. An HDP system operating in VAV mode and configured as shown in figure 5.11. Two outdoor air treatment units are installed approximately halfway up the building (on floor 11, say), each serving ten floors. Within each floor, each zone is served by a variable air volume fan-coil unit (FCU) containing a return-air coil. The treated return air is mixed with treated outdoor air supplied by the central plant, before entry to the fan. The fan and coil for the fan-coil unit serving each zone must be sized according to the peak sensible load for the zone. The configuration selected in this case will permit the fan-coil units to be mounted within the ceiling space, thus eliminating the need for plantrooms on all but the eleventh floor, and increasing the overall rentable space within the building. The need to size the units to fit within the

ceiling space will be considered in selecting the cooling coils. (The choice of ceiling mounting may not be acceptable to some building managers, but this option is chosen here to illustrate the flexibility of the design procedure).

In order to establish the design conditions, load calculations were performed using TEMPER for the design sequences listed in section 6.3 above. Zone conditions at the time of the individual zone peaks, and at the time of the overall peak were as tabulated below for Adelaide:

Peak	Month	Time	NW Zone		NE Zone		SE Zone		SW Zone		Int. Zone	
			SH	SHR	SH	SHR	SH	SHR	SH	SHR	SH	SHR
NW Zone	Apr	16:00	**16·306**	**0·952**	7·741	0·904	7·328	0·899	7·407	0·900	21·666	0·830
NE Zone	Apr	11:00	6·890	0·893	**15·310**	**0·949**	6·554	0·888	6·537	0·888	21·579	0·830
SE Zone	Dec	10:00	6·328	0·885	11·459	0·933	**11·316**	**0·932**	7·359	0·899	22·298	0·834
SW Zone	Dec	17:00	12·612	0·937	8·435	0·911	8·454	0·911	**13·095**	**0·941**	22·276	0·834
Int. Zone	Dec	10:00	6·328	0·885	11·459	0·933	11·316	0·932	7·359	0·899	**22·298**	**0·834**
Overall	Dec	17:00	12·612	0·937	8·435	0·911	8·454	0·911	13·095	0·941	22·276	0·834

The corresponding conditions for Kuala Lumpur were as follows:

Peak	Month	Time	NW Zone		NE Zone		SE Zone		SW Zone		Int. Zone	
			SH	SHR	SH	SHR	SH	SHR	SH	SHR	SH	SHR
NW Zone	Jun	16:00	**14·923**	**0·948**	7·082	0·896	6·911	0·893	8·125	0·908	21·056	0·826
NE Zone	Jun	09:00	5·900	0·877	**13·409**	**0·942**	6·574	0·888	5·820	0·876	20·226	0·820
SE Zone	Dec	09:00	6·050	0·880	6·400	0·886	**14·144**	**0·945**	6·137	0·881	20·319	0·821
SW Zone	Dec	16:00	7·913	0·906	7·007	0·895	7·218	0·897	**15·375**	**0·949**	21·122	0·827
Int. Zone	Dec	16:00	7·913	0·906	7·007	0·895	7·218	0·897	15·375	0·949	**21·122**	**0·827**
Overall	Sep	15:44	11·755	0·934	7·192	0·897	7·214	0·897	12·491	0·938	21·078	0·826

It will be observed that the design peak loads for the perimeter zones in Kuala Lumpur fall within a narrow band. For Adelaide, which is located somewhat further from the Equator, there is a significant difference between the peak design loads for the southerly facing zones, and the northerly facing zones.

Two levels of ventilation air flow rate have been considered in the present study. A ventilation air flow rate of 10 L/s per person accords with the minimum level recommended by many current standards (see Chapter 3). Levels as low as 2·5 L/s per person are however commonly specified in monsoonal Asia.

The three factors, system configuration, climate and ventilation air flow rate define a matrix of test cases for the study. In evaluating the fan, duct and other miscellaneous temperature gains, the following assumptions were made:

i. For the multizone VAV and CAV systems, the pressure drop through the duct system is 500 Pa at peak air flow. Note that the pressure drop through each branch of the duct

system must be balanced to match that through the branch with the highest pressure drop, in accordance with the principles derived in Chapter 9. For the HDP system, the duct runs are considerably shorter, and the duct systems are correspondingly less complex. The pressure drop at peak air flow has been taken as 100 Pa for the perimeter zones, and 150 Pa for the interior zone. For the HDP system only, allowance must be made for a pressure drop through the outdoor air duct from the central outdoor air treatment plant to the fan-coil units; 400 Pa has been allowed. In all cases, the fan efficiency is 0·75, and the motor efficiency is 0·9, with the motor located *in* air stream.

ii. Return air temperature gains are as calculated using TEMPER. For the multizone systems, a supply-air temperature gain of 0·5 °C has been used throughout. For the HDP systems, duct runs will typically be shorter than for the multizone systems, and the supply-air temperature gain is assumed to be 0·2 °C for the perimeter zones, and 0·3 °C for the interior zone. The treated outdoor air to the fan-coil units is also subject to a temperature rise of 0·4 °C in the supply duct.

iii. The chilled water supplied to the multizone units, and to the outdoor air treatment unit in the HDP system, is at 6·5 °C[158]. Within the pipework between the outdoor air treatment unit and the fan-coil units, the chilled water in the HDP system is subject to a temperature rise of 0·4 °C.

The analysis has been undertaken as a two-stage process:

1. The coils were selected to offset the appropriate peak load values, as tabulated above. For the VAV units, the supply air temperature is 15 °C. The supply air flow to each zone served by the CAV system has also been selected to provide a supply-air temperature of 15 °C at the design peak load for the zone. The dry-bulb temperature in all the zones has been maintained at 24 °C throughout the study.

   Details of the coils selected and of system performance at peak load may be found in *Appendix F*.

2. The performance of each of the systems was evaluated for operation over a daily period for each of two months for each site. The appropriate loads were calculated using TEMPER. The parameters defining the daily variation in ambient conditions are as tabulated in section 13.3.2. The months selected were:

   a. For Adelaide,
      i. *January* provides the highest overall load (within the analysis sequence) of any month, and

---

158 The chilled water temperature has been selected to conform with common practice. In principle, higher temperatures are feasible using the HDP methodology. The extent to which chilled water temperature can be raised in practice is currently an active area of research.

ii. *April* represents the end of the cooling season, and provides the most difficult part-load conditions.

b. For Kuala Lumpur,
   i. *March* marks the end of the North-East Monsoon, and is associated with the highest sensible loads, and
   ii. The onset of the North-East Monsoon occurs during *November*, during which month the highest average humidity levels are encountered (Sham, 1980).

A summary of the major results from the simulation process are presented in the appendices, as follows:

a. The ambient temperature and zone loads, represented as plots of sensible load and sensible heat ratio, are contained in *Appendix B*. These are plotted over the period from 07:00 to 20:00, a common abscissa being used throughout appendices B to E. Cooling loads only are shown.

   It will be observed that the sensible load in the large interior zone dominates, and remains nearly constant throughout the period when the building is at or near full occupancy. During this period, the zone sensible heat ratio in the interior zone generally remains lower than that in the perimeter zones.

   The loads in the perimeter zones generally peak in the early morning or late afternoon. In the early morning the westerly facing perimeter zones receive little or no direct solar radiation, and sensible loads are generally low. This may give rise to a short period of extremely low sensible heat ratio when the building first reaches full occupancy. This situation will potentially tax the ability of an air conditioning system to offer adequate dehumidification. The effect is particularly pronounced in Adelaide, but less so in Kuala Lumpur. The corresponding effect is also far less pronounced in the easterly facing zones in the late afternoon, when heat stored in the building fabric during the day will contribute significantly to the zone sensible loads.

b. The conditions obtained in each zone, as measured by the relative humidity, are plotted in *Appendix C* for each point in the test matrix. Plots are also presented illustrating the effect of staging on room conditions for an HDP system in Kuala Lumpur. Note that the zone conditions in all cases are those obtained without the use of overcooling and reheat.

c. Power consumption for the various points in the test matrix is described and intercomparisons are drawn in *Appendix D*. The power consumption has been calculated on a building-wide basis is itemized into four components:

   i. *Refrigeration.* The cooling load for the coils is calculated directly by ZEBRA. To derive the power consumption of the associated chiller,

the coefficient of performance of the refrigeration cycle is assumed to be 4·0, and the efficiency of the compressor to be 0·8.

ii. *Fan power*, calculated as the electrical power input to the fan motor.

iii. *Pump power*, calculated on the basis of pressure drops through pipework and cooling coils. In fact, the latter component dominates, but the pump power is in any case never more than a minor portion of the total.

iv. *Reheat.* Calculated for CAV systems only, where overcooling and reheat have been applied to reduce zone humidity to an acceptable level of 60%.

The plots fall into three series:

i. Power consumption for a conventional multizone VAV system is compared with that of an HDP system.

ii. Power consumption is graphed for a conventional multizone CAV system, with overcooling and reheat applied as described above. In this series it is convenient to use a different ordinate scale than that which has been used in the other graphs.

iii. Power consumption is compared for an HDP system with and without staging.

d. *Appendix E* compares the chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system.

The results of the study are discussed below for each of the three system configurations considered. It is emphasized that the study as described here is of a *preliminary* nature only, and that a number of issues have not been addressed, the most important of these being the need to define a strategy to control the system under conditions of very low load. These issues will be considered further in the following.

13.3.4.1. Conventional Multizone VAV System.

The coils used for the VAV system simulations were selected on the basis of the simultaneous peak load. Specifications for the coils selected, and details of the system performance at the simultaneous peak load conditions will be found in Appendix F. Performance over a range of part-load conditions is described by the graphs of appendices B to E.

The coils for this application were selected to operate with an air face velocity of about 2·5 m/s and a chilled water velocity of about 1 m/s at peak. Under peak load conditions the relative humidity for all zones is maintained below 55%, and in Adelaide, that of the perimeter zones

is maintained below 50%. As expected, the humidity of the air in the zones rises under part-load conditions, and it is evident from the results of the simulations that dehumidification performance approaching that of the design peak will only be experienced over a very small portion of the operating cycle of the system. Nevertheless the off-peak behaviour of the system is considerably better than that of the single-zone conventional VAV system considered in connection with Case 1 above. This is because two major factors serve to dampen out the variations in the *total* space load in course of a working day. First, the major contributor to the total space load is the interior zone and, as has been remarked upon earlier, the load within this zone shows little variation while the building is at full or near full occupancy. Secondly, while the solar load on the easterly facing zones will decrease during the course of a day from an early morning peak, that on the westerly facing zones will increase towards a late afternoon peak. The sum of the contributions of the perimeter zones to the total space load thus varies smoothly within a relatively narrow band during the course of the working day. As a result, the multizone VAV system is able to reduce fluctuations in the humidity levels within the various zones to a minimum during working hours. Zone humidity levels only really start to deteriorate significantly when the *total* space load falls.

It is interesting to note that dehumidification performance of this system is quite insensitive to the quantity of outdoor air handled. A four-fold increase in the outdoor air flow in the humid tropical climate of Kuala Lumpur only results in an increase in the relative humidity within the conditioned space of approximately 1%, this increase being experienced equally in all zones at all times. The corresponding increase in zone relative humidity in the comparatively dry climate of Adelaide is even less. The dehumidification performance of the multizone VAV system is rather more sensitive to variations in space load. Thus in Kuala Lumpur, where the climate variations are contained within a fairly narrow band in the course of a year, thus constraining the range over which the total zone space load varies, the daily zone humidity profile remains almost constant from one month to the next. The picture is somewhat different in Adelaide, which is subject to significant seasonal climatic variations together with a larger diurnal swing than is commonly encountered in the Tropics. In January, when the loads on the perimeter zones are high, relative humidity is maintained between 50 and 55% for all zones except for a brief period in the early morning. By contrast during April, at the end of the cooling season, the relative humidity in some zones will exceed 55% for an appreciable portion of the day, although levels in excess of 60% are never encountered, as a result of the moderating influence of the multizone system which was remarked upon earlier.

While the effect of outdoor air quantity on dehumidification performance is minimal for both Adelaide and Kuala Lumpur, an increase in this parameter may have a significant adverse impact on power consumption, which is highly dependent upon the moisture content of the outdoor air. The effect of a fourfold increase in outdoor air quantity in the relatively benign climate of Adelaide results in an increase in the required refrigeration capacity which at no time exceeds 16% between the hours of 09:00 and 17:00, and is considerably smaller over much of this period. By contrast, the corresponding increase in Kuala Lumpur during March does not fall below 27%, and at times approaches 40%. A comparison of tables F.2 and F.5 shows that for Adelaide the increase in refrigeration capacity required by the increased outdoor air quantity is almost exclusively used for *sensible* cooling, while for Kuala Lumpur the *latent* component is far larger than the sensible component, and is also appreciably larger than the

increase in sensible component caused by increase outdoor air quantity in the case of Adelaide, even though the peak dry-bulb temperature for Kuala Lumpur is somewhat lower than that for Adelaide. The difference in the impact of increased outdoor air quantity for the two sites on the *annual* power consumption for each will be even more marked than a comparison of peak months would suggest. For Kuala Lumpur, as remarked previously, the climate varies within a fairly narrow band in the course of a year, and temperature and humidity level remains high throughout the day. Consequently, the cooling season extends over the whole year, and it is also likely that the air conditioning plant would be operated on a 24-hour basis for a building such as that under consideration. In Adelaide the outdoor climate varies considerably during the course of the cooling season, and also from day to day within the cooling season. A comparison of figures D.3 and D.4 shows that the adverse effect of increased air quantity on power consumption is much less for April than for January, and that there is even a period in the early morning when the effect of the increased outdoor air is beneficial to power consumption. In addition, the cooling season for Adelaide only lasts for about six months, and it is frequently possible to take advantage of an economy cycle, in which outdoor air quantities are increased subatantially, to reduce power consumption during the cooling season, a strategy which is not applicable in a humid tropical climate.

13.3.4.2. Conventional Multizone CAV System.

The coils for the CAV system were selected to offset the individual peak loads for the zones served, and as in the case of the VAV system, each coil was selected to operate with a face velocity of about 2·5 m/s and a chilled water velocity of about 1 m/s at peak.

As shown in tables F.9 and F.12, the coils selected are able to maintain conditions in their respective zones below 55% for each perimeter zone, when the zone is subject to its individual peak load. The relative humidity in the interior zone lies just above 55% for both Adelaide and Kuala Lumpur under similar conditions. However, dehumidification performance deteriorates markedly during part-load conditions, for reasons which have been discussed in section 5.1.1. For the load sequence used to simulate zone conditions in Adelaide during January, the relative humidity within the perimeter zones lies between 60 and 65% while the building experiences full or near-full occupancy. During the same period the relative humidity in the interior zone, which is subject to near-peak loads throughout this period regardless of ambient conditions, varies between 56 and 59%. In April, when the loads in the perimeter zones are, for the most part, considerably less than their January values, the relative humidity in *all* zones rises by some 3 to 5%. This provides a reminder that this is indeed a multizone system. Even though each zone has its own cooling coil and supply air distribution system, they share a common return air system, and this serves to dampen inter-zone variations to a certain extent. With a fall-off in building occupancy levels, the relative humidity in all zones rises rapidly to levels in excess of 70% in January, and greater than 80% in April. In the relatively dry climate of Adelaide, dehumidification performance appears to be almost entirely insensitive to the outdoor air flow rate.

In Kuala Lumpur, the multizone CAV system exhibits behaviour which differs markedly from that experienced in Adelaide. While the building is at or near full occupancy the relative humidity within the interior zone is maintained between 55 and 60%, regardless of ambient

conditions or outdoor air quantity. There is a marked difference however in the response of the perimeter zones to changes in both ambient conditions, and to the quantity of outdoor air delivered. Increasing the quantity of (humid) outdoor air generally raises the level of humidity experienced in the perimeter zones. It also increases the degree of variation in humidity levels between zones, and in the fluctuations in humidity level experienced in the course of a working day. These effects, already evident in March, are exacerbated in November. Two factors probably operate to produce this seasonal effect, these being the higher ambient humidity levels, and greater diversity of perimeter zone loads resulting from the lower sun angles in November (see Appendix B). As for Adelaide, dehumidification performance deteriorates markedly when building occupancy levels fall, the relative humidity levels in all zones approaching or exceeding 80%.

The relative humidity plots of Appendix C document the performance of a CAV system with no control placed on the humidity levels. The resulting dehumidification peformance is, for the perimeter zones at least, unacceptable by any standards at almost all times. The exercise has been repeated with overcooling and reheat applied as necessary to constrain the relative humidity level within all zones to a maximum of 60%, which falls on the upper boundary of the ASHRAE comfort zone, as defined in ASHRAE Standard 55-1989 (see Chapter 3). The resulting relative humidity levels are not documented here. However, the power consumption for a system subject to such control is plotted in figures D.9 to D.16. The extra power required to overcool, and more particularly to reheat the air, considerably increases the power consumption when compared with the VAV systems which are, for the most part, able to maintain relative humidity levels below 60% without any control action. At near-peak conditions for Adelaide (January, 17:00), the extra energy consumption is of the order of 70%, and becomes progressively larger as the zone loads fall. The power consumption required to maintain acceptable relative humidity levels in Kuala Lumpur is generally much higher than that required in Adelaide. For both locations, the power consumption required to maintain acceptable conditions outside of working hours is exorbitant. For Adelaide, the effect of a four-fold increase in outdoor air quantity is minimal, increasing power consumption slightly when ambient temperatures are high, and decreasing it slightly when ambient temperatures are low. In Kuala Lumpur, the effect of increasing outdoor air quantity is always to increase power consumption, although on a percentage basis the increase is much smaller than that experienced with a VAV system. This is however cold comfort for the building owner faced with the high cost of running a CAV system.

The second feature to emerge from the power consumption plots is the contribution of the fan, which is considerably higher than that of the VAV systems, and remains constant throughout the operating cycle. This is to be expected, since the CAV system design requires that the air quantity delivered to each zone at all times equals that required to offset the individual peak load for the zone, the supply-air temperature rising to satisfy reduced cooling requirements. In the VAV systems on the other hand, the air flow rate to a zone adjusts to offset the zone load while maintaining a constant supply-air temperature.

As a final note, HDP principles may be used to improve the dehumidification performance of a CAV system without incurring the energy penalties associated with traditional overcooling and reheat practices. The performance of such a system in an apartment block will be

considered in the next case study. The use of a two- or three-speed fan also offers a means of improving the part-load performance of CAV systems. Both of these options should be examined in a more comprehensive study than has been possible here.

13.3.4.3. HDP System.

For the HDP system considered in this study, the coil in each fan-coil unit has been selected to offset the peak load for the zone which it serves. Unlike the two systems considered previously, the HDP system modelled is *not* a multizone system and each coil, with the possible exception of that serving the interior zone, is consequently required to handle a greater range of zone loads than the single coil in the multizone VAV system. This factor should be borne in mind in drawing comparisons between the various systems described. Some characteristics of the performance of the system described may be attributed directly to the fact that it is an HDP system; others relate more to the fact that it uses a distributed network of terminal units in preference to a system of floor-by-floor multizone units. The *High Driving Potential* concept does not prescribe a particular system configuration. The principles may be applied with benefit to both VAV and CAV systems serving both single and multiple zones. In addition, HDP systems may treat the outdoor air in one or more dedicated central units, as has been done here, or they may package an outdoor air coil and a return air coil together in a stand-alone unit, as has been done in the other two case studies considered here. The HDP principle lends itself particularly to the distributed configuration considered here since, given a sufficiently high flow rate of outdoor air, the size of the coils in the fan-coil units may be reduced, facilitating their location in the ceiling plenum.

The outdoor air treatment units were designed to supply treated outdoor air at temperatures of between 7·5°C and 10°C across the range of operation of the system (see Appendix H)[159]. The corresponding water temperature rise varied between 1°C and 4°C. The constraints on the quantity of chilled water which can be used in the outdoor air units will be discussed below. Coil performance at the simultaneous peak load condition is tabulated in Table F.15 for Adelaide, and Table F.20 for Kuala Lumpur. Considerable dehumidification occurs, and in the case of Kuala Lumpur latent heat removal in fact dominates. The implications of this for the terminal units can be deduced by examining the cooling loads on the coils of these latter units (Tables F.16 and F.21). At the lower outdoor air flow rate considered, the cold, dehumidified outdoor air offsets much of the latent load in the return air; at the higher outdoor air flow rate, it effectively cancels the latent load, while at the same time reducing the sensible load on the coils.

The terminal units have been designed according to conventional principles, since a high level of dehumidification is not called for. The resulting units are indeed compact.

Comparing first the power consumption for the HDP system with that for the multizone VAV system (Figures D.1 to D.8), it will be seen that there is little to choose between the two

---

[159] The large coils required at the higher outdoor air flow rate could be replaced with two or more smaller coils of equivalent total capacity, and it may even be desirable to replace the single outdoor air unit with two smaller units, one of which could be shut down under part-load conditions.

systems in terms of the refrigeration component, conditions at times slightly favouring the multizone VAV system, and at other times slightly favouring the HDP system. Fan power consumption is considerably lower for the HDP system, but this is due to the fact that each zone is served by its own unit, and duct pressure losses can consequently be reduced in comparison with those for the multizone system. In any case, the duct pressure losses used in this study are based on reasoned assumptions. A more objective analysis using the principles developed in Chapters 9 and 10 would be required to determine the extent to which this advantage can be realised in practice.

Turning next to the dehumidification performance of the HDP systems, it is instructive to consider performance at the lower outdoor air flow rate first. Each unit, when operating at its individual peak load is capable of maintaining relative humidity within the zone served at or below 55% (see Tables F.17 and F.22). Some deterioration is experienced at part-load, but relative humidity is for the most part maintained between 55 and 60% during working hours for the perimeter zones, and close to 55% for the interior zone. The early morning dip in sensible heat ratio in the westerly facing perimeter zones experienced in January in Adelaide, and commented on earlier, proves to be particularly problematic. The deterioration in dehumidification performance at part-load is undoubtedly due to the fact that this system uses single zone terminal units, a fact that is borne out when one compares the diversity in zone conditions at any time with that associated with the multizone VAV system. For the latter system, the zone conditions are almost always contained within a fairly narrow band. For the former, they may span a considerable range at those times when some zones are near peak load, while the load on other zones is comparitively low. The effect of the cold outdoor air in the HDP system is to contain the degree of deterioration which would otherwise be experienced. As the load falls, the outdoor air progressively comprises a greater proportion of the whole. When the occupancy of the building falls, the dehumidification potential of the air becomes sufficient to arrest, and in fact to reverse the upperward trend in zone humidity. This behaviour is the inverse of that which is experienced with conventional systems.

The use of *staging*, described in sections 5.2.1, 6.5.4 and 8.3.3, offers a further means of improving the part-load performance of the system. In the present study, the effect of staging on the indoor climate in the perimeter zones has been investigated for Kuala Lumpur. The staging scheme involves deactivating half the circuits in the first two rows of the four-row coil when the chilled water velocity falls below 0·5 m/s, and reactivating the circuits when the velocity again climbs above 1·5 m/s. The improvement in performance is dramatic, as shown in figures C.25 and C.26. Relative humidity in the zones where staging is used is now maintained below 55% at almost all times. Similar levels of improvement would be expected for Adelaide. In the simulation sequences shown, all coils were operated throughout in the reduced configuration. With the sequence intitiated in the early morning with the coils fully activated, coolant velocities were sufficiently low to trigger an immediate transition to the reduced configuration, as would be expected. At no time in the day did the coolant velocity in any coil climb to a level sufficiently high to trigger a reverse transition. It is important to realise that the coils selected are *not* oversize; the full coil is required to offset the design peak load for each zone, and will *occasionally* be required to offset the loads encountered in an operational cycle. This underlines the *importance of designing for part-load conditions* which are, after all, the norm. Secondly, comparing the performance of this system with that of the

multizone VAV system, it becomes clear that *it is particularly important to consider part-load conditions when designing for single-zone units*. Finally, it should be noted that the benefits of staging are obtained at the cost of a negligible energy penalty, as shown in figures D.17 and D.18. Insofar as there is an energy penalty, it can almost entirely be attributed to the extra pump power required as a result of the increased water pressure drop through the coils.

When the higher quantity of treated outdoor air is used, the latent load on the return air coils is effectively cancelled, and the use of staging becomes redundant. The relative humidity in all zones at all times lies well below 55%. Another problem now becomes apparent. With falling zone loads a point is reached at which the cooling capacity of the outdoor air exceeds the return air cooling load, even with the return air coil deactivated. This problem has not been considered in the present study, but has been solved comprehensively in the case of an actual HDP installation, the Attikarn Prasit Apartments in Bangkok, which will be considered shortly. A part-load strategy would probably comprise a combination of the following actions:

i. Permitting the zone dry-bulb temperatures to drift downwards until a new equilibrium point is reached.

ii. Reducing the chilled water flow through the coil of the outdoor air treatment unit.

iii. Reducing the outdoor air flow. This is permissible during periods of low occupancy, provided ventilation standards are met.

A part-load strategy should preferably aim to prevent the relative humidity level in any zone falling below 40 to 45%.

It is opportune at this time to consider the factors which affect the choice of a suitable chilled water flow rate for the outdoor air unit. Clearly, a lower constraint will be imposed by the requirement that the total flow through this unit equals or exceeds the total requirements of the terminal units served by the outdoor air unit. Given that constant speed pumps are the norm in practice, the flow through the outdoor air unit will in fact need to equal or exceed the peak total requirements of the terminal units. In fact, it will be usual to select a somewhat higher flow rate, as has been done here. The HDP strategy in most cases reduces the chilled water requirements of the units on the various floors of the building[160], as shown in Appendix E. The reduction is particularly impressive in Kuala Lumpur when the higher outdoor air quantity is used. This will reduce pumping costs, which it has been noted are a minor component of the total energy bill. More importantly, it may enable the selection of smaller capacity pumps, and smaller diameter pipes, both of which will result in significant savings in capital costs. The graphs of Appendix E only tell part of the story in this regard. Pipes and pumping equipment must be sized to handle the chilled water required to offset the *simultaneous peak load* for the building. The chilled water requirements (L/s) per floor under simultaneous peak load conditions are tabulated below for the cases considered:

---

[160] It is assumed that the outdoor units will be located in close proximity to the chiller(s).

	VAV		HDP	
	O/A @ 2·5 L/s/person	O/A @ 10 L/s/person	O/A @ 2·5 L/s/person	O/A @ 10 L/s/person
Adelaide	2·115	2·398	1·655	2·386
Kuala Lumpur	2·506	3·995	1·868	2·392

A higher limit on the chilled water flow rate through the outdoor air treatment unit is imposed by the requirement that the temperature rise of the water returned to the chiller after having passed through both the outdoor air coil and the return air coil must exceed some threshold if stable operation of the chiller is to be attained. Clearly, there is some scope to reduce the chilled water flow rate through the outdoor air unit in certain of the cases tabulated above, at the expense possibly of increasing the chilled water requirements of the terminal units, and increasing humidity levels within the zones. It is conceivable that the task of selecting an optimal chilled water flow rate could be formulated as a constrained optimization problem. A further aspect of the HDP system which has not been addressed here is that the chilled water supply temperature may in many designs be raised by one or more degrees above the conventional 6·5°C or 7°C. Again the selection of the optimal chilled water temperature could be formulated as a constrained optimization problem.

13.3.4.4. Summary.

It should be emphasized again that the study described in the above is of an exploratory nature only, and is intended primarily to illustrate the manner in which the simulation techniques developed in this thesis can be used to gain new insights into problems of practical importance. A far more exhaustive study is required to generate a concrete set of recommendations for air conditioning the building under consideration. A more extensive range of systems needs to be considered, and part-load strategies need particular attention for both conventional and HDP systems. The required studies can be undertaken using facilities which currently reside within, or can readily be added to the ZEBRA programme. In the preliminary form developed above the study does however provide valuable insight into the manner in which a number of possible design solutions for the building will respond when subjected to various load regimes.

13.4. Case 3 - Attakarn Prasit Staff Apartments, Bangkok.

The final case study considered here is the Attakarn Prasit Staff Apartments at the Australian Embassy in Bangkok. The author was involved in the preliminary studies associated with this project. The definitive design and the development of the control strategy was undertaken by Mr. Ivan Koptchev, a postgraduate student in the author's department in association with Professor R.E. Luxton. The author is indebted to Mr. Koptchev for permission to reproduce some of the results of his research in the following. The novel ideas developed in this design study will be fully described and analyzed in future publications. All were able to be accommodated within the Zebra software with only modest additions to the code. The purpose of the following is to provide a brief summary of two aspects of the design, the application of High Driving Potential concepts to a CAV system, and the development of a

control strategy for an HDP system which is capable of providing satisfactory indoor air conditions over a particularly broad range of part-load conditions. The discussion of these aspects of the design reflect on our current understanding of the practical applications of the HDP concept. As such, this section may be considered to complement the case study described above.

The Attakarn Prasit Apartments comprise two seven-storey apartment blocks in the grounds of the Australian Embassy in Bangkok. There are four apartments on each floor, a number of which are designated to be 'party' apartments. The other two areas which require air conditioning are the childrens'recreation area and the lounge bar. Each of these areas poses specific design problems. The following discussion is largely restricted to the approach taken to solving those problems specific to the lounge bar, but with suitable modifications, the techniques described are applicable to a range of similar design problems.

Commercial considerations dictated the use of self-contained constant air volume units, one for each apartment, and one each for the childrens' recreation area and the lounge bar. It was also a requirement of the design brief that the chilled water flow rates remain constant throughout the operating cycle of the plant. A preliminary set of simulations were conducted by the author on the basis of interim load estimates supplied by the contractor, and subject to the commercial requirements described above. The results of this exercise confirmed that it is practical within the commercial constraints to use HDP concepts to overcome the part-load problems associated with CAV systems, so removing from the owner the burden of the otherwise excessive energy penalty of a CAV system in a humid climate. While the results shown below are not entirely representative of the system as installed, it is worth reproducing a sample here to illustrate the concepts involved.

The system which was the basis of the preliminary investigation comprised separate outdoor air and return air coils in a common case, with the fan mounted in a draw-though configuration. Chilled water is fed in series to the outdoor air coil, and then to the return air coil, thus maximizing the dehumidification potential of the outdoor air coil which handles the more humid air stream. The chilled water flow rate through the outdoor air coil is fixed at a constant value. A three-way valve provides continuous modulation of the chilled water flow through the return-air coil as required to offset the zone load, the excess water being bypassed. Figure 13.11 shows the variation in room relative humidity with changing load for three different combinations of air flow and water flow. As the zone load decreases from the peak value, which occurs when the ambient dry-bulb temperature is 34 °C, the relative humidity of the air in the zone rises sharply at first in the manner characteristic of CAV systems. Unlike conventional systems however, relative humidity does not continue to rise unchecked with a further fall in load. A point is reached where the dehumidification potential of the dry air from the outdoor air coil is sufficient to offset the latent load on the return air, and the zone relative humidity in fact falls as the zone load decreases further.

The layout and circuiting of the air handling unit specified for the lounge bar is fundamentally similar to that described above. The outdoor air coil is 850 mm long by 24 tubes high, and 8 rows deep; the return air coil is 650 mm long by 10 tubes high and 3 rows deep. Both coils have a fin density of 6 fins per inch. What distinguishes this unit from the basic unit

considered in the preliminary studies is the features which have been incorporated as part of a comprehensive control strategy to enable the unit to maintain room conditions within acceptable limits when subject to an extremely broad range of load conditions. Ambient dry-bulb temperature may vary from approximately 18°C up to 34°C, with the occupancy varying from 60 people down to perhaps 4. The part-load control strategy may perhaps most clearly be understood by considering the stages involved in accommodating a progressively decreasing sequence of loads:

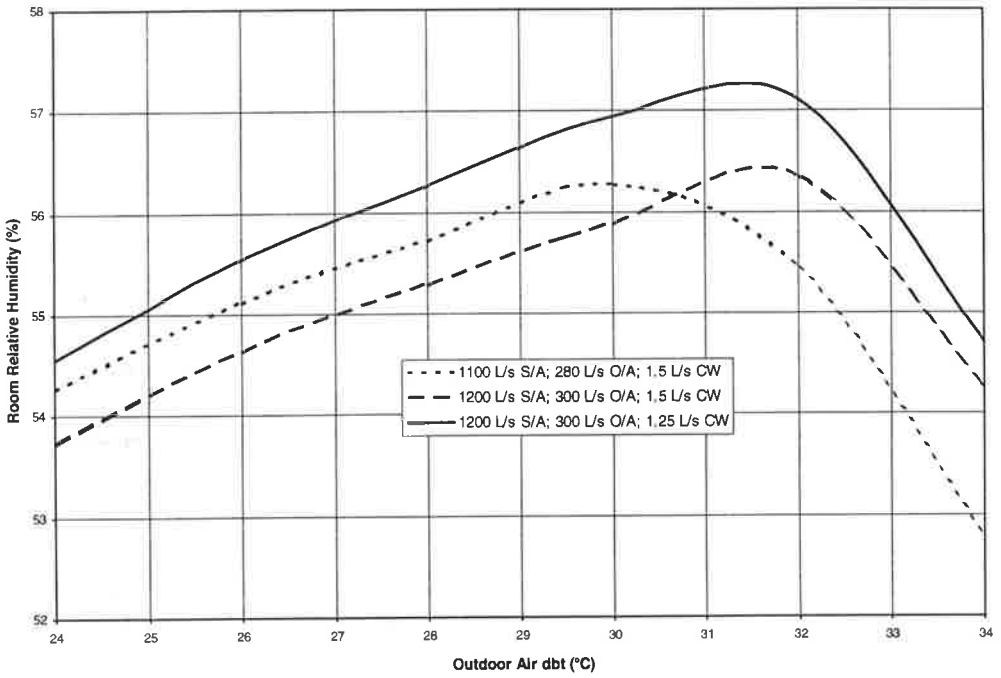


Figure 13.11. Results of a preliminary investigation of the performance of a CAV unit configured according to HDP principles for the Attakarn Prasit Apartments.

1. At full load, the outdoor air flow is 600 L/s. A fall in load is accommodated in the first instance by throttling the chilled water flow through the return air coil to maintain the dry-bulb temperature within the space at the first thermostat set point of 24°C.

2. When the chilled water velocity in the return air coil falls to 0·2 m/s[161], it is no longer practical to maintain proportional control of the room temperature by modulating the flow through the return air coil. At this point the three-way valve moves to its fully-closed position to isolate the return air coil. As the load further decreases, reheat is

---

[161] The original proposal envisaged a two-stage reduction in the capacity of the return air coil by eliminating two of the original five circuits when the chilled water velocity first falls to 0·2 m/s, and then a second pair of circuits when the chilled water velocity again falls to 0·2 m/s, the coil being finally deactivated the third time the chilled water velocity falls to this value. This strategy was proposed to extend the range over which proportional control of room temperature could be maintained by modulating chilled water flow, rather than to assist dehumidification as in cases 1 and 2. This proposal was not accepted by the contractor.

required to offset the excess cooling capacity of the outdoor air coil, which continues to receive the full quantity of chilled water, if the room temperature is to be maintained at 24°C. The quantity of reheat required is small, and is provided by recovering waste heat from the spill air using an air-to-air heat exchanger between the spill air and the relatively modest quantity of low temperature air leaving the outdoor air coil. Reheating up to a maximum of about 7·5 kW is possible by this means. Proportional control is maintained during this stage by a set of dampers which modulate the amount of spill air passing through the air-to-air heat exchanger.

3. When the amount of spill air passing through the air-to-air heat exchanger reaches a maximum of 600 L/s, proportional control of room temperature is no longer possible. A further decrease in load will cause the room temperature to fall. This trend continues until the room temperature falls to 21·5°C, which is the second set point on the thermostat. At this point, the outdoor air damper closes down to reduce the outdoor air flow to 300 L/s, and the spill air fan speed[162] simultaneously changes to reduce the spill air flow rate to match. Proportional control of room dry-bulb temperature at the first thermostat set-point of 24°C again becomes possible by modulating chilled water flow or applying spill air reheat, as appropriate. The control sequence is reset to its original state when rising loads force the room temperature up to 24·5°C.

The rationale underlying the reduction in outdoor air flow rate in stage 3 can be inferred by reference to figure I.3[163]. From this diagram it can be seen that at the full outdoor air flow rate of 600 L/s and 30 people in the lounge bar, the room temperature cannot fall below 21·5°C unless the ambient temperature drops below 20°C. With higher occupancy of the lounge bar, the reduction in outdoor air flow rate to 300 L/s will be delayed until an even lower ambient temperature is attained. Thus, in other than exceptional circumstances, the desired ventilation rate of 10 L/s is achieved at all times.

The dehumidification performance attainable using the above control strategy is as shown in figures I.1 and I.2. The former shows the room relative humidity when the full outdoor air quantity of 600 L/s is used, the latter when it is halved to 300 L/s. The lower rate is of course only applicable when the occupancy falls below 30 people. It is seen that the relative humidity is maintained within a band between 39·1% and 59·4% at all times. These levels are a great improvement over those attainable using conventional CAV systems, and are attained without the profligate expenditure of power which would be required to overcool and reheat the whole mass of the supply air to achieve acceptable zone conditions using conventional strategies.

---

[162] This is a two-speed fan.

[163] See Appendix I.

13.5. Summary.

The ZEBRA package has been conceived as a tool which will assist the designer in understanding the processes occurring within an air conditioning system, and consequently provide direction in the search for a near-optimal design solution. The present chapter describes in some detail the application of the Zebra Kernel to three design studies, one of a conceptual nature and the other two describing operational projects. The studies have been presented in the chronological order in which they were undertaken, and not only provide a demonstration of the capabilities of the Zebra Kernel, but also illustrate the interactive processes which have been involved in the HDP method of air conditioning. The understanding which accrues from studies such as those described plays a vital rôle in stimulating technical innovation, highlighting deficiencies in current understanding of various aspects of plant and system performance, and in providing guidance in the development and automation of methodologies for optimal design.

Chapter 14. Conclusions and Recommendations.

The objective comparison of the relative merits of contending design solutions for an indoor climate control problem presupposes the availability of a means of modelling the performance of the various systems which the designer might wish to consider. As a minimum, the designer should be able to extract from the modelling process a measure of the zone conditions which may be maintained by a system in the face of anticipated variations in zone loads and other operating parameters, an estimate of the power consumption required to maintain those conditions, and a means of identifying an appropriate control strategy. Given that this information can be obtained, the designer can estimate the capital and operating costs associated with promising candidate design solutions, and recommend a configuration and control strategy which will enable a prescribed set of air quality indices to be maintained, while at the same time minimizing the cost of owning and operating the system. The modelling tools used should be efficient, flexible and easy to use. They should also be established on a firm physical foundation, and conceived within a framework which takes a holistic approach to the entire specification and design process. When provided with such tools, the designer is encouraged to experiment, and gains new insight into the factors which effect the air conditioning system performance. The current study has been undertaken as a component of an ongoing project to develop such a set of tools, and to establish procedures to enable maximum productivity to be gained from their use. This is necessarily an evolutionary process, but the tools developed today to increase productivity within a semi-automated and partially fragmented design environment should be conceived with the clear foresight that they will in the near future be required to operate within an environment which seeks to obtain much closer integration between the various facets of the design process. In this concluding chapter of the thesis, it is timely to review the progress which has been made to date in meeting the above objectives, and our immediate priorities for future developments. In this respect, it will be convenient in the following to examine three aspects of the modelling programme, namely:

1. The position of the Zebra Kernel as a component of an integrated environment for the design and analysis of air conditioning systems.

2. The need for further experimental research to support the modelling effort.

3. The need to develop further modelling capabilities within the Zebra Kernel.

14.1. An Integrated Design and Analysis Environment.

The case studies described in the preceding chapter indicate clearly the manner in which the ZEBRA programme may be used both as a design tool, and as a means of establishing general principles and recommendations for general classes of air conditioning problems. In each case the *need* to consider system operation over a broad range of part-load conditions was clearly demonstrated. The reader familiar with the modelling tools available in most, and probably all design offices, will recognize that the thorough analyses involved in those studies could only be undertaken by traditional means at the expense of months of manual computation even with

the assistance of conventionally available software. Such time investment would not be feasible in a normal design office. Use of the Zebra Kernel in its present form has the potential to increase greatly the productivity of the design engineer by providing a means whereby the characteristics of a range of prospective air conditioning design solutions can be rapidly evaluated for their response to both peak and part-load conditions. On this basis cooling coils can be selected, and control strategies devised to handle potentially difficult part-load conditions. Nevertheless, certain routine data processing tasks associated with these studies have been mundane and labour intensive, and are prime candidates for automation. This can be undertaken most conveniently if the Zebra Kernel runs as a process within a multitasking environment such as Windows, as has always been envisaged (see Chapter 2). The task of converting the code to run as a Windows process, and establishing the protocols whereby the Zebra Kernel will be able to communicate and interact with other Windows processes therefore assumes a very high priority. The other tasks with which the Zebra Kernel will be required to interact may be standard software packages, such as CAD packages, spreadsheets, databases, and programmes which have either been written or modified expressly for use within the general ZEBRA framework, such as user interfaces, graphics and reporting programmes, and load calculation programmes. The manner in which these tasks are integrated into the ZEBRA framework will ultimately determine the useability of the package. Certain aspects of the integrated environment are considered below.

1. A Windows based user interface for the ZEBRA programme is presently being developed. This will provide the user with a means by which to communicate interactively with the Zebra Kernel, and with other components of the ZEBRA environment. It will also provide a means of accessing and editing existing system specification files which have been stored in the format outlined in section 2.4. These are currently created and edited using a standard text editor, a procedure which requires an intimate knowledge of the Zebra file structure.

2. The runtime log file (section 12.3) is the primary repository for the results of a simulation exercise. It is however to be regarded as an interim, and usually volatile data store only, from which information may be extracted for more permanent storage in a database (see below), or for the generation of reports and graphical output. Three basic types of report are envisaged. By default, printed reports will be formatted according to a standard style. It is also desirable that a (limited) facility be provided to enable the user to define and store custom styles, whereby only those parameters of particular interest to the user may be selected for output. Finally, a facility will be provided to enable data to be transferred directly from the log file to a spreadsheet programme such as Excel.

   Data will also be extracted from the runtime log file for graphical output. There is a clear need for a facility to plot the process lines representing the results of a simulation run on a screen-based psychrometric chart. Provision will also be made to generate other types of plots, such as those presented in Appendix E. Certainly, these can be generated via the medium of a spreadsheet package, but where such plots need to be generated on a routine basis, this procedure becomes quite tedious. The graphics generation routine will make provision for graphs to be stored to disc or printed, and

in the longer term to be linked to other documents using the *Object Linking and Embedding* (OLE) mechanism.

3. In current practice, the results of a load analysis performed using a load calculation programme need to be transcribed manually into a form suitable for input to ZEBRA when performing a simulation covering a time sequence of load conditions, as was done in Case Study 2 in Chapter 13. Most if not all extant load calculation programmes generate output files reporting on an extensive range of parameters, most of which are of little relevance to the user concerned with simulating air conditioning plant performance. Machine extraction of those parameters which are of interest, namely the space loads, is by no means straightforward. Generation of an auxiliary output file in an appropriate format would certainly improve the situation, but a more intimate degree of interaction is really desirable. What is required is the ability to establish a link between the load calculation programme and the Zebra Kernel, possibly via an intermediary. This is not simply a matter of convenience. Such a mechanism is necessary for the accurate modelling of a class of problems which are of growing interest; those in which proportional control of room temperature is not possible at some point, either as the result of a conscious design decision, or because of poor equipment selection. Under such conditions the room temperature is not held at the thermostat set point, and to find the equilibrium zone condition, it is necessary to be able to establish the functional relationship between room dry bulb temperature and space load. Clearly, a Windows-hosted load calculation programme is required. This could possibly be developed by modifying one of the existing codes, such as TEMPER.

4. A database system is projected as an essential and integral component of the ZEBRA package. A database is an ideal means of storing and retrieving information relating to equipment items such as fans, and possibly also information relating to components, such as pipe and duct fittings. In the case of such entities there is little argument as to exactly *what* should be stored, and the task of designing the database is relatively straightforward.

   The project database is potentially an extremely powerful tool for archiving and maintaining information relating to an air conditioning design project. Potentially, this could act as a repository for a quite diverse body of information[164], including system specifications and performance data, load data, and information of a more general nature, such as drawings, costing information and time schedules. There is however a danger that the database could become unwieldy if close attention is not paid to its design, and to the establishment of procedures for its use and maintenance. During the early stages of a project, the designer will probably wish to trial a number of possible design solutions and to archive the specifications and performance data for future comparison. The origins and status of the data must be clearly identified if future

---

[164] In fact, most of the items which follow would probably *not* be stored directly in the database. Rather, the database would store the names of files where this information could be found.

confusion is to be avoided. The investigation of suitable means of managing the data associated with a design project is potentially an important area for research.

The structuring and composition of the climate database is more straightforward, but still raises questions. Obviously, the database will be organized according to location, and will include such basic data as the altitude and geographic coordinates of the site. Meteorological data will be stored in raw or processed forms, or both. Monthly climate summaries are an obvious candidate for inclusion, as is the design locus, which figures so prominently in the group's work at the present time. 'Test Reference Year' data sets, comprising 8760 hourly sets of climatic data which are declared to be representative of the location according to some agreed criterion, are commonly used as a standard for estimating annual energy usage (Clarke, 1985). Yoshida and Terai (1992) advocate the use of time series analysis both as a means of compacting the data storage, and as a means of deriving a record which is more truly representative than that for a 'typical' year. A data set which is appropriate for estimating energy usage will probably not form a suitable basis for the actual air conditioning design. Von Thun and Witte (1991) suggest the use of a 'design weather year'. These matters suggest areas for further investigation of how best to select climate data which are suitable for inclusion in a climate database intended for use in the design and analysis of air conditioning systems, and of the tools which are required to access and maintain such a database.

5. In Chapter 9 a methodology was developed for calculating the temperature and pressure distributions in a duct system. Although not yet fully operational, it is envisaged that this will be an important component of the ZEBRA package. It was originally envisaged that this component would form an integral part of the Zebra Kernel. It now seems preferable that it should be implemented as a self-contained component which can either be used in a standalone mode or accessed by the Zebra Kernel via a link. The duct model is constructed in computer memory from a flat file using a recursive algorithm which is described in detail in section 9.4.5. The flat file itself can be created using a text editor. This procedure is suitable for use during the development phase but is too labour intensive for routine operational use. Suitable alternative tools will need to be developed. As was mentioned in Chapter 9, the data structures used to implement the duct model are, with minor modifications, suitable for use with the T-method for the optimal design of systems (Tsal et al., 1988a, b). The possibility of adapting this component for use as a tool not only for analysis for also for design should be investigated at some future date.

6. The Zebra Kernel itself has been designed with the intention that, once loaded with a set of system specifications, it may be accessed in several different ways. Most obviously, a user at a keyboard may conduct an interactive session with the loaded Zebra Kernel. Alternatively, a set of instructions may be supplied to the Zebra Kernel in the form of a metafile. The analyses involved in Case Study 2 of Chapter 13 were performed in this manner. Once the Zebra Kernel has been ported to a multitasking environment it will become possible for another running process to initiate an interactive session with the loaded Zebra Kernel. This facility is mainly intended to

provide access to the Zebra Kernel by expert system and optimization software. The identification of aspects of the system design which are amenable to optimization, and the formulation of the associated optimization problems, is still in an exploratory stage.

14.2. Experimental Research.

In developing the Zebra Kernel, every effort has been made to ensure that the algorithms used are not only computationally efficient, but that they are firmly based on fundamental scientific knowledge and sound engineering principles. Necessarily, a strong dialogue has been established between our modelling programme, and our programme of fundamental research into the principles upon which a *science* of air conditioning can be founded. This arrangement operates to the benefit of both programmes. The experimental programme provides data which is essential to the modelling process, but equally the insights gained from modelling of air conditioning processes are leading to the identification of areas where further research is needed.

Considerable work has been undertaken in the past to establish a firm basis for the testing and rating of cooling coils (Sekhar, 1990). However, our database currently only contains correlations for wet and dry coils one, two and four rows deep, and having a tube diameter of ⅝" and a fin density of 6 fins per inch. Support for coils two and four rows deep, and having fin densities of 9 and 12 fins per inch, has been added by synthesizing data[165]. The accuracy of the correlations derived for these latter coils has not been tested, although there is reason to believe that the accuracy is fairly high. The lack of support for single row coils in these higher fin densities seriously restricts the range of coil geometries which may be simulated in practice. There is an urgent need to extend our test matrix, ideally to provide a universal correlation, or series of correlations, which will permit the simulation of ⅝" tube coils covering all depths from one to six rows, and fin densities from 4 to 14 fins per inch. Extension to chilled water coils with ½" diameter tubes and DX coils with ½" diameter and ⅜" diameter tubes is also needed (see also item 2 below). It is to be hoped that fundamental research into the nature of the condensation process in cooling coils which is currently in progress will provide the theoretical understanding needed to derive the appropriate correlations, allowing us to restrict testing to a sparse subset of the total matrix.

There is also a need for improved experimental data relating to air pressure drops through cooling coils. We currently use McQuiston's correlations (see section 7.7), and comparisons with results given by a proprietary coil selection programme are good. However, it has recently been brought to our attention that a more recent version of the same proprietary programme predicts pressure drops which are substantially lower than those obtained from the

---

[165] In this experiment, a series of synthesized 'experiments'were run using data output from a proprietary coil selection programme, and the results processed in the usual way. Our experience when using this programme to predict the performance of coils having a fin density of 6 fins per inch, for which we do have a substantial body of experimental data, leads us to believe that the resulting correlations will be acceptable.

earlier version. In any case, McQuiston's expressions are unwieldy, and the physical basis for some of them is not altogether clear.

Other areas in which we can foresee the results of fundamental research feeding back into the modelling process include the following:

1. Estimation of entrainment rates for induction systems. The functional relationships upon which these depend need to be established by numerical modelling using a commercial CFD package and verified by definitive experiment.

2. Fundamental experimental and computational research on various aspects of the performance of DX system components is required to support the extension of the ZEBRA modelling methodology to handle such systems. The most immediate requirement in this regard is that of establishing a set of correlations for the heat transfer and pressure drops for refrigerants in an evaporator coil.

3. Numerical modelling of pressure losses in duct fittings, and of fan system effects. As was noted in Chapter 9, the data in the ASHRAE database has been collated from a disparate collection of sources, and varies greatly in its style of presentation, and also one suspects in accuracy. Numerical modelling using a CFD package offers the potential to establish a more uniform sytem of presentation, to broaden the range of geometries which can be modelled, and also to study the influence of upstream fittings on the pressure drop through a particular fitting.

14.3. Additional Modelling Capabilities.

In its current form, the Zebra Kernel is able to model a wide range of air conditioning system configurations. In designing the software, it was recognized that there would be an ongoing need to enhance and broaden the capabilities of the model, and a great deal of emphasis has been placed on developing data structures which contain the flexibility necessary to facilitate this process. The use of the object model has assisted greatly in this regard. Future enhancements envisaged for the model include the following:

1. Simulation of DX systems. The basis for a framework for modelling DX system performance has been established in Chapter 11. There is need to continue this work, in conjunction with a programme of basic research (see above).

2. Modelling of heating systems. The basic procedures for rating cooling coils described in Chapter 7 are equally applicable to coils using hot water as a cooling medium, with the simplification that we no longer need to consider condensation. The software modules currently used to simulate heat transfer do include some aspects which are

specific to the cooling situation, but these restrictions can easily be removed to permit the heating situation to be modelled also[166].

3. Perimeter induction units experienced a considerable degree of popularity during the 1950's, 1960's and 1970's. These systems offer the advantage that they occupy less space than conventional units. For ceiling-mounted induction the small size allows the depth of plenum to be reduced, thus reducing the slab-to-slab height and hence increasing the number of floors of rentable space in the building within a prescribed height. However, as installed they are noisy, and offer poor energy efficiency. The latter factor in particular led to a decline in their popularity during the energy conscious 1970's. Many of these systems are still in use, and since the buildings involved were custom built to accommodate the induction systems, space limitations preclude the possibility of retrofitting more conventional technology. Recent (as yet unpublished) work in our laboratories, supported by field trials, indicates that by redesigning the induction nozzles, entrainment rates can be enhanced, allowing primary air supply pressures to be reduced and leading to both a substantial reduction in noise level and an increase in efficiency. Use of HDP principles potentially offers a means of effecting further improvements in efficiency. For these reasons, the addition to the Zebra Kernel of a facility to model induction systems has become highly desirable.

4. The basic system model, as described in Chapter 6, offers the capability of modelling a wide range of load and control situations. A class of problem which cannot yet be handled is that in which proportional control of room temperature is precluded. Algorithms to solve for this case can be developed, but their implementation will need to wait until the appropriate links to a load estimation programme have been established (see section 14.1 above). A related problem, and one for which a solution can be implemented now, is that in which a space is subject to humidity control, but the cooling coil lacks the capacity to deliver the necessary overcooling. In this case, it is desirable to be able to estimate the room relative humidity which the system can sustain. The ability to model this situation will be incorporated into the Zebra Kernel in the near future.

5. The modelling effort to date has concentrated on the air side of the system. In future, it is intended to extend this work to cover the water side also. Pipe networks can be modelled using the methods developed in Chapter 9. A chiller model will also be developed.

6. Waste heat recovery through the use of air-to-air heat exchangers has recently become an important component of the HDP control strategy (see section 13.4). The air pressure drop through such units has implications for the energy consumption and

---

166 In fact, since we are no longer concerned with finding the wet and dry surface areas of the coil, a fundamental operation in the simulation of cooling coils, it will probably be more efficient to write a separate top-level module for the heating situation, rather than attempting to produce a universal module. Needless to say, most of the lower level modules can be shared.

pressure balance for the system, while efficiency determines the amount of heat transfer which is possible. It is desirable that the Zebra Kernel be provided with the means to estimate these parameters as an integral component of the system model.

14.4. Conclusions.

The thesis has described the architecture of an integrated design and analysis environment for air conditioning systems. The Zebra Kernel which, as its name implies , is at the heart of this environment, provides a structured framework within which the presently known physics of the processes occurring within an air conditioning system can be incorporated to provide a rigorous simulation of the system and its component plant items. In its present form the Zebra Kernel is a robust and proven tool which incorporates the ability to model a wide range of air conditioning system configurations. The physical bases and solution algorithms which underlie the modelling methodology embodied in the Zebra Kernel have been described. It has been demonstrated how the use of the modelling tool has the potential to facilitate the task of the air conditioning designer greatly when seeking a solution to an indoor climate control problem. This is achieved not only by automating familiar aspects of the design cycle, but also by providing the designer with new insights into the thermodynamic and transport processes involved in an air conditioning system. Future work may be expected to proceed in two broad directions. Development of the overall framework within which the Zebra Kernel is designed to operate will make the code more accessible and so facilitate its acceptance by industry. Fundamental research together with the continuing application of the modelling strategy to a range of applied problems, including studies which on the one hand may be of a conceptual nature and on the other hand may have commercial significance, will serve both to verify and strengthen the basis for the modelling strategy, and will encourage enhancements and extensions to the basic model to increase further its value and range of applicability.

Appendix A. Coil Simulation Plots.

The following plots complement the text of section 13.1, which contains a full description of the cases considered. Briefly, the coil configurations used in simulations used to generate these plots were as follows:

A.1. A simple four-row coil.

A.2. Two coil components with the chilled water flow to each circuited in parallel, as per the arrangement of figure 8.1. (Section 13.1, Case 1).

A.3. Two coil components circuited in a counterflow arrangement with respect to the chilled water flow, as per the arrangement of figure 8.2. (Section 13.1, Case 2).

A.4. As per the preceding case, but with three coil components circuited in a counterflow arrangement with respect to the chilled water flow. (Section 13.1, Case 3).

A.5. A coil subject to a changeover involving deactivation of circuits, as per the arrangement of figures 8.3 and 8.4. (Section 13.1, Case 4).

Figure A.1.a. Cooling capacity vs. CW velocity for an air face velocity of 1·5 m/s.

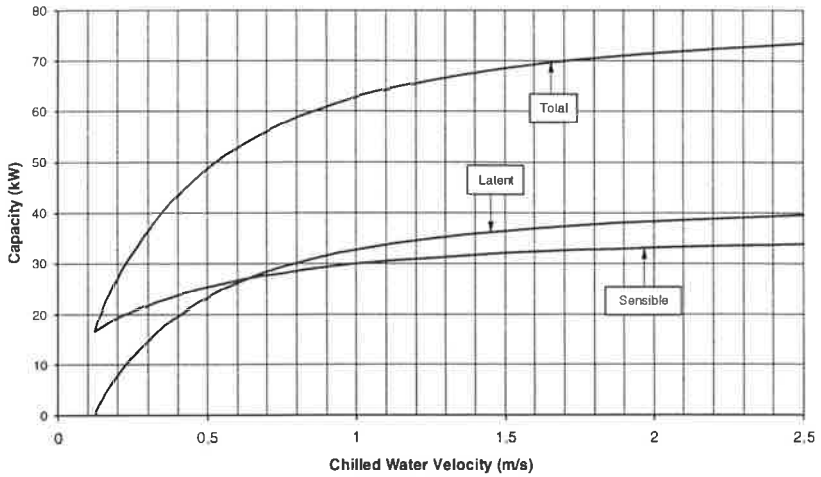


Figure A.1.b. Sensible heat ratio vs. CW velocity at air face velocities 0·5 m/s to 2·5 m/s.

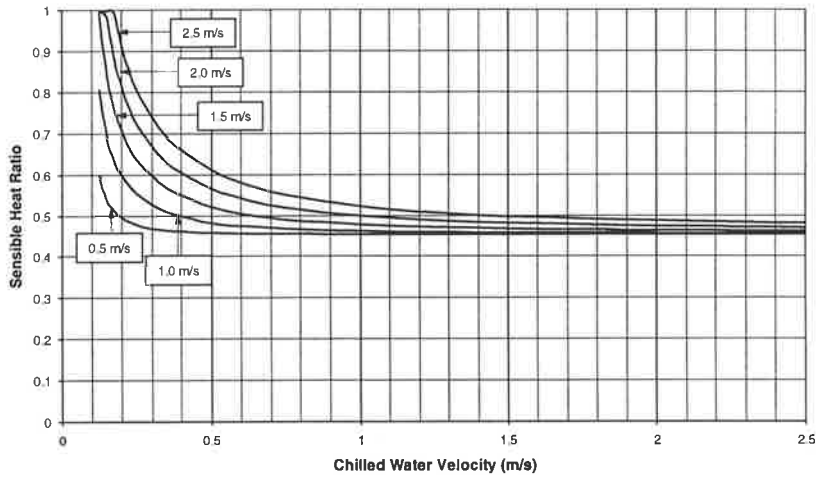


Figure A.1.c. Total capacity vs. CW velocity at air face velocities from 0·5 m/s to 2·5 m/s.

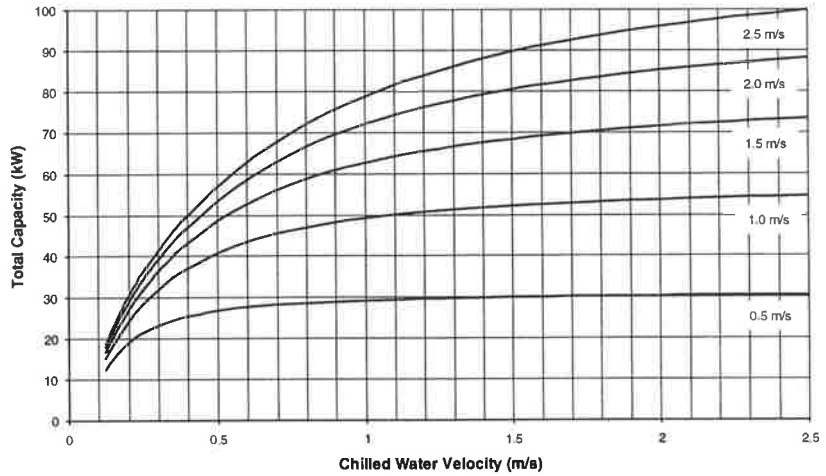


Figure A.1.d. Leaving air db temperature vs. CW velocity at air face velocities from 0·5 m/s to 2·5 m/s.

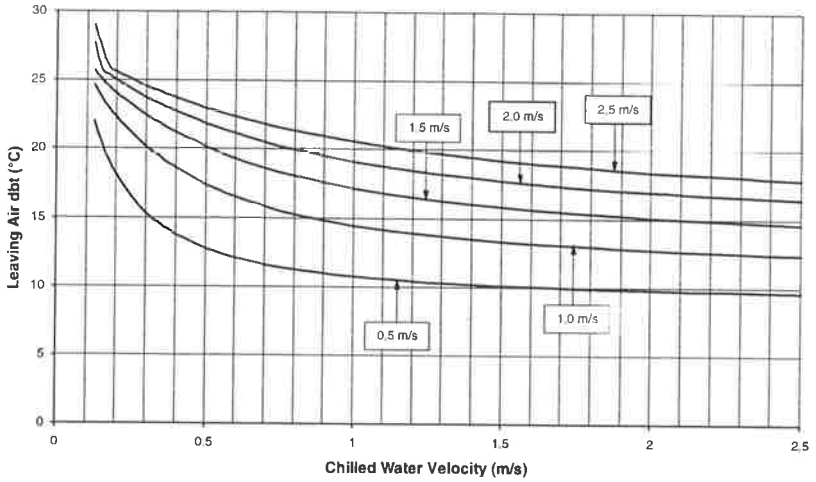


Figure A.1.e. CW temperature rise vs. CW velocity at air face velocities from 0·5 m/s to 2·5 m/s.

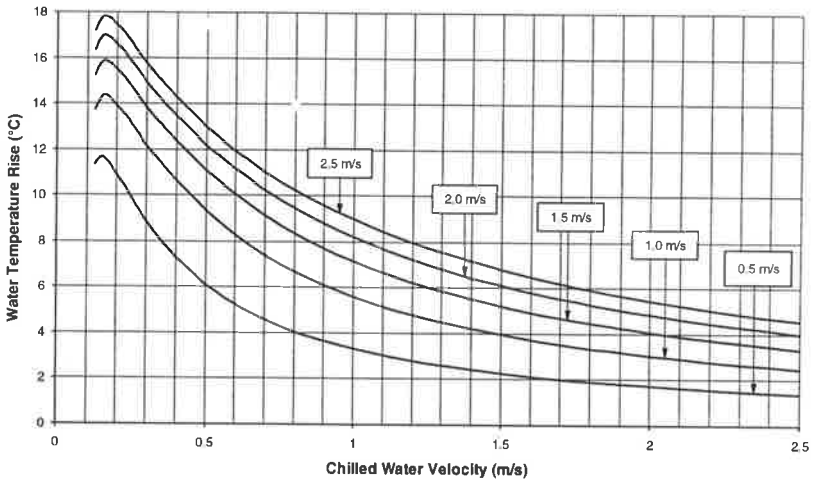


Figure A.1.f. Surface wetness fraction (A_w/A_o) vs. CW velocity at air face velocities from 0·5 m/s to 2·5 m/s.

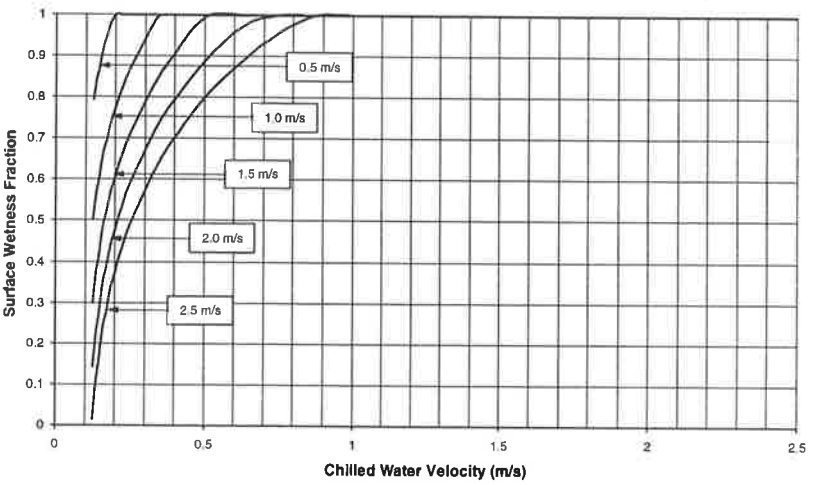


Figure A.1.g. Water pressure drop vs. CW velocity for an air face velocity of 1·5 m/s.

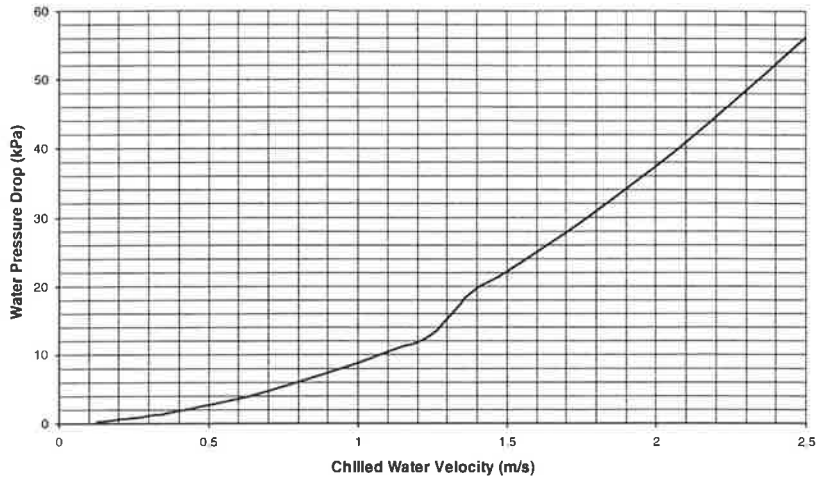


Figure A.1.h. Air pressure drop vs. CW velocity for an air face velocity of 1·5 m/s.

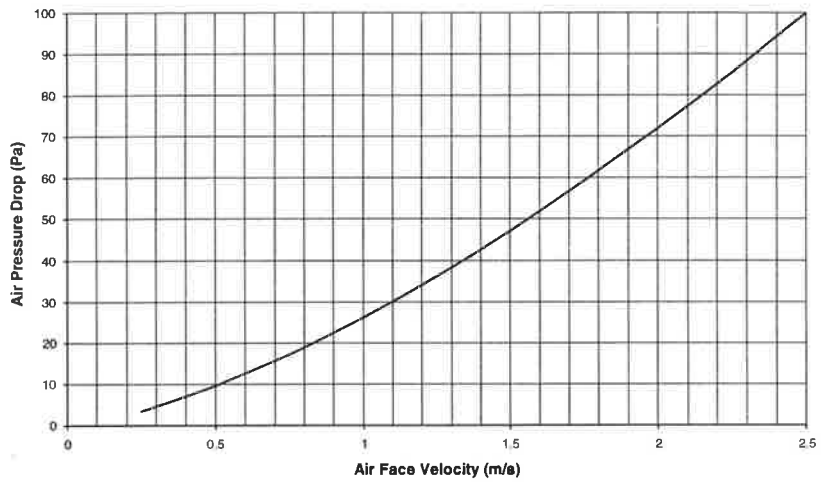


Figure A.2.a. Total capacity vs. CW velocity. Air face velocity = 1·5 m/s.

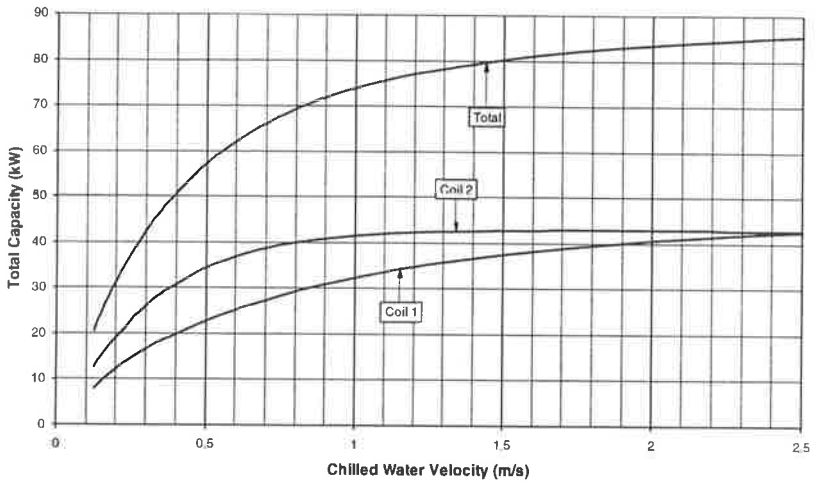


Figure A.2.b. Sensible capacity vs. CW velocity. Air face velocity = 1·5 m/s.

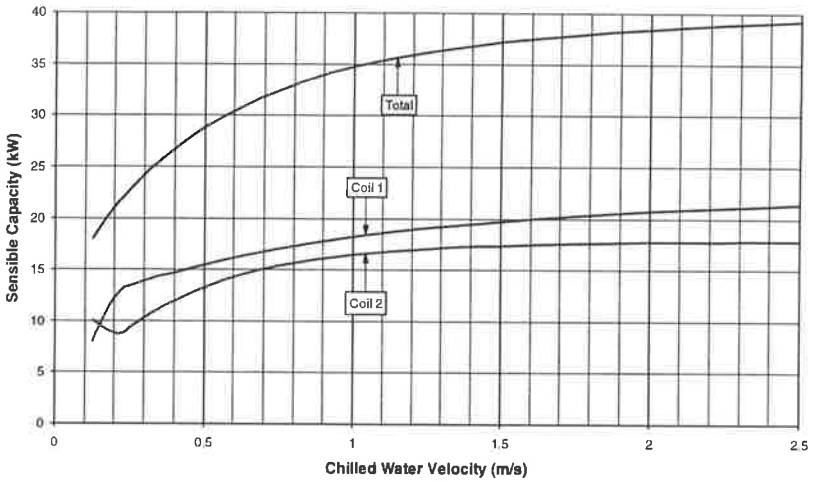


Figure A.2.c. Latent capacity vs CW velocity. Air face velocity = 1·5 m/s.

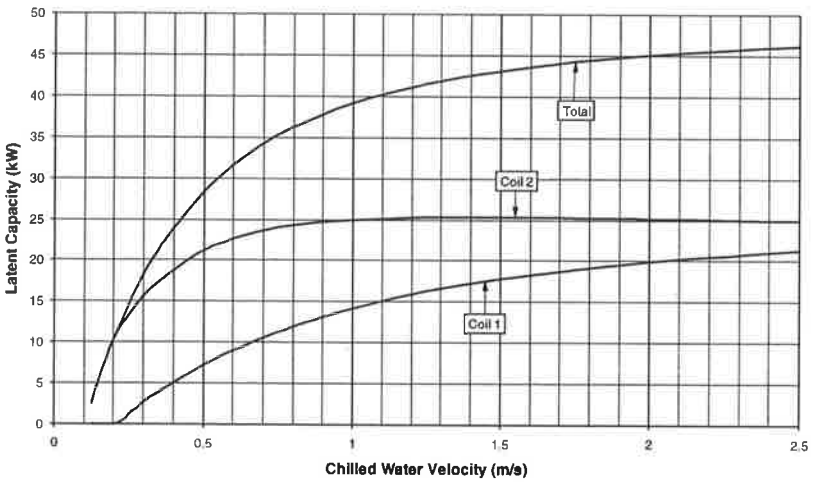


Figure A.2.d. Sensible heat ratio vs. CW velocity. Air face velocity = 1·5 m/s.

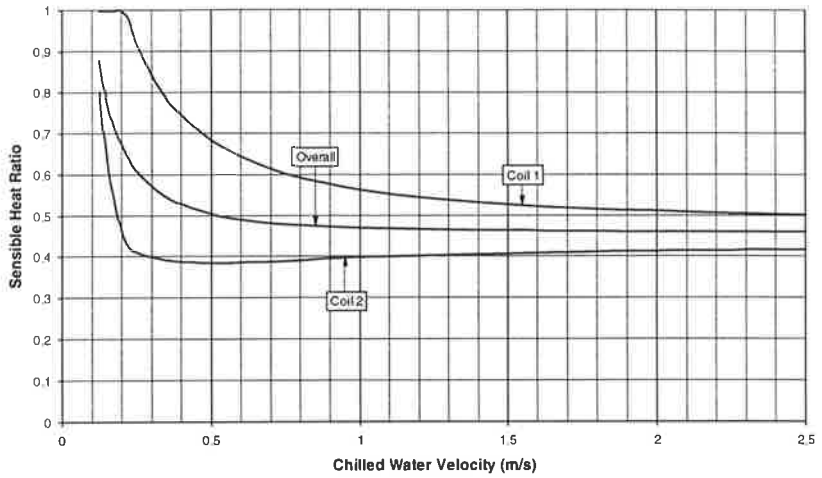


Figure A.2.e. Db temperature at various points in the coil vs. CW velocity. Air face velocity = 1·5 m/s.

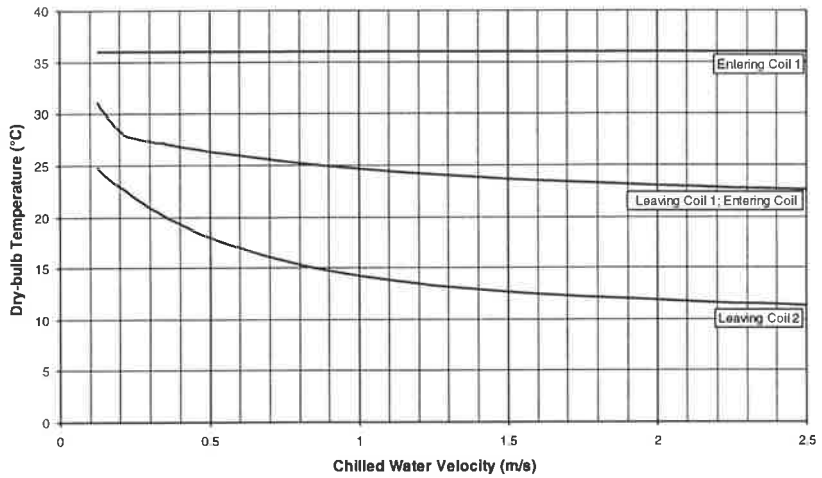


Figure A.2.f. Humidity ratio at various points in the coil vs. CW velocity. Air face velocity = 1·5 m/s.

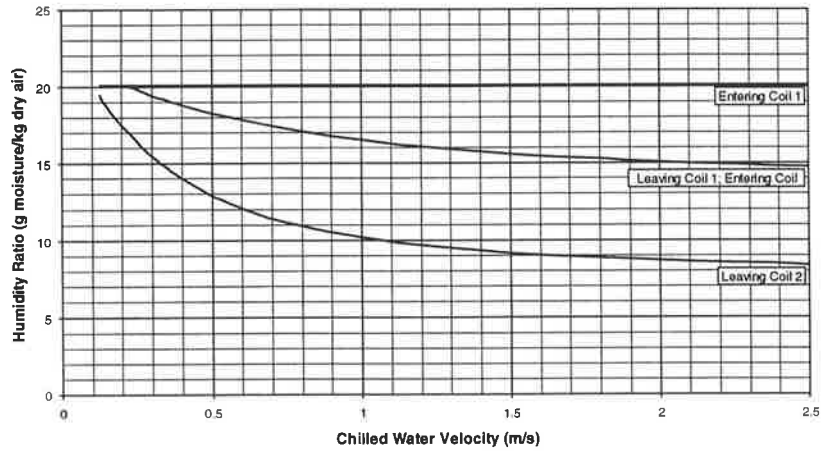


Figure A.2.g. Water pressure drop vs. CW velocity. Air face velocity = 1·5 m/s.

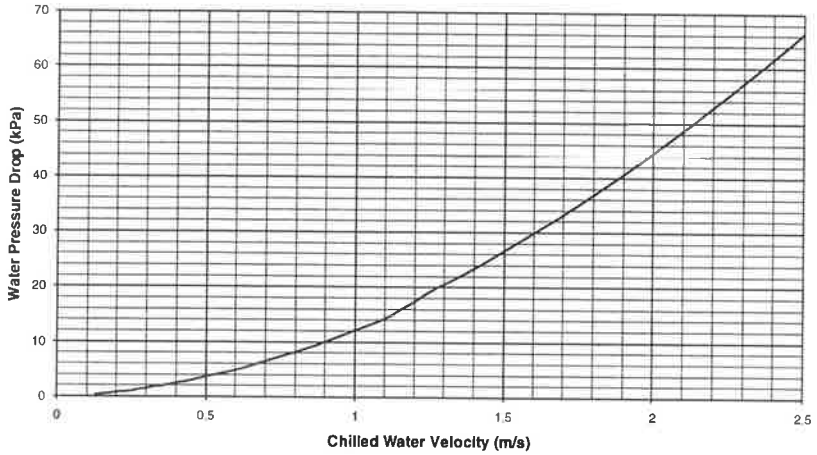


Figure A.2.h. Water temperature rise vs. CW velocity. Air face velocity = 1·5 m/s.

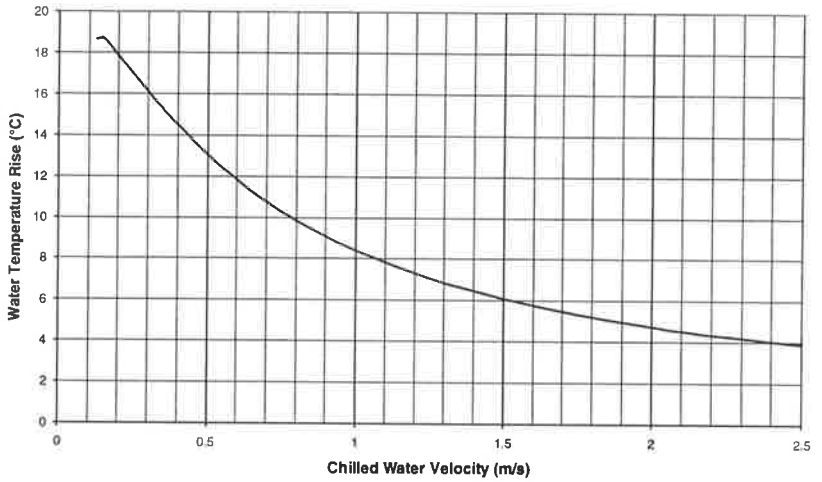


Figure A.3.a. Total capacity vs. CW velocity. Air face velocity = 1·5 m/s.

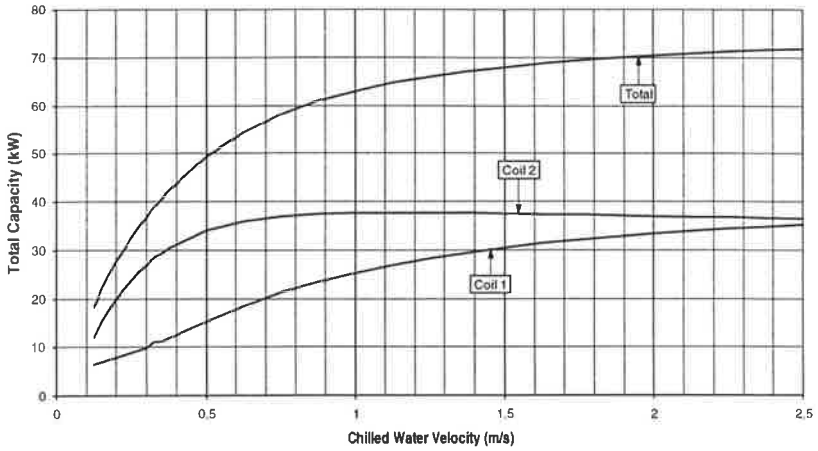


Figure A.3.b. Sensible capacity vs. CW velocity. Air face velocity = 1·5 m/s.

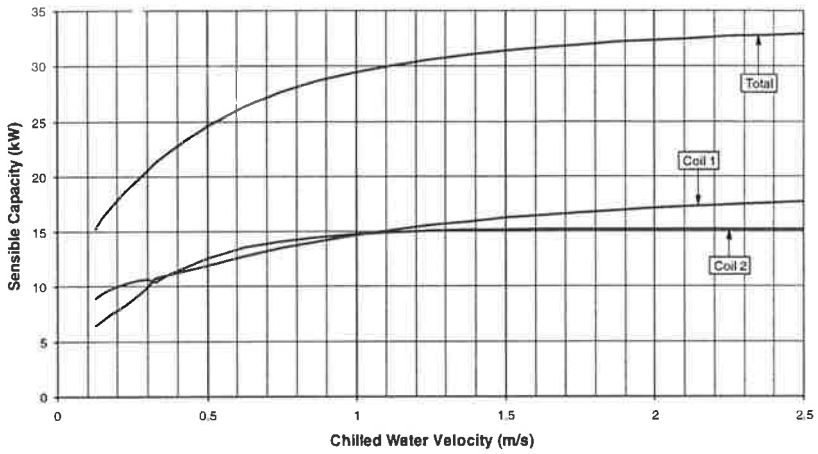


Figure A.3.c. Latent capacity vs. CW velocity. Air face velocity = 1·5 m/s.

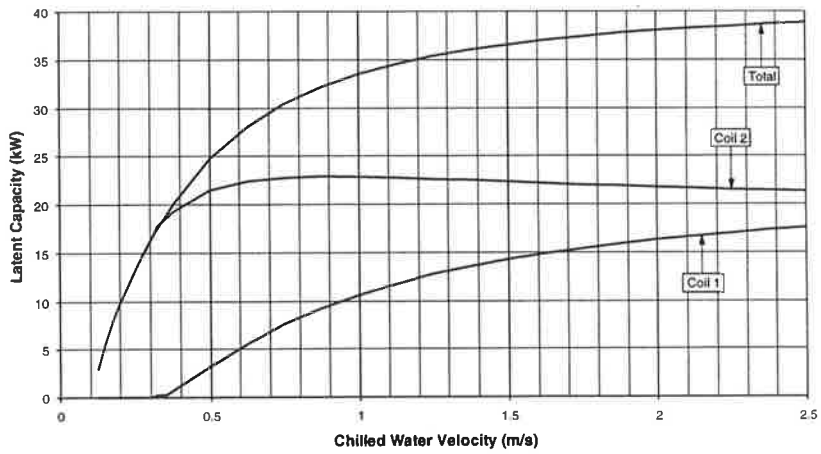


Figure A.3.d. Sensible heat ratio vs. CW velocity. Air face velocity = 1·5 m/s.

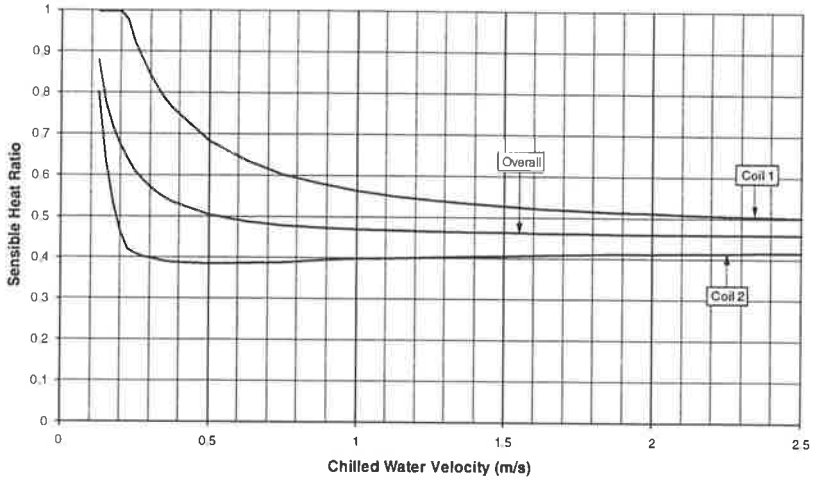


Figure A.3.e. Db temperature at various points in the coil vs. CW velocity. Air face velocity = 1·5 m/s.

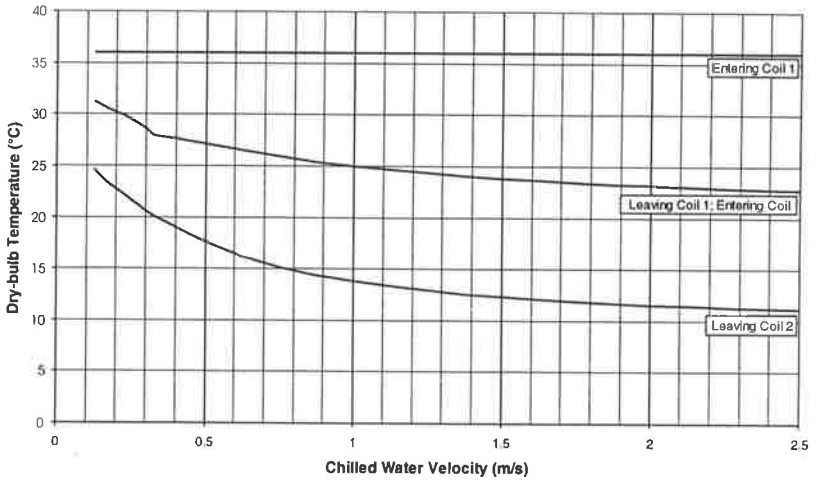


Figure A.3.f. Humidity ratio at various points in the coil vs. CW velocity. Air face velocity = 1·5 m/s.

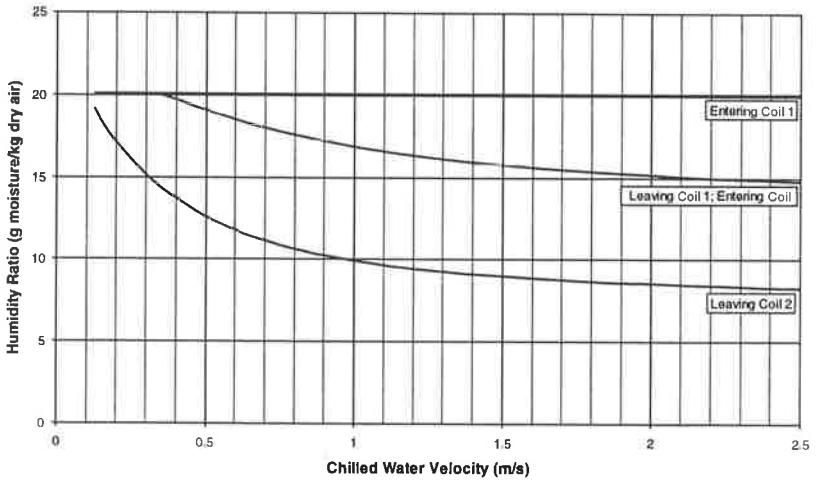


Figure A.3.g. Water pressure drop vs. CW velocity. Air face velocity = 1·5 m/s.

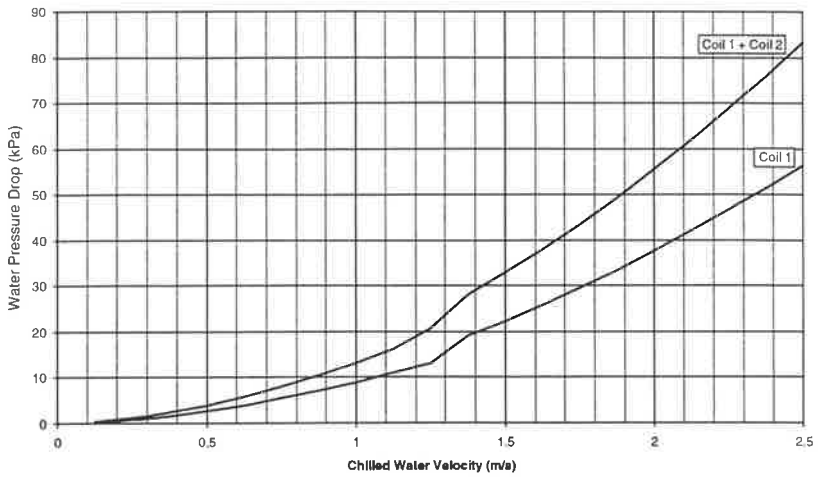


Figure A.3.h. Water temperature rise vs. CW velocity. Air face velocity = 1·5 m/s.

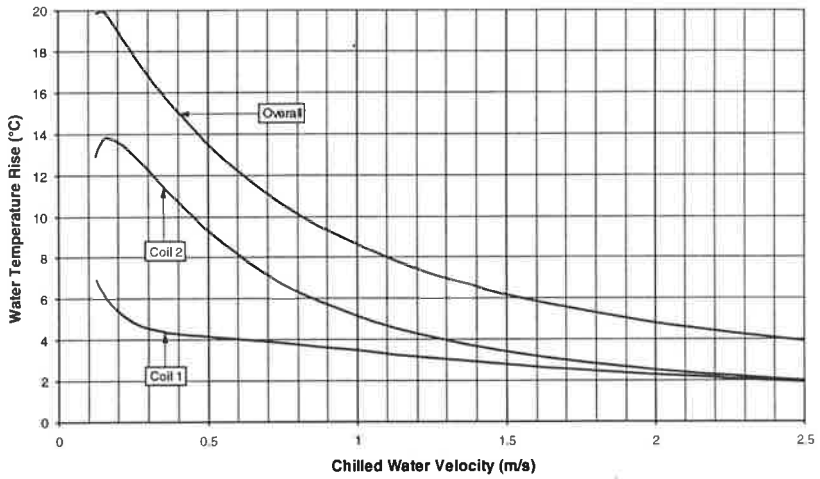


Figure A.4.a. Total capacity vs. CW velocity. Air face velocity = 1·5 m/s.

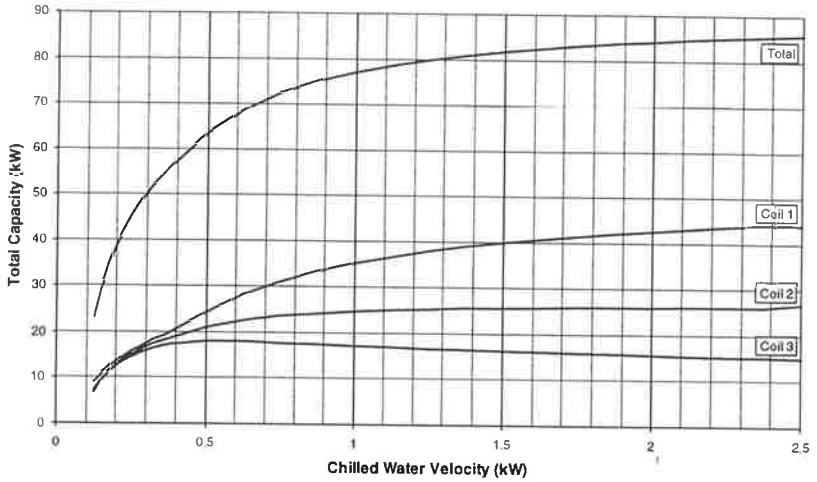


Figure A.4.b. Sensible capacity vs. CW velocity. Air face velocity = 1·5 m/s.

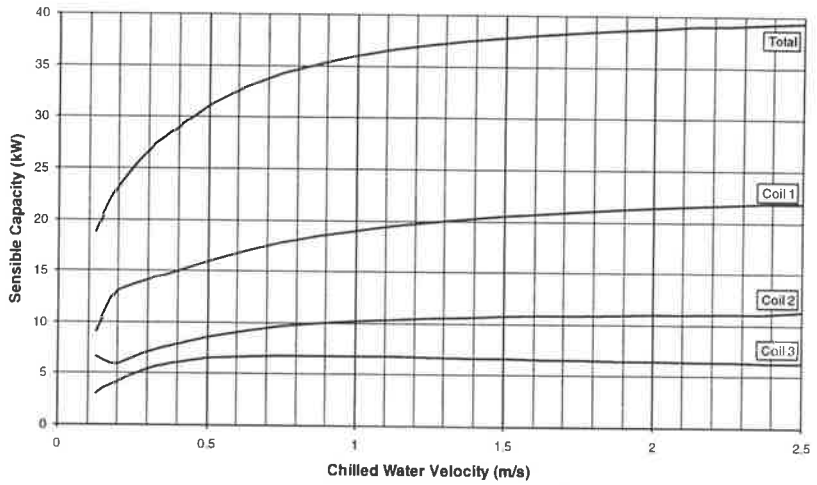


Figure A.4.c. Latent capacity vs. CW velocity. Air face velocity = 1·5 m/s.

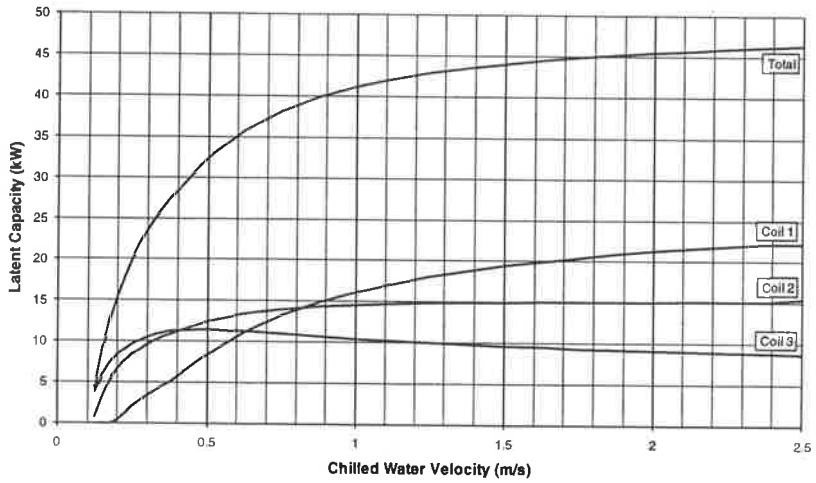


Figure A.4.d. Sensible heat ratio vs. CW velocity. Air face velocity = 1·5 m/s.

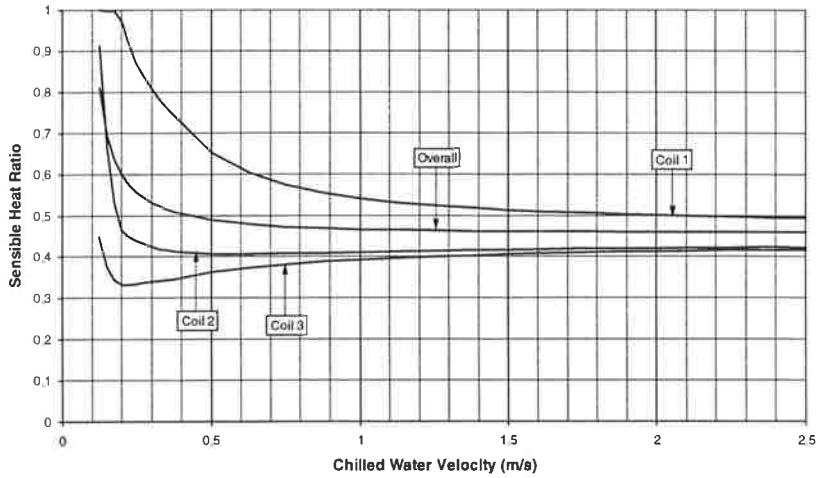


Figure A.4.e. Db temperature at various points in the coil vs. CW velocity. Air face velocity = 1·5 m/s.

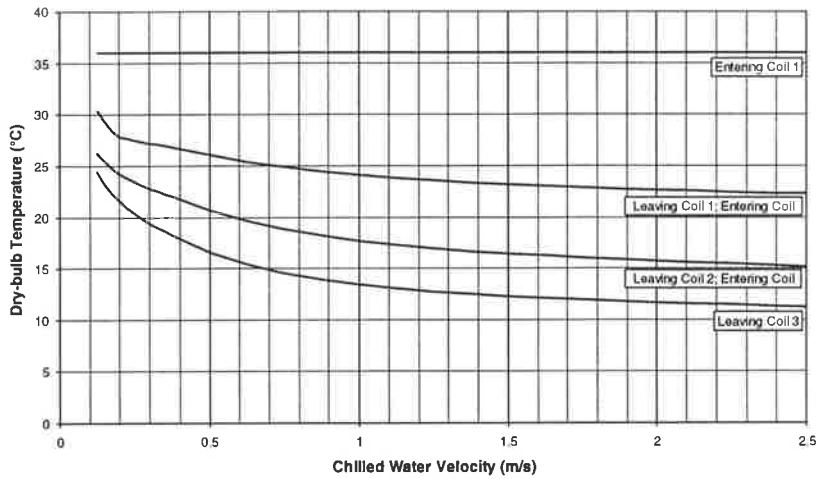


Figure A.4.f. Humidity ratio at various points in the coil vs. CW velocity. Air face velocity = 1·5 m/s.

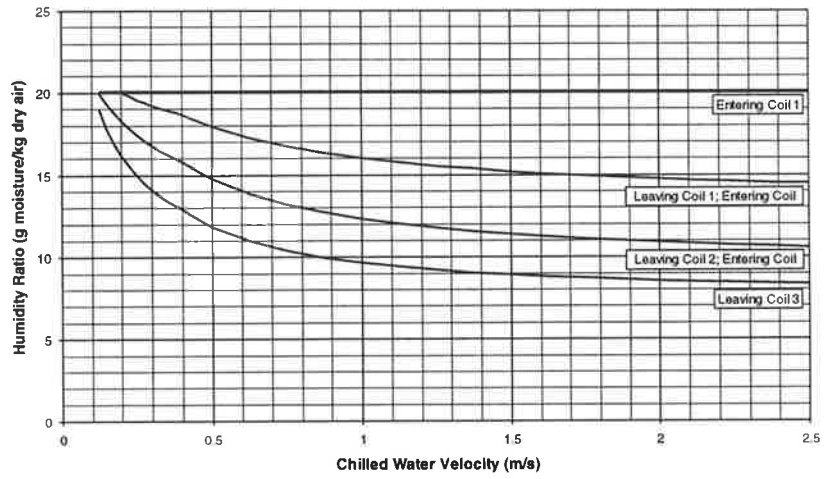


Figure A.4.g. Water pressure drop vs. CW velocity. Air face velocity = 1·5 m/s.

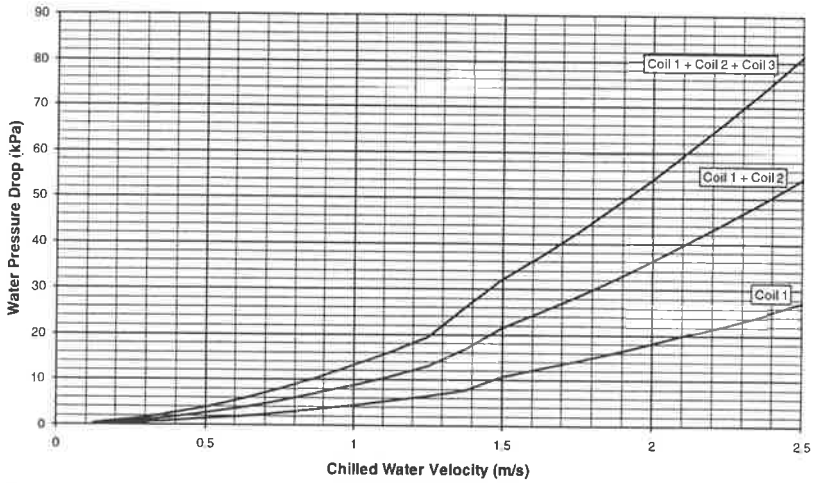


Figure A.4.h. Water temperature rise vs. CW velocity. Air face velocity = 1·5 m/s.

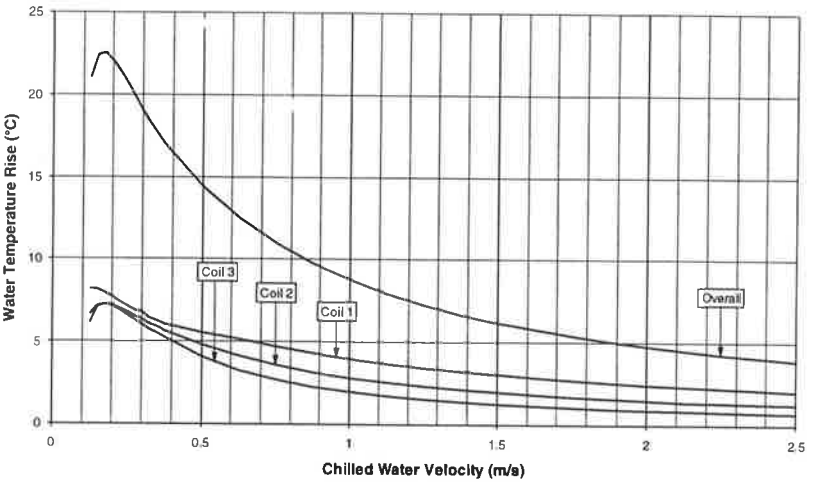


Figure A.5.a. Total capacity vs. CW velocity. Air face velocity = 1·5 m/s.

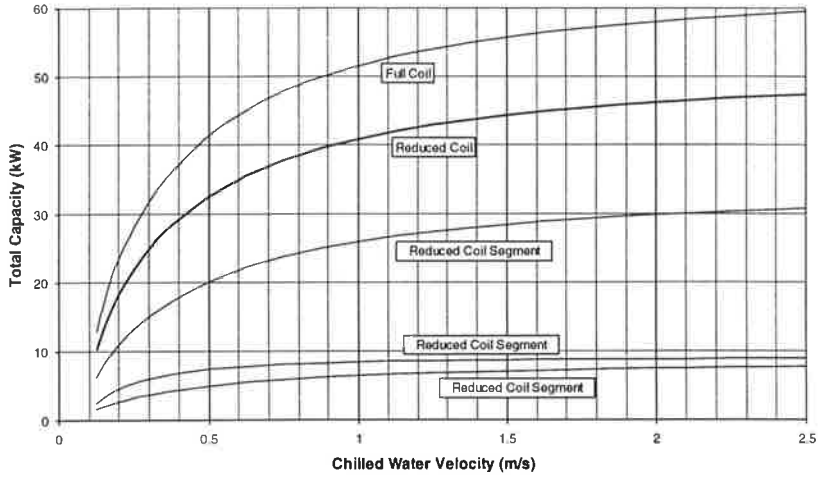


Figure A.5.b. Sensible capacity vs. CW velocity. Air face velocity = 1·5 m/s.

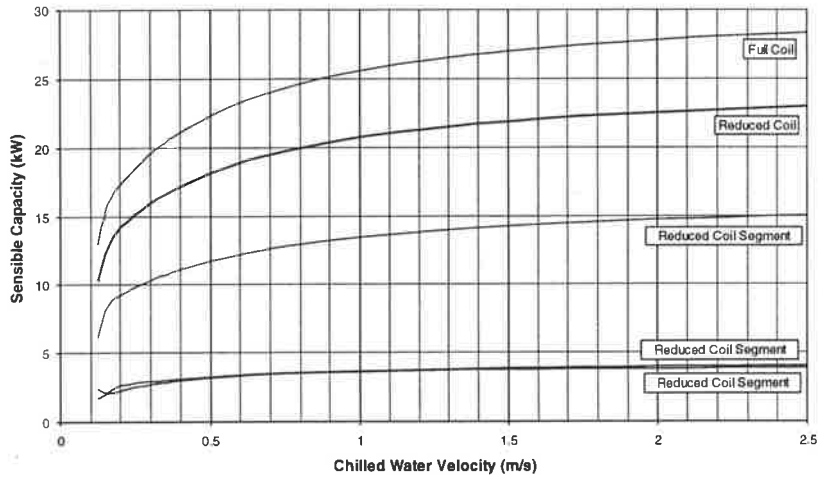


Figure A.5.c. Latent capacity vs. CW velocity. Air face velocity = 1·5 m/s.

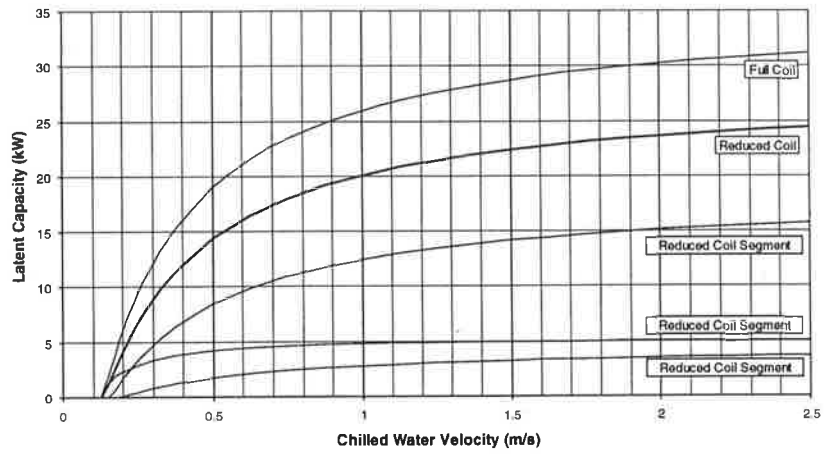


Figure A.5.d. Sensible heat ratio vs. CW velocity. Air face velocity = 1·5 m/s.

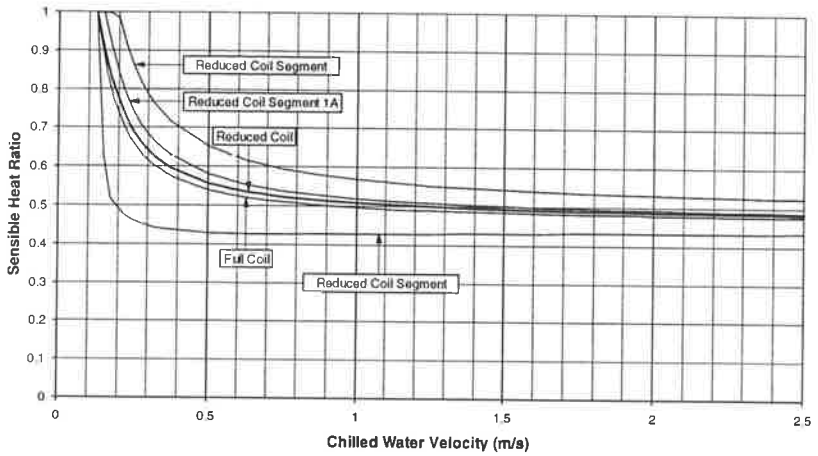


Figure A.5.e. Db temperature at various points in the coil vs. CW velocity. Air face velocity = 1·5 m/s.

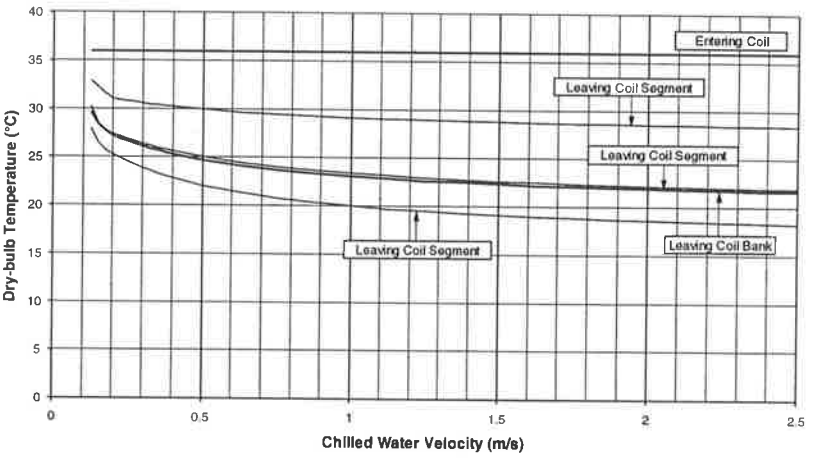


Figure A.5.f. Humidity ratio at various points in the coil vs. CW velocity. Air face velocity = 1·5 m/s.

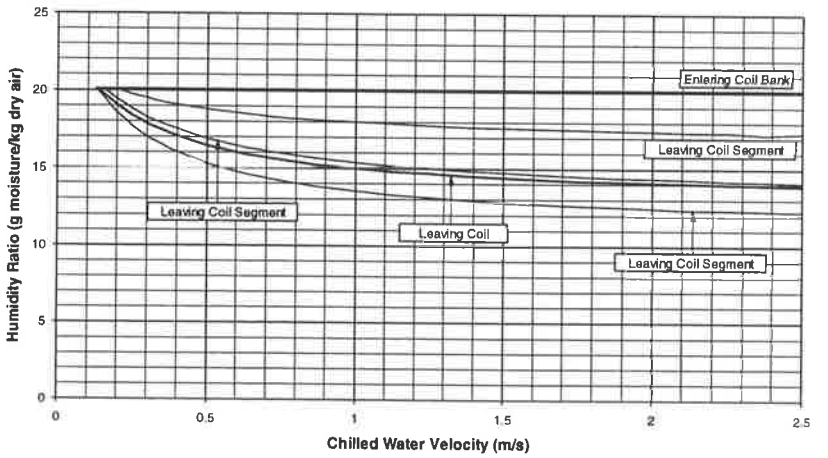


Figure A.5.g. Water pressure drop vs. CW velocity. Air face velocity = 1·5 m/s.

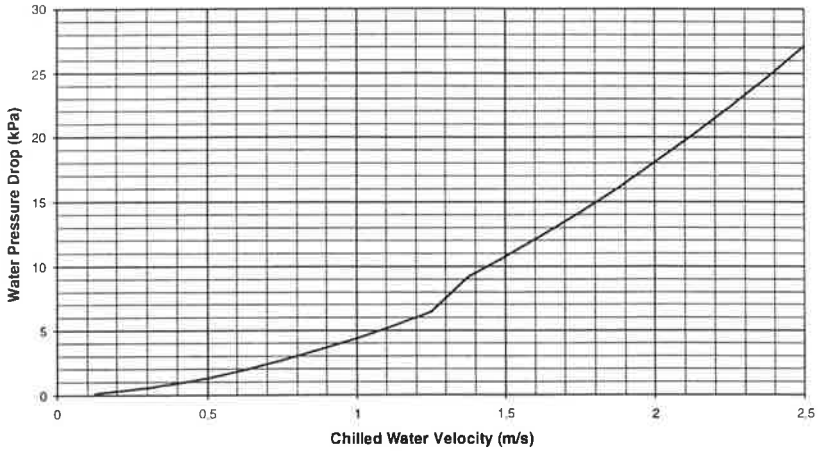
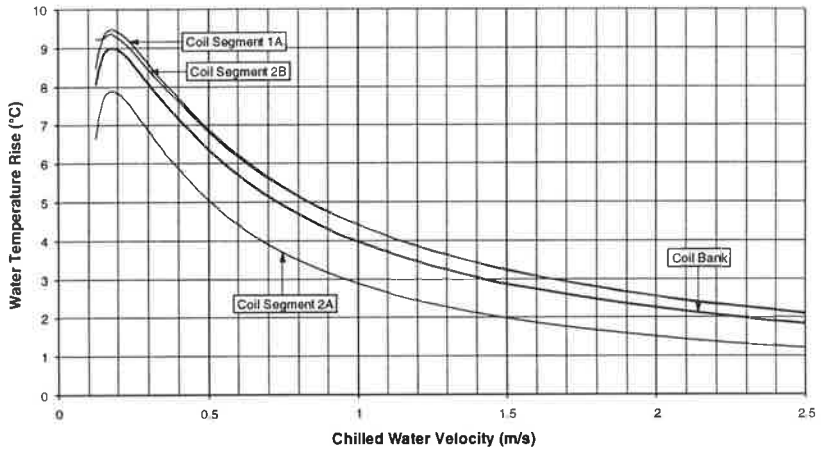


Figure A.5.h. Water temperature vs. CW velocity. Air face velocity = 1·5 m/s.



Appendix B. Load Data for the Multistorey Building.

The following graphs describe the ambient conditions and zone loads for simulating the performance of a series of air conditioning plant configurations in a multistorey office building. Full details of the office building and the test matrix will be found in section 13.3. The data presented in the following graphs models the ambient conditions and zone load situations for:

a. January, Adelaide. 07:00 to 20:00.

b. April, Adelaide. 07:00 to 20:00.

c. March, Kuala Lumpur. 07:00 to 20:00.

d. November, Kuala Lumpur. 07:00 to 20:00.

Figure B.1. Ambient dry and wet bulb temperatures for January. Adelaide.

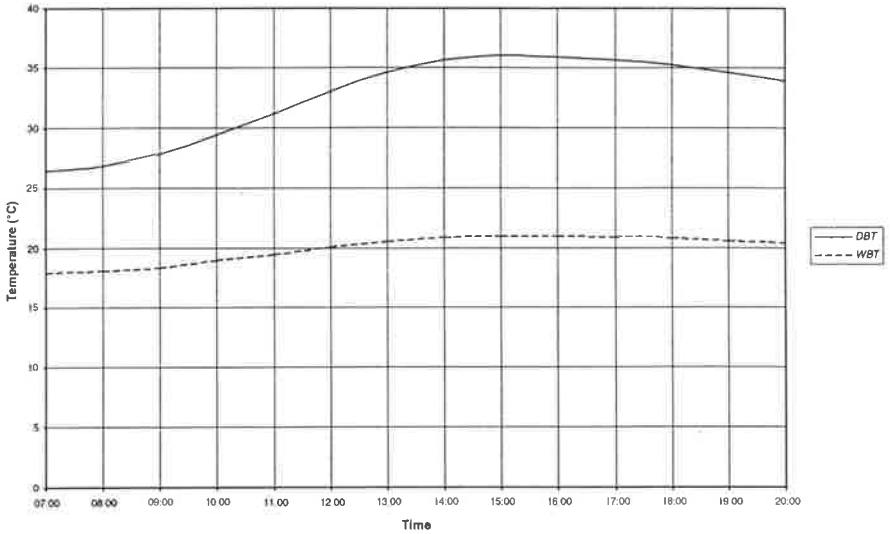


Figure B.2. Zone sensible loads for January. Adelaide.

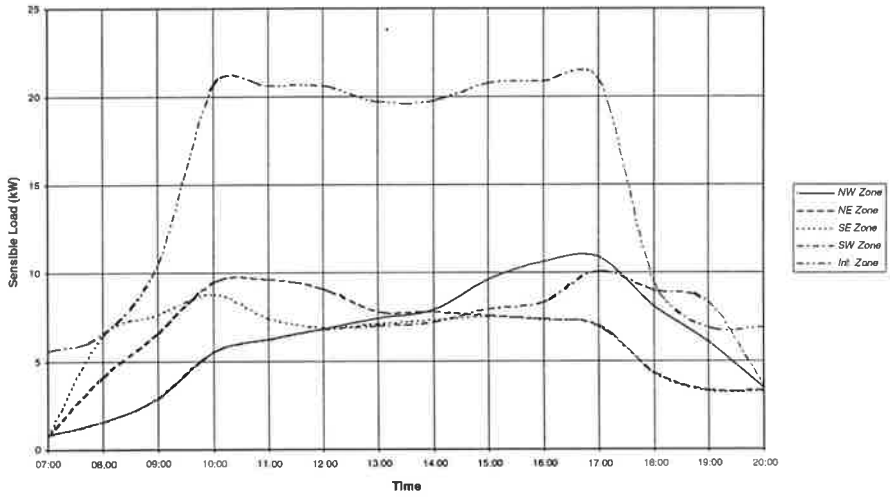


Figure B.3. Zone sensible heat ratios for January. Adelaide.

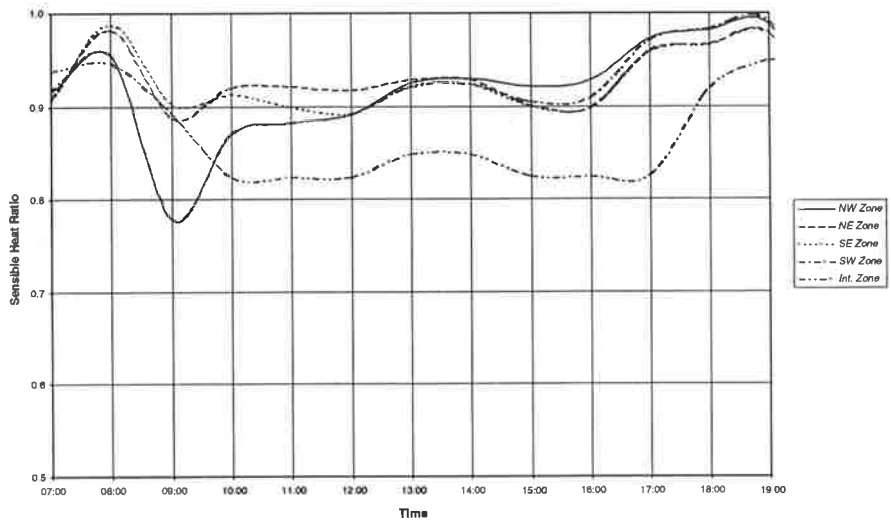


Figure B.4. Ambient dry and wet bulb temperature for April. Adelaide.

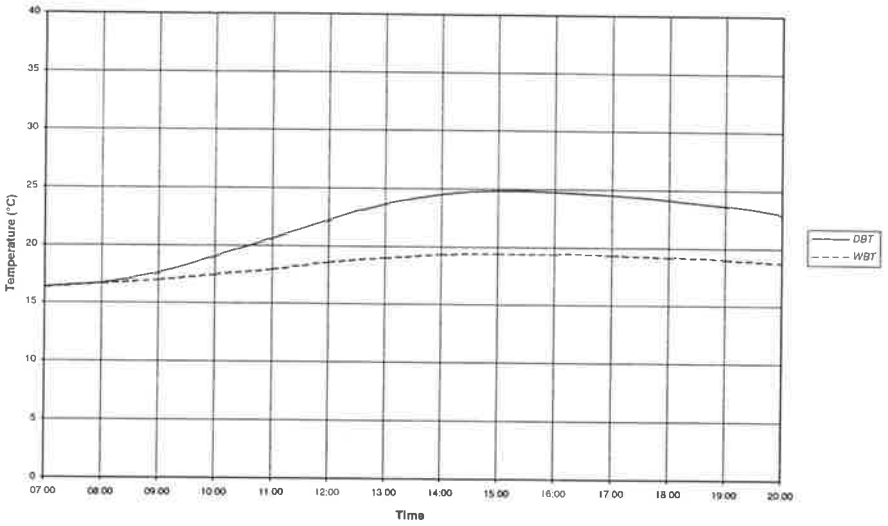


Figure B.5. Zone sensible loads for April. Adelaide.

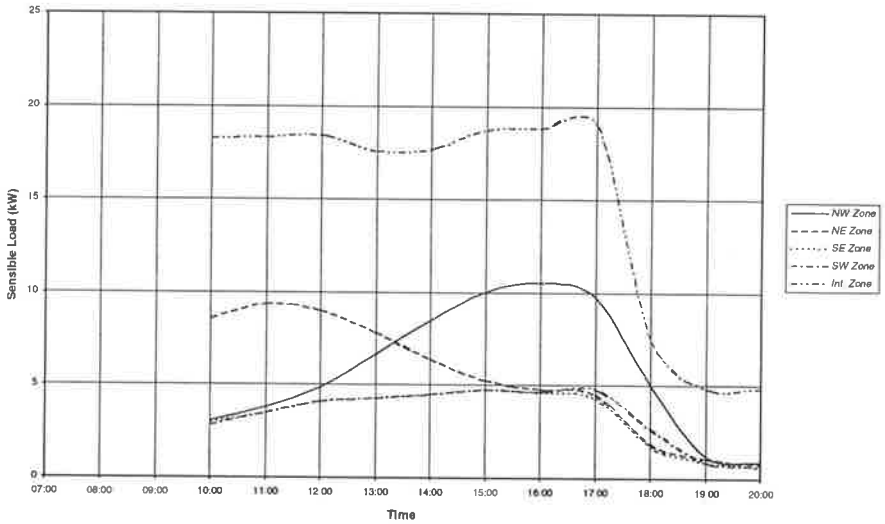


Figure B.6. Zone sensible heat ratios for April. Adelaide.

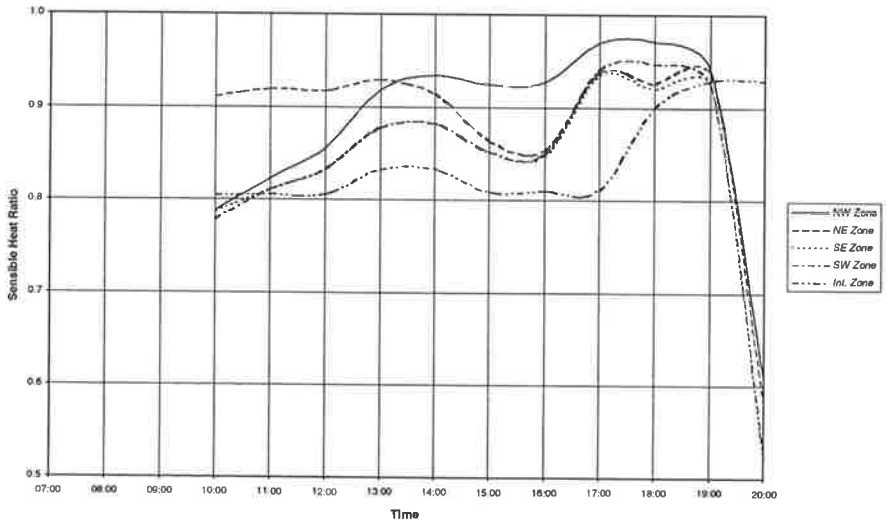


Figure B.7. Ambient dry and wet bulb temperature for March. Kuala Lumpur.

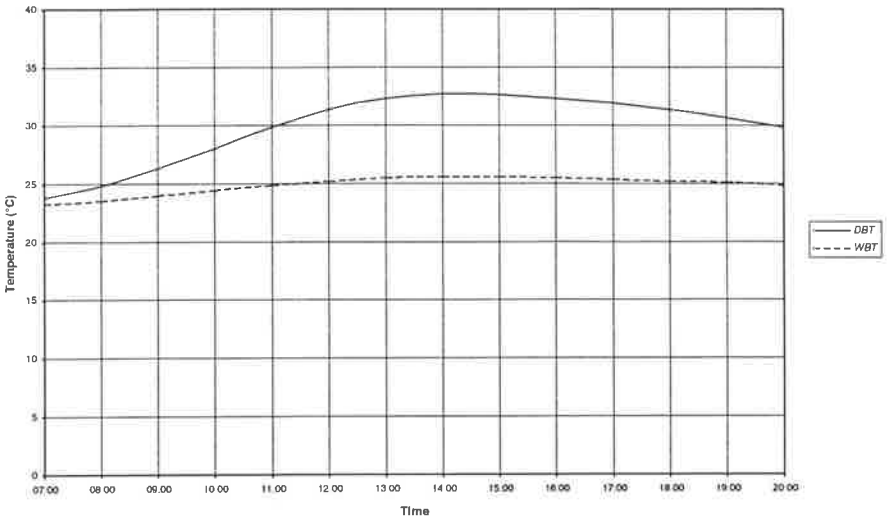


Figure B.8. Zone sensible loads for March. Kuala Lumpur.

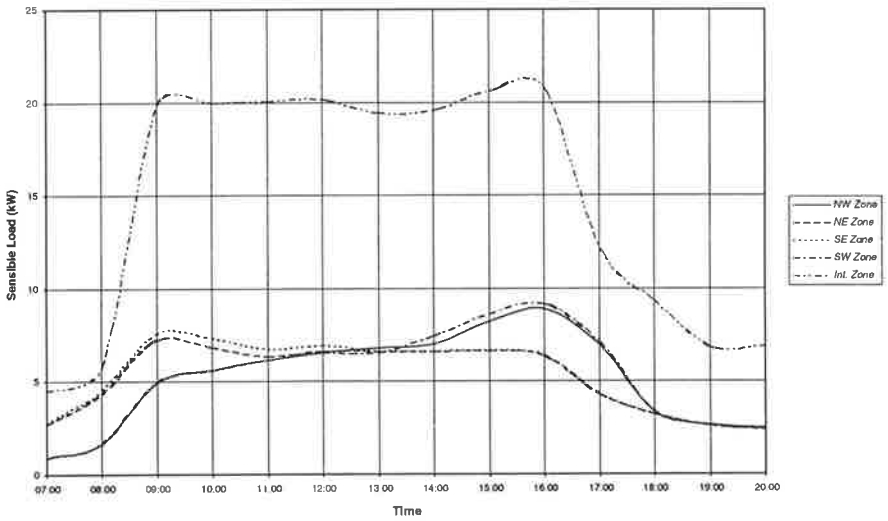


Figure B.9. Zone sensible heat ratios for March. Kuala Lumpur.

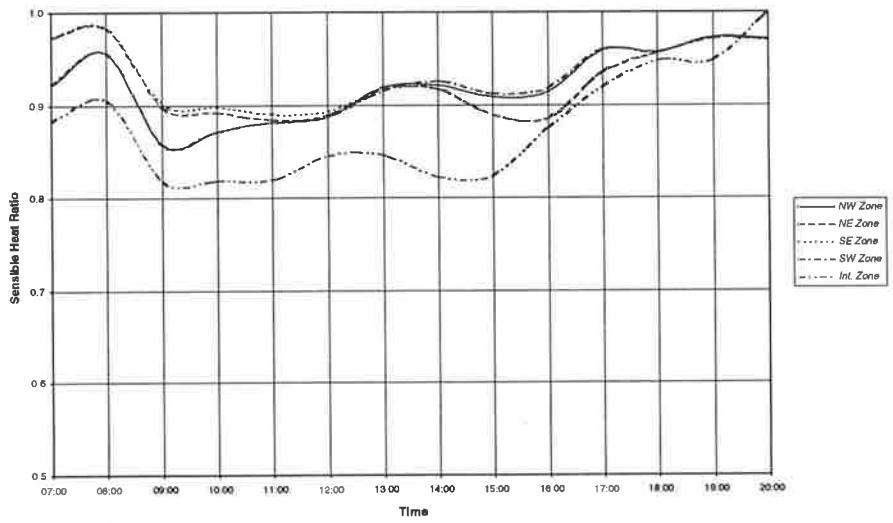


Figure B.10. Ambient dry and wet bulb temperature for November. Kuala Lumpur.

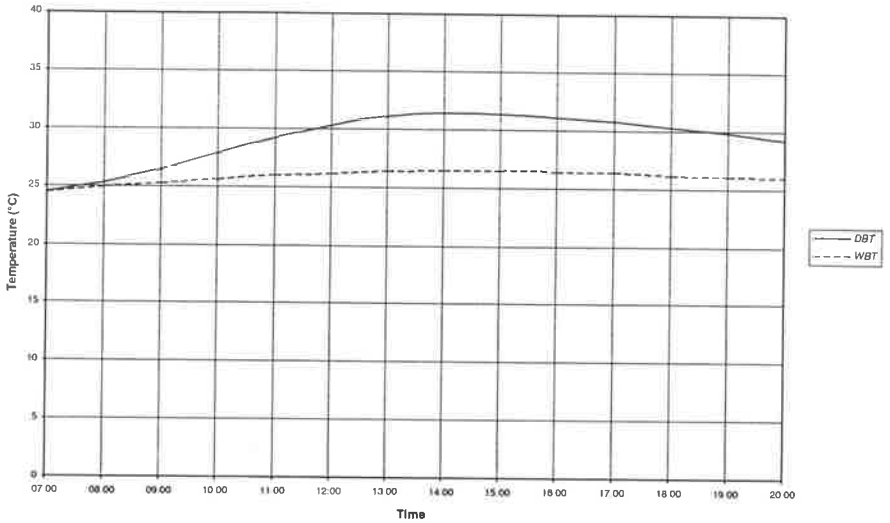


Figure B.11. Zone sensible loads for November. Kuala Lumpur.

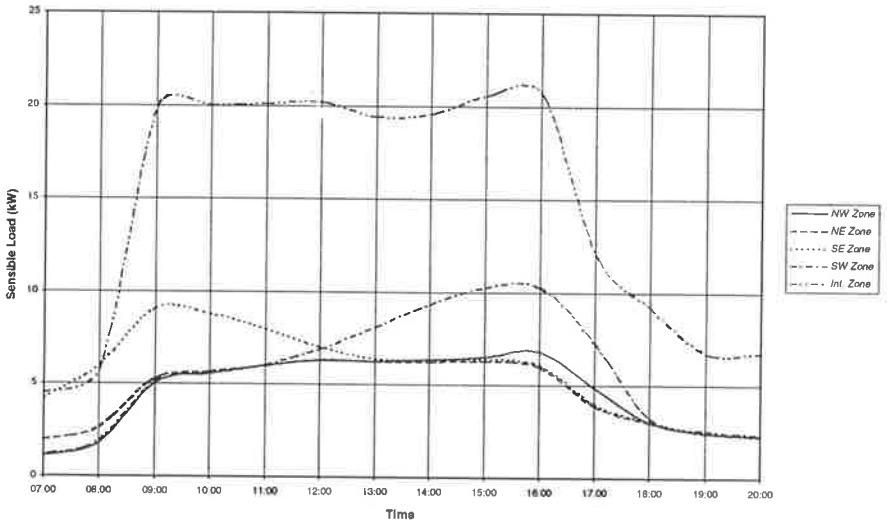
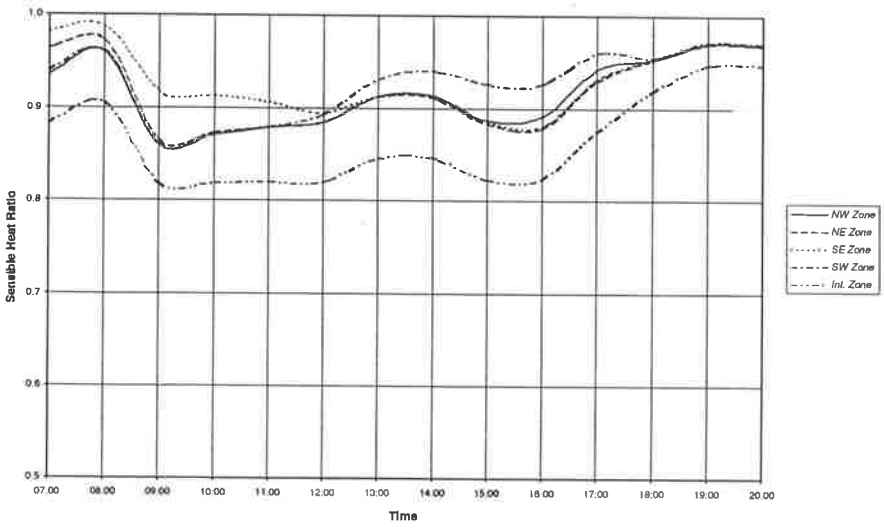


Figure B.12. Zone sensible heat ratios for November. Kuala Lumpur.



Appendix C. Zone Conditions for the Multistorey Building.

The following graphs plot the zone relative humidity attained in a multistorey office building subject to a test matrix defined by the following parameters:

a. Climatic regime, the following cases being considered:

   i. Adelaide for the months of January and April;
   ii. Kuala Lumpur for the months of March and November.

b. Ventilation air flow rates of:

   i. 2·5 L/s per person;
   ii. 10 L/s per person.

c. Air conditioning system configurations:

   i. Conventional multizone VAV;
   ii. Conventional multizone CAV;
   iii. HDP.

In all cases, the zone dry-bulb temperature has been maintained at 24°C. Full details of the building, the test matrix and the modelling procedure will be found in section 13.3.

Figure C.1. Zone conditions obtained using a conventional multizone VAV system. Adelaide, January. Ventilation air flow 2·5 L/s per person.

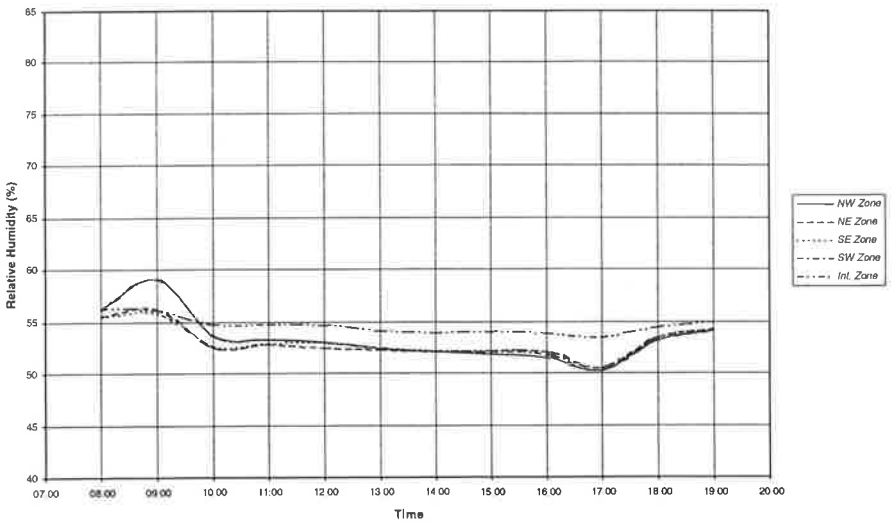


Figure C.2. Zone conditions obtained using a conventional multizone CAV system. Adelaide, January. Ventilation air flow 2·5 L/s per person.

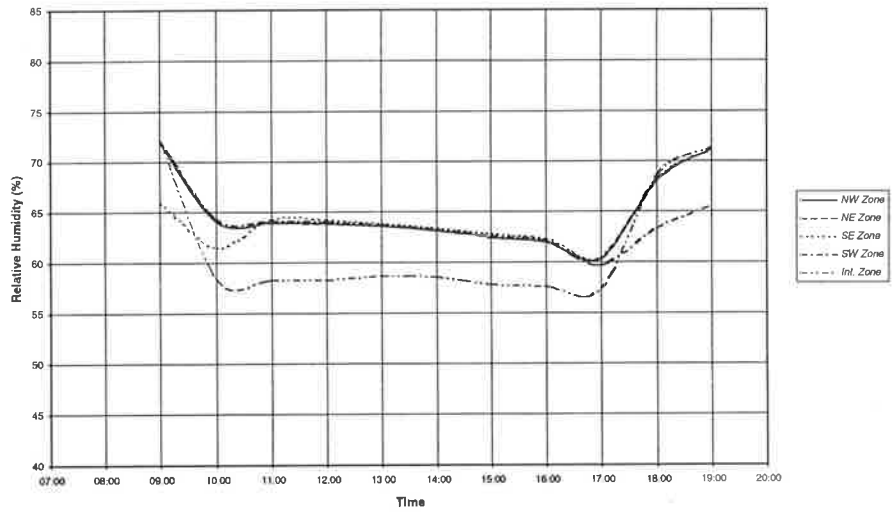


Figure C.3. Zone conditions obtained using an HDP system *without staging*. Adelaide, January. Ventilation air flow 2·5 L/s per person.

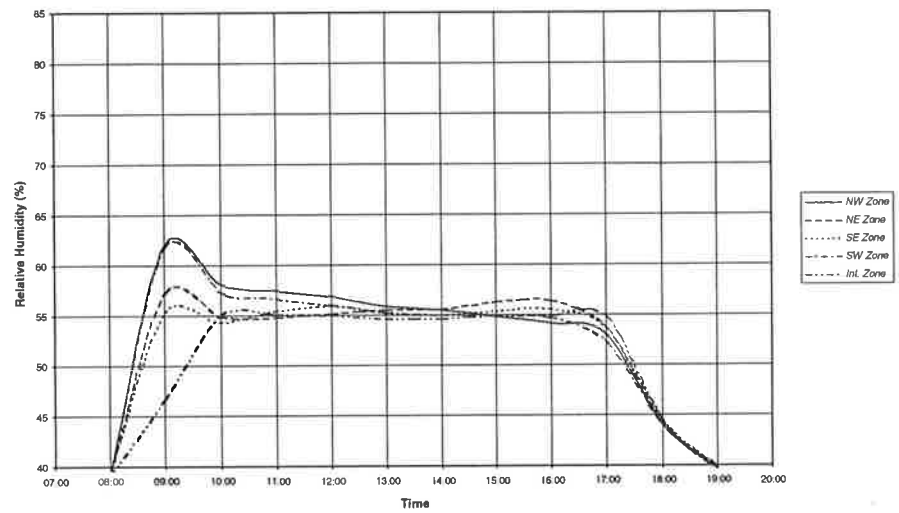


Figure C.4. Zone conditions obtained using a conventional multizone VAV system. Adelaide, January. Ventilation air flow 10 L/s per person.

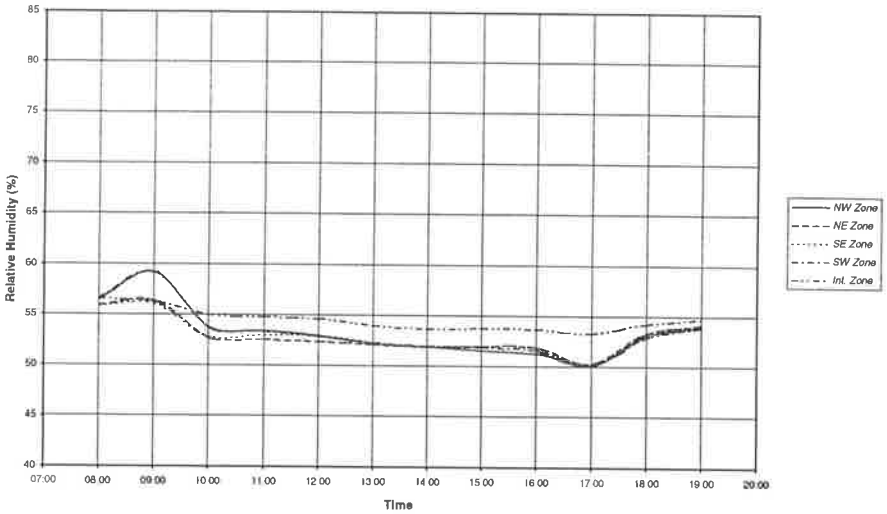


Figure C.5. Zone conditions obtained using a conventional multizone CAV system. Adelaide, January. Ventilation air flow 10 L/s per person.

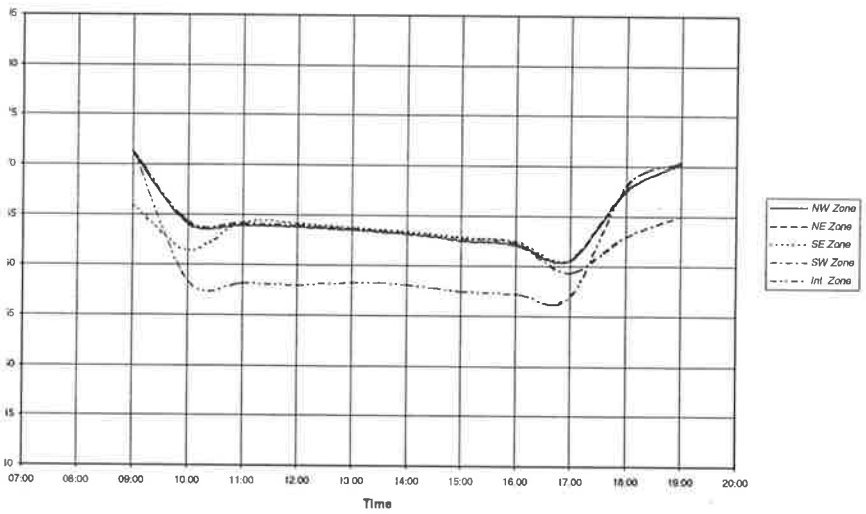


Figure C.6. Zone conditions obtained using an HDP system without staging. Adelaide, January. Ventilation air flow 10 L/s per person.

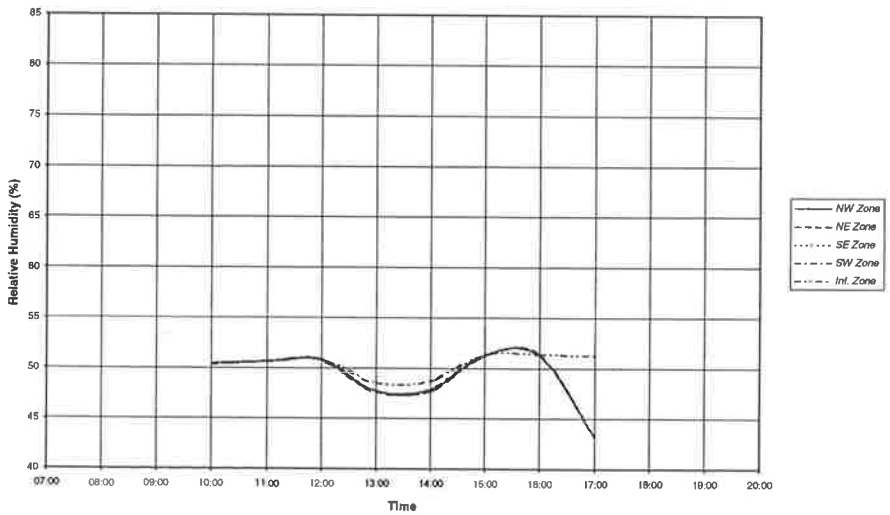


Figure C.7. Zone conditions obtained using a conventional multizone VAV system. Adelaide, April. Ventilation air flow 2·5 L/s per person.

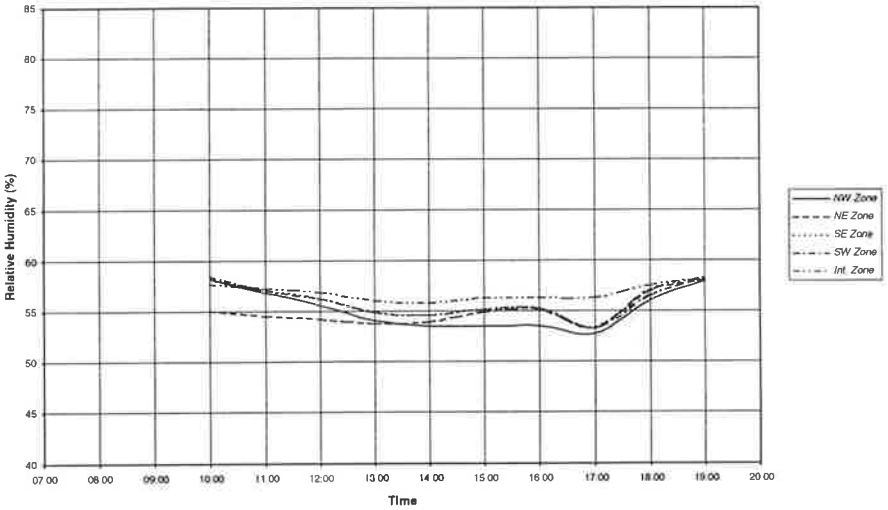


Figure C.8. Zone conditions obtained using a conventional multizone CAV system. Adelaide, April. Ventilation air flow 2·5 L/s per person.

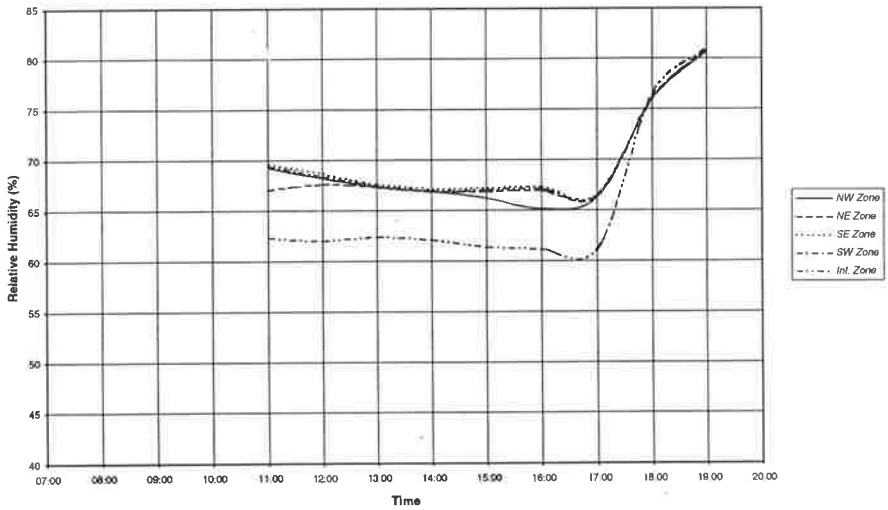


Figure C.9. Zone conditions obtained using an HDP system *without staging*. Adelaide, April. Ventilation air flow 2·5 L/s per person.

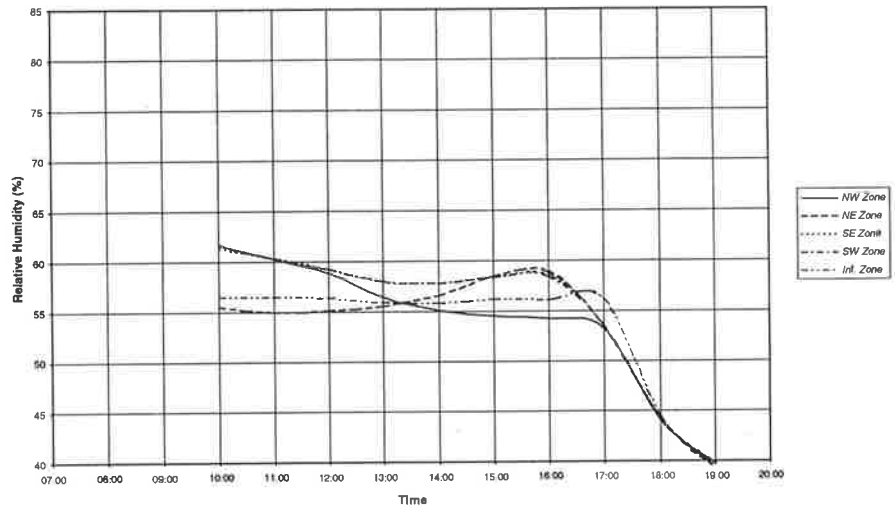


Figure C.10. Zone conditions obtained using a conventional multizone VAV system. Adelaide, April. Ventilation air flow 10 L/s per person.

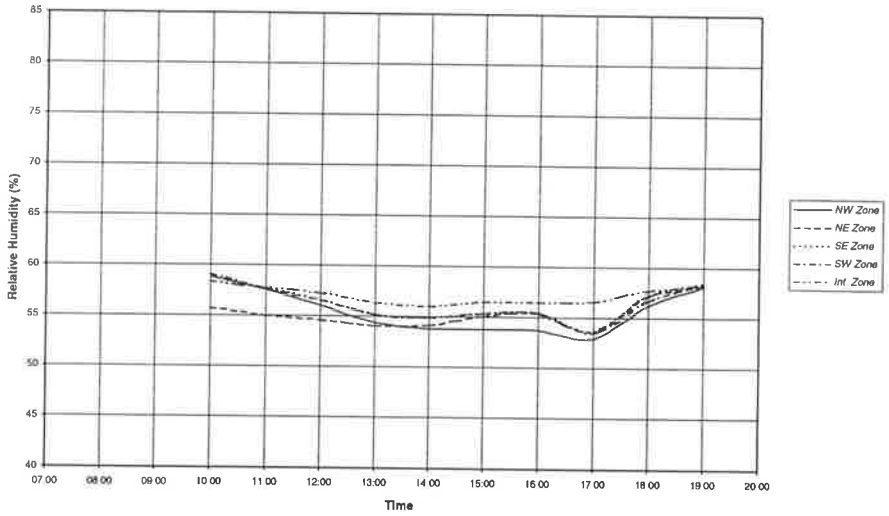


Figure C.11. Zone conditions obtained using a conventional multizone CAV system. Adelaide, April. Ventilation air flow 10 L/s per person.

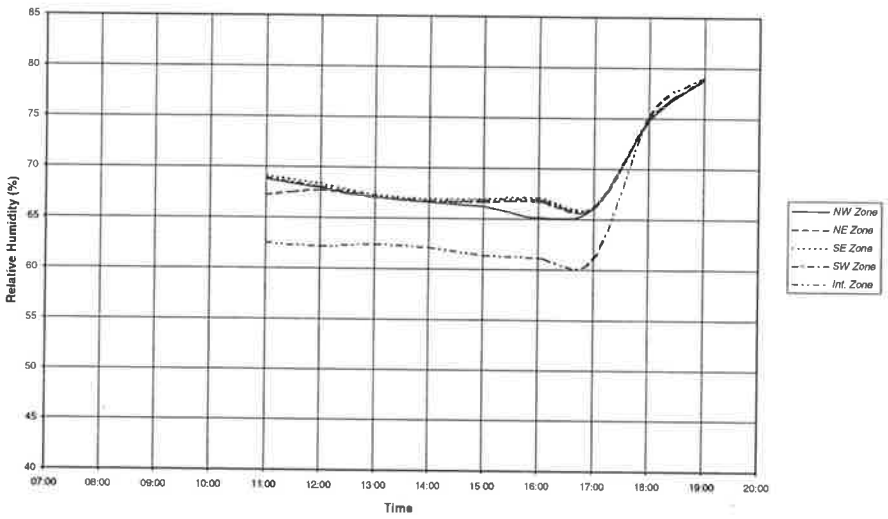


Figure C.12. Zone conditions obtained using an HDP system *without staging*. Adelaide, April. Ventilation air flow 10 L/s per person.

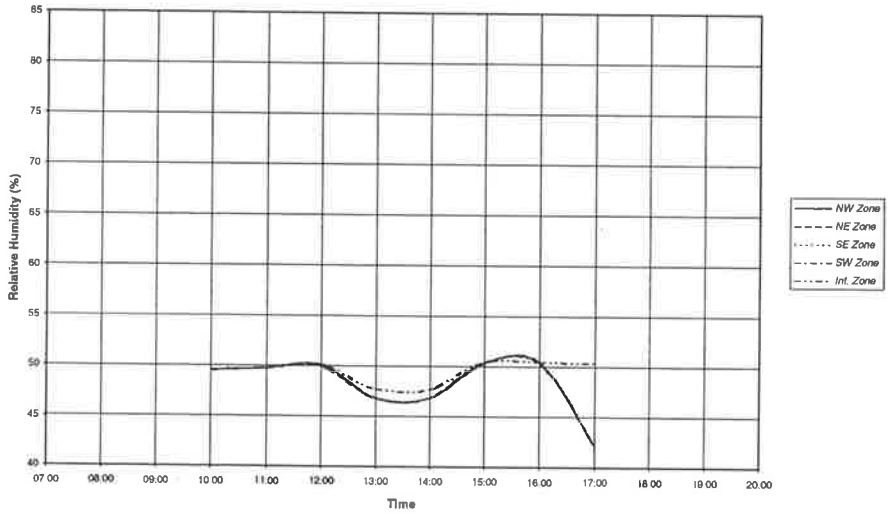


Figure C.13. Zone conditions obtained using a conventional multizone VAV system. Kuala Lumpur, March. Ventilation air flow 2·5 L/s/person.

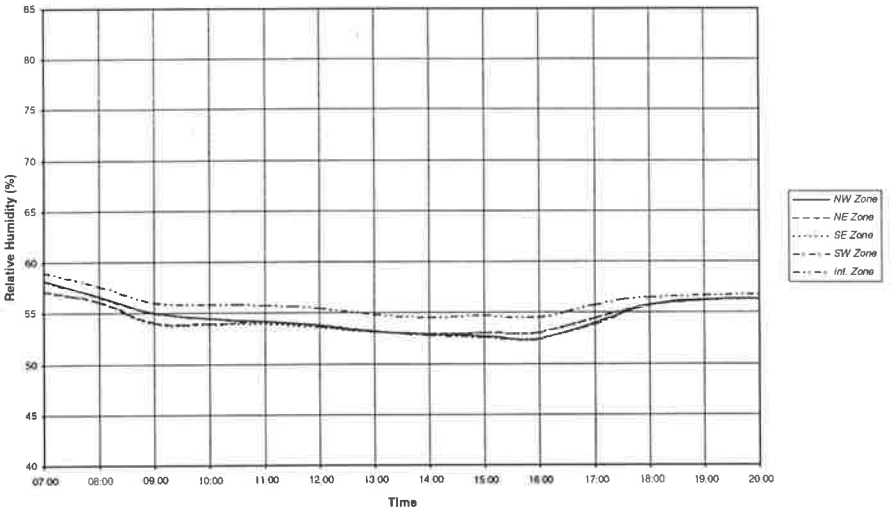


Figure C.14. Zone conditions obtained using a conventional multizone CAV system. Kuala Lumpur, March. Ventilation air flow 2·5 L/s/person.

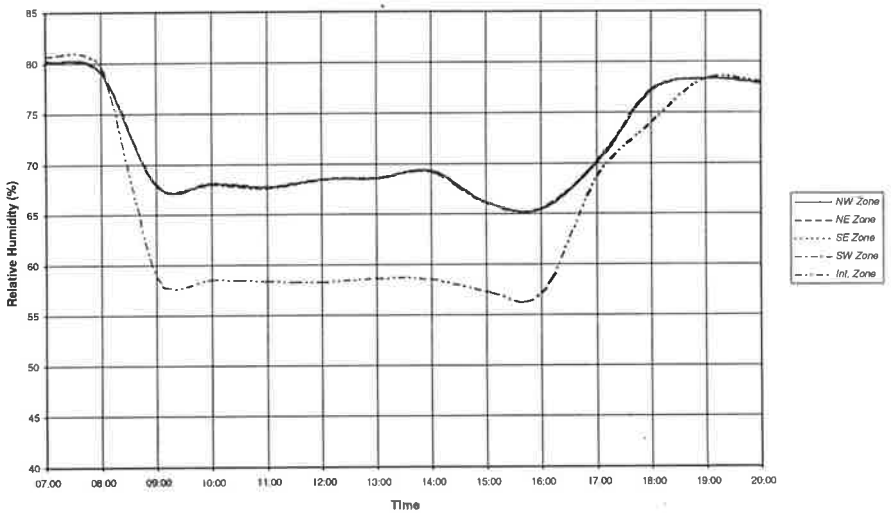


Figure C.15. Zone conditions obtained using an HDP system *without staging*. Kuala Lumpur, March. Ventilation air flow 2·5 L/s/person.

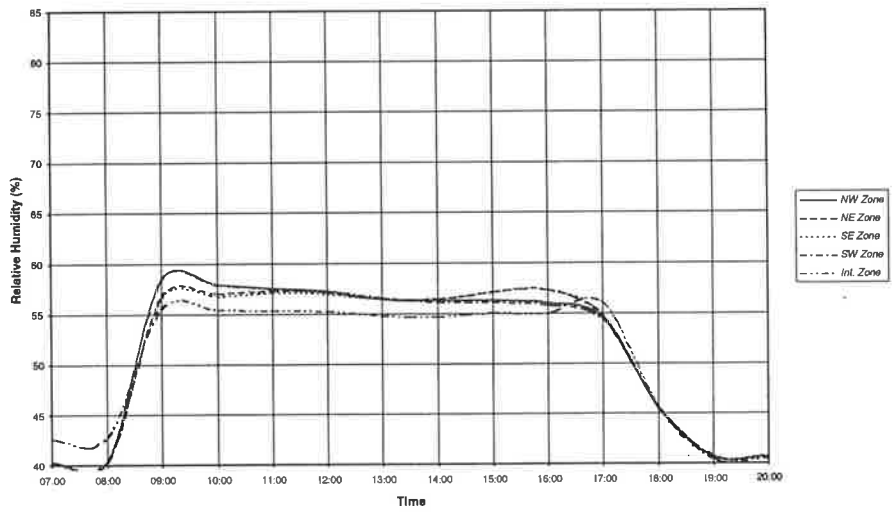


Figure C.16. Zone conditions obtained using a conventional multizone VAV system. Kuala Lumpur, March. Ventilation air flow 10 L/s/person.

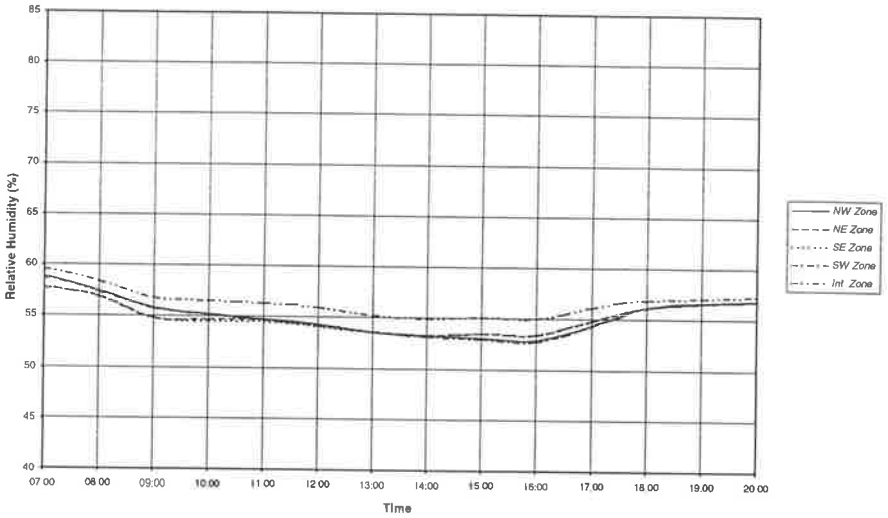


Figure C.17. Zone conditions obtained using a conventional multizone CAV system. Kuala Lumpur, March. Ventilation air flow 10 L/s/person.

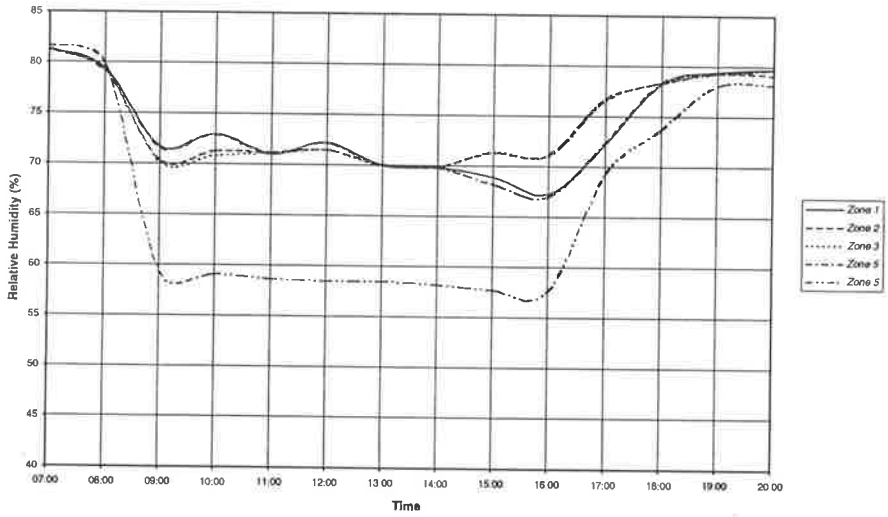


Figure C.18. Zone conditions obtained using an HDP system *without staging*. Kuala Lumpur, March. Ventilation air flow 10 L/s/person.

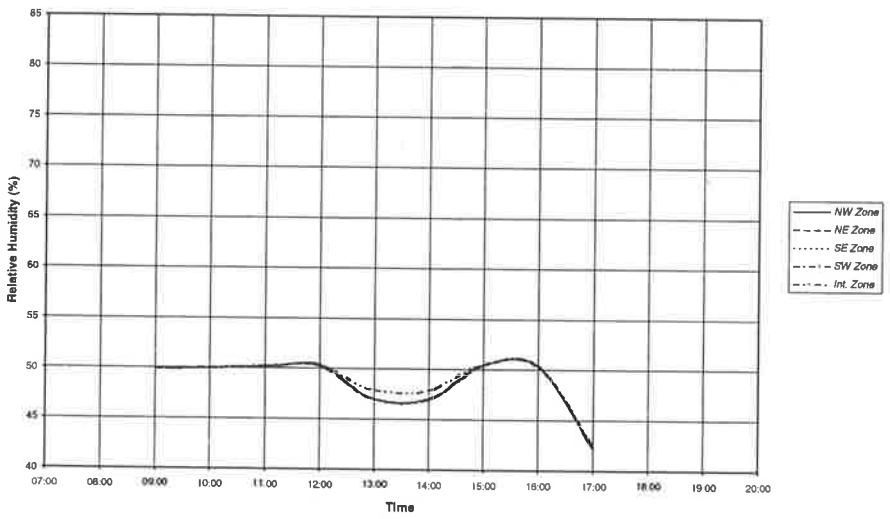


Figure C.19. Zone conditions obtained using a conventional multizone VAV system. Kuala Lumpur, Nov. Ventilation air flow 2·5 L/s/person.

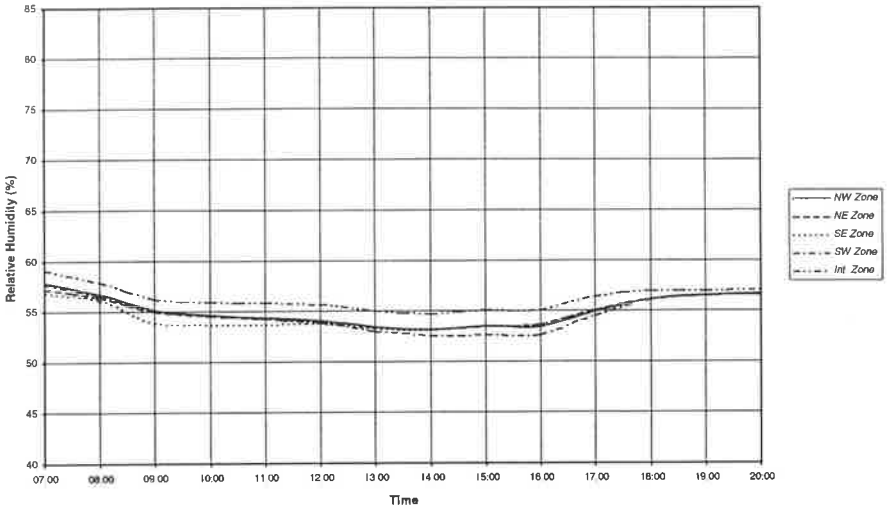


Figure C.20. Zone conditions obtained using a conventional multizone CAV system. Kuala Lumpur, Nov. Ventilation air flow 2·5 L/s/person.

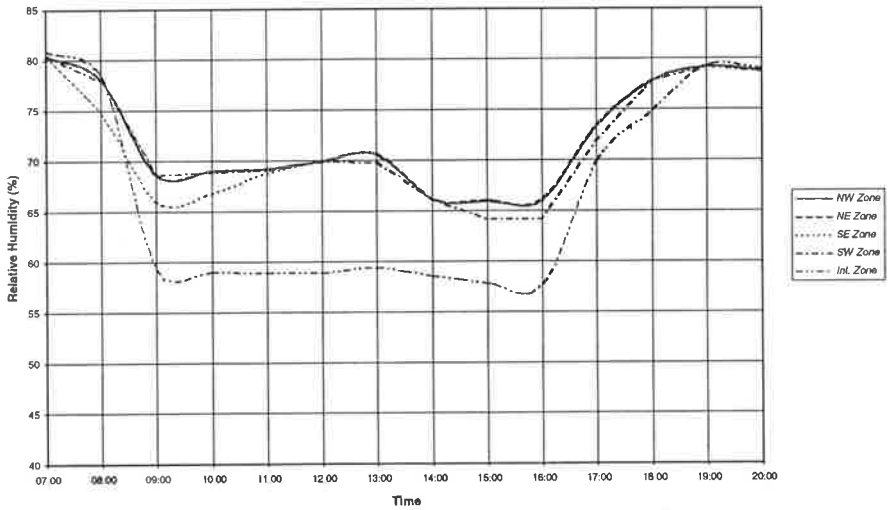


Figure C.21. Zone conditions obtained using an HDP system *without staging*. Kuala Lumpur, Nov. Ventilation air flow 2·5 L/s/person.

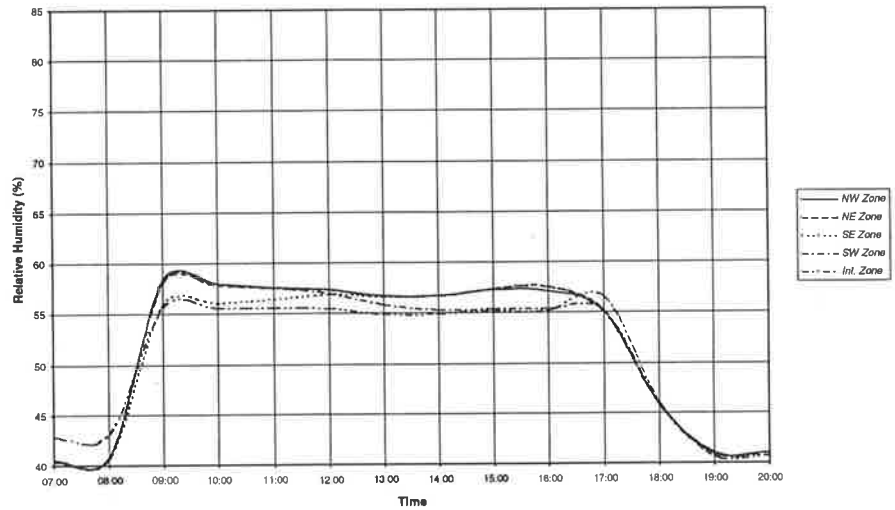


Figure C.22. Zone conditions obtained using a conventional multizone VAV system. Kuala Lumpur, Nov. Ventilation air flow 10 L/s/person.

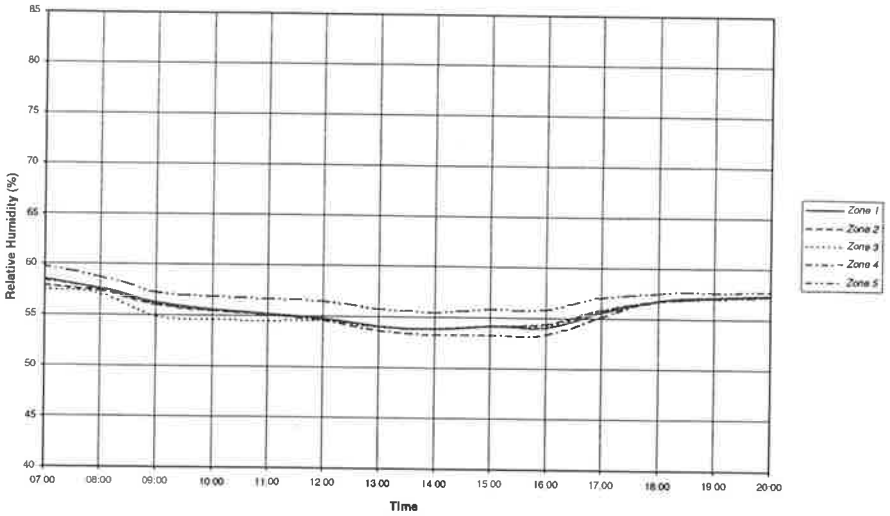


Figure C.23. Zone conditions obtained using a conventional multizoneCAV system. Kuala Lumpur, Nov. Ventilation air flow 10 L/s/person.

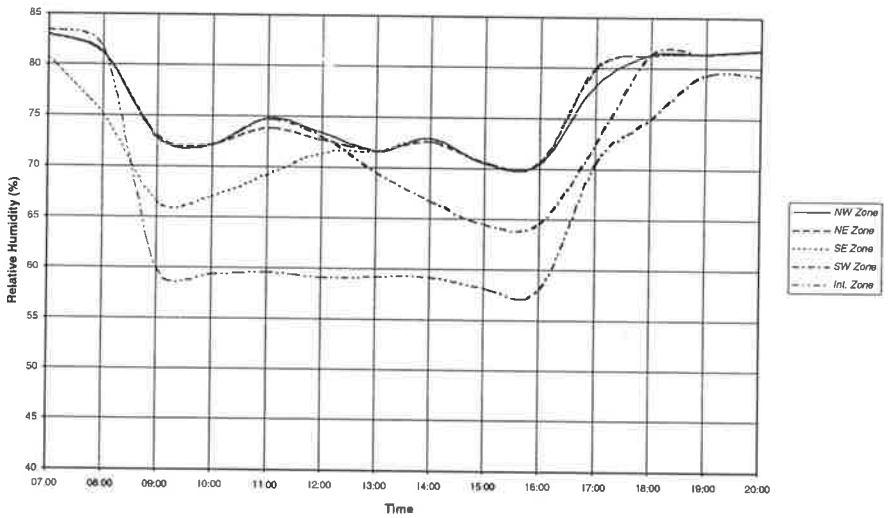


Figure C.24. Zone conditions obtained using an HDP system *without staging*. Kuala Lumpur, Nov. Ventilation air flow 10 L/s/person.

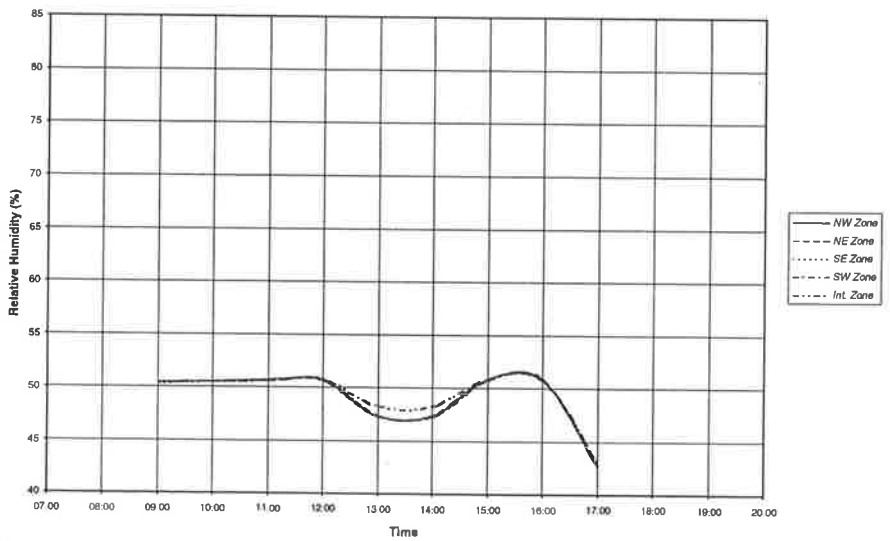


Figure C.25. Zone conditions obtained using an HDP system *with staging on perimeter zones*. Kuala Lumpur, March. Ventilation air flow 2·5 L/s per person.

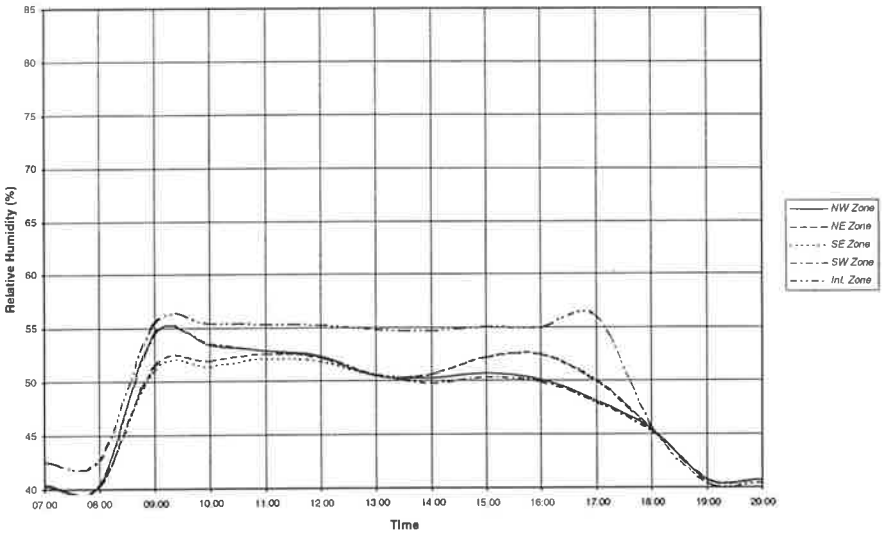
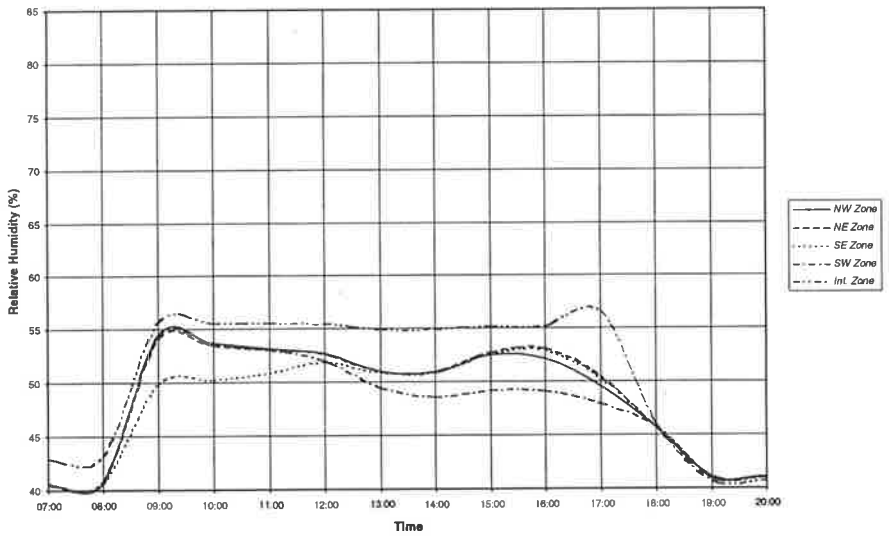


Figure C.26. Zone conditions obtained using an HDP system *with staging on perimeter zones*. Kuala Lumpur, November. Ventilation air flow 2·5 L/s per person.



Appendix D. Power Consumption for the Multistorey Building.

The following graphs plot the power consumption of various system configurations used to air condition a multistorey office building. The power consumption is itemized in terms of refrigeration, fan and pump power consumption, and has been calculated as power consumption for the entire building. In summary, the contents of the graphs are as follows:

1. Graphs *D.1* through to *D.8* compare the power consumption for a conventional multizone VAV system, and an HDP system, without staging. Neither system is subject to humidity control.

2. Graphs *D.9* through to *D.16* plot the power consumption for a conventional multizone CAV system. In simulations used to derive this series of graphs, overcooling and reheat has been applied as necessary to constrain the maximum zone humidity to a level of 60%. In other words, to just satisfy the provisions of ASHRAE Standard 55-1989. Reheat has been plotted as an additional item of power consumption in this series of graphs, and it has been found desirable to use an ordinate scale which differs from that used for the other graphs.

3. Graphs *D.17* and *D.18* compare the power consumption for an HDP system without staging, and one with staging applied to the perimeter zones. As with the first series of graphs, neither system is subject to humidity control.

Full details of the building, the test matrix and the modelling procedure will be found in section 13.3.

Figure D.1. Power Consumption for a conventional VAV system and an HDP VAV system for January. Adelaide. Ventilation air flow 2·5 L/s per person.

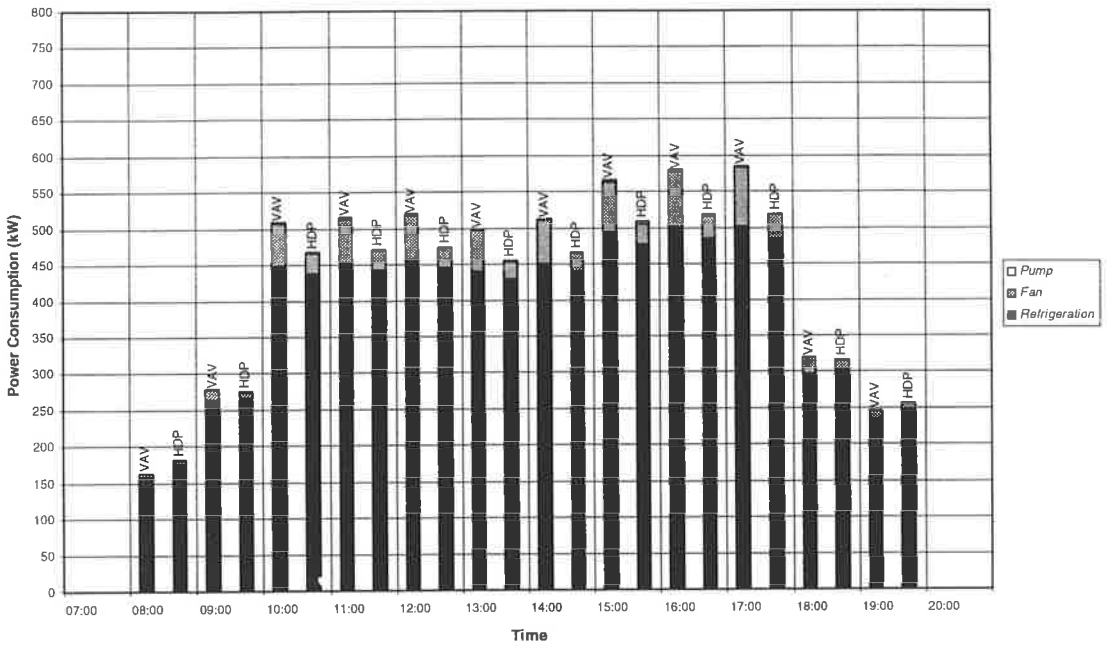


Figure D.2. Power consumption for a conventional VAV system and an HDP VAV system for January. Adelaide. Ventilation air flow 10 L/s per person.

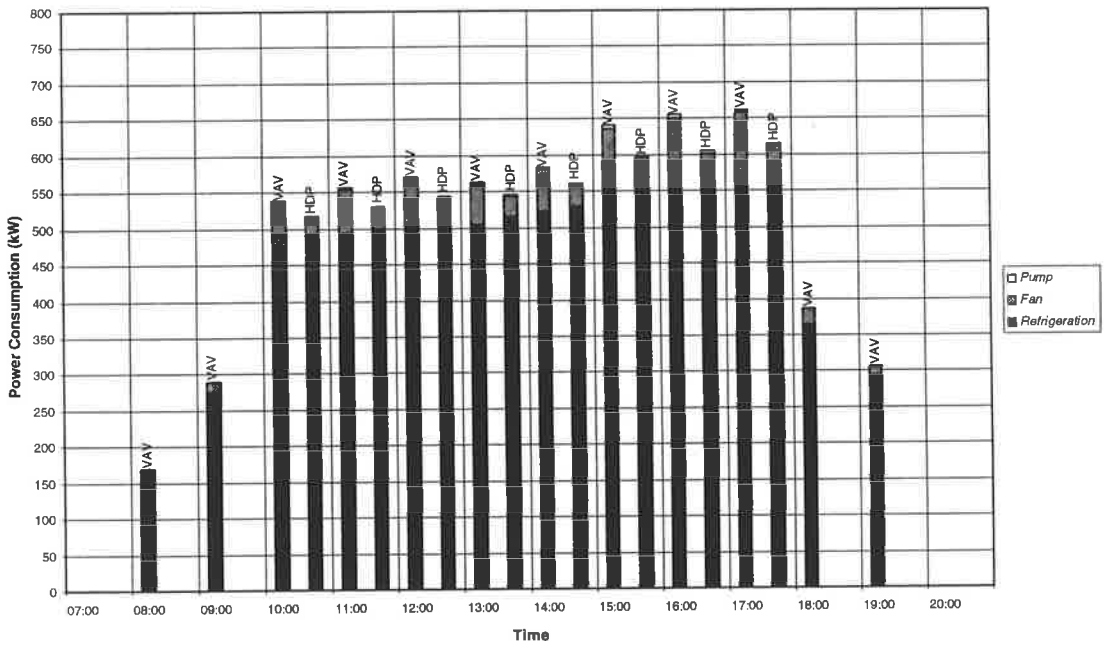


Figure D.3. Power consumption for a conventional VAV system and an HDP VAV system for April. Adelaide. Ventilation air flow 2·5 L/s per person.

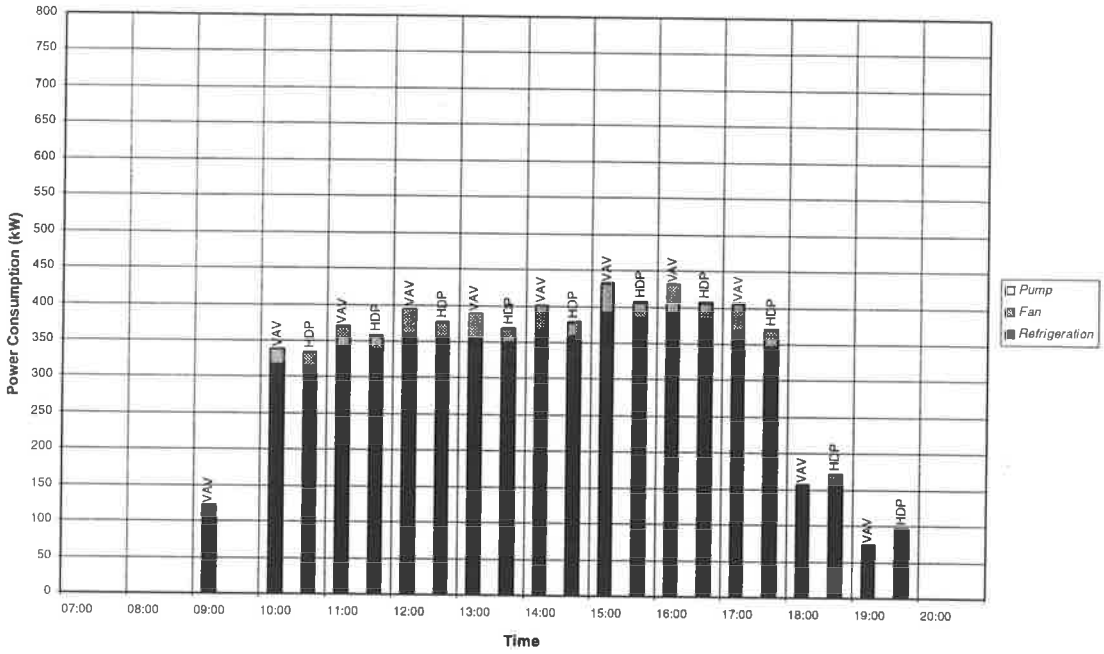


Figure D.4. Power consumption for a conventional VAV system and an HDP VAV system for April. Adelaide. Ventilation air flow 10 L/s per person.

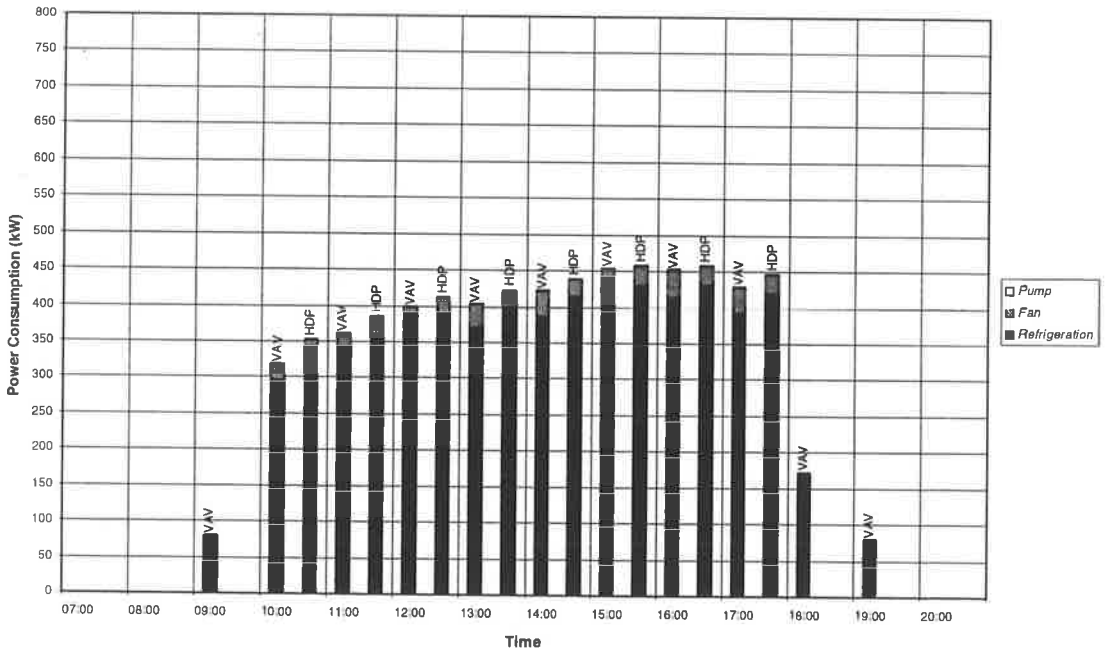


Figure D.5. Power consumption for a conventional VAV system and an HDP VAV system for March. Kuala Lumpur. Ventilation air flow 2·5 L/s per person.

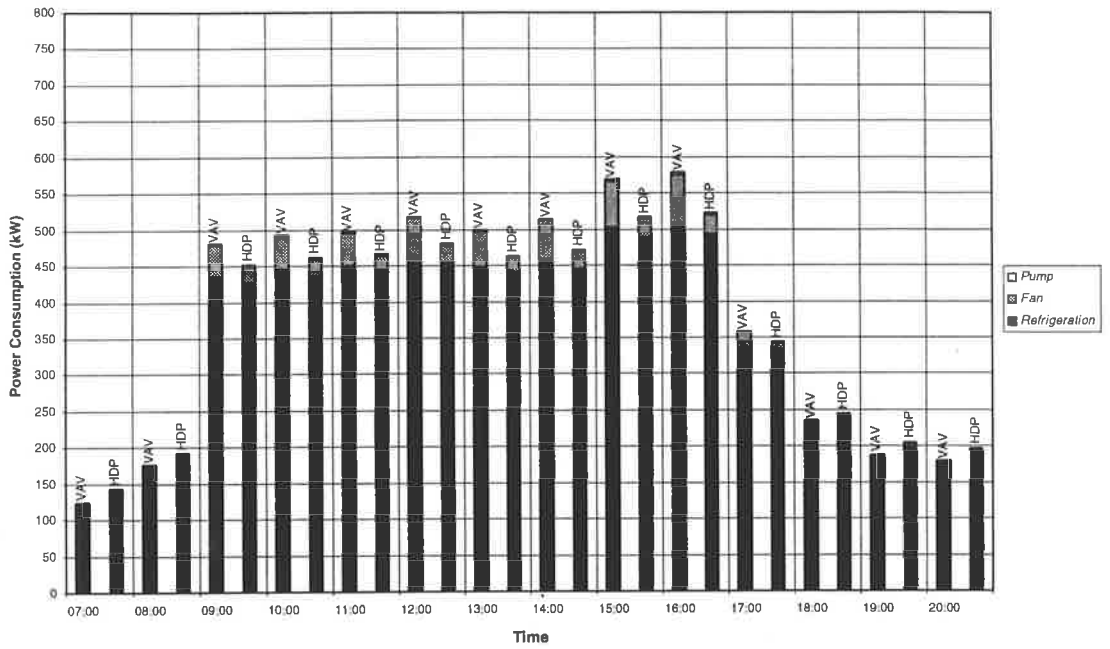


Figure D.6. Power consumption for a conventional VAV system and an HDP VAV system for March. Kuala Lumpur. Ventilation air flow 10 L/s per person.

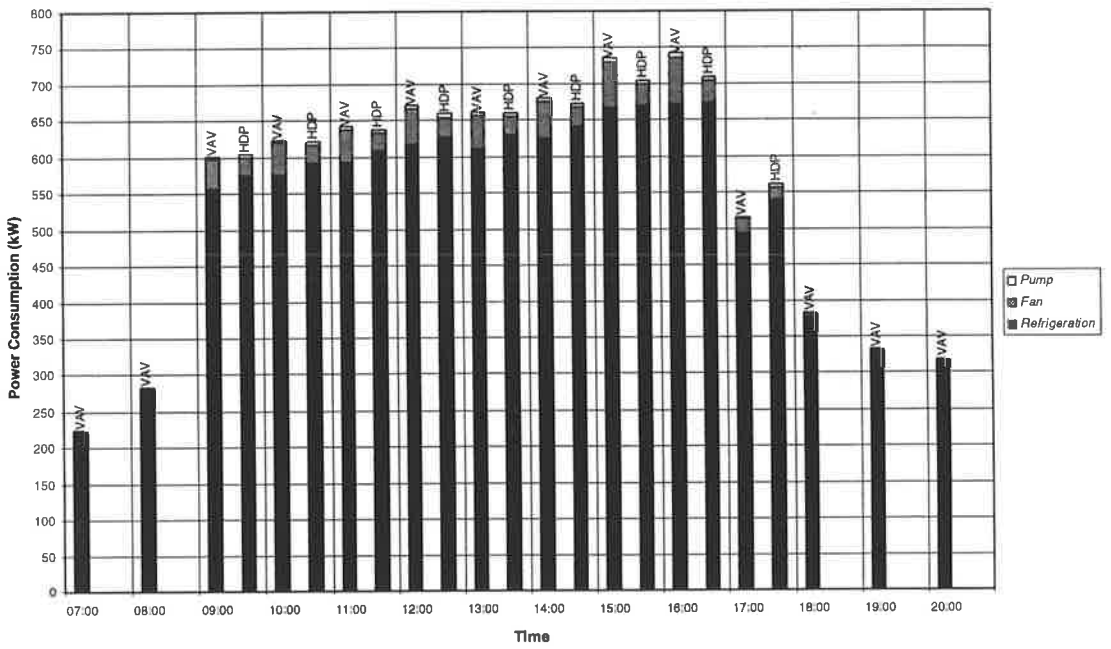


Figure D.7. Power consumption for a conventional VAV system and an HDP VAV system for November. Kuala Lumpur. Ventilation air flow 2·5 L/s per person.

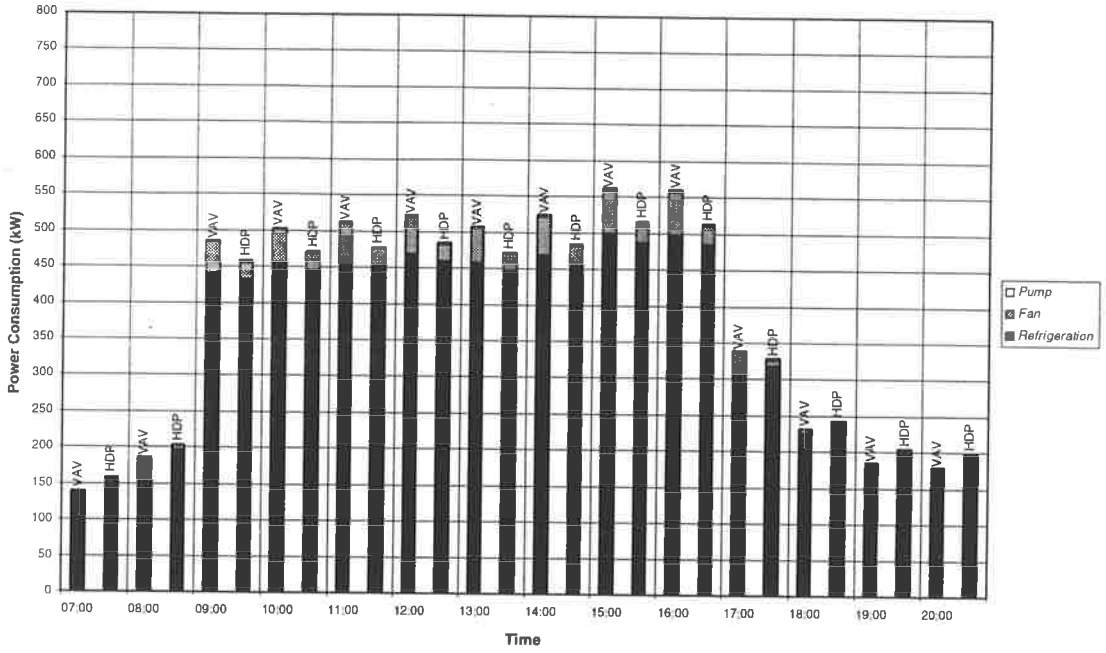


Figure D.8. Power consumption for a conventional VAV system and an HDP VAV system for November. Kuala Lumpur. Ventilation air flow 10 L/s per person.

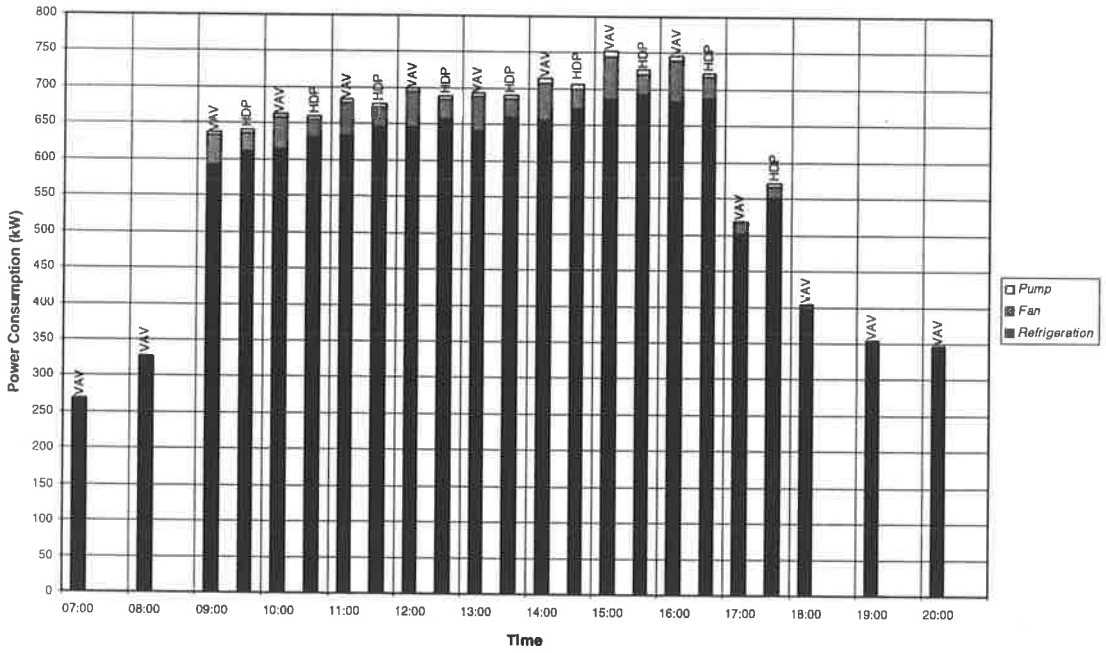


Figure D.9. Power consumption for a conventional CAV system for January. Adelaide. Ventilation air flow 2·5 L/s. Overcooling and reheat applied to constrain maximum zone humidity to 60%.

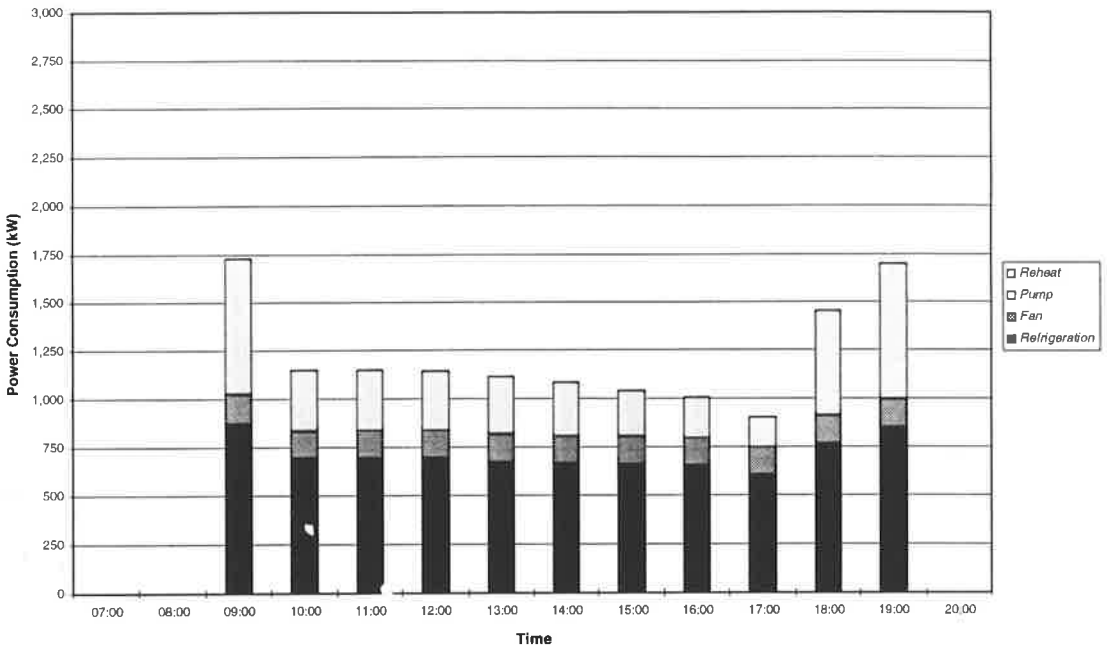


Figure D.10. Power consumption for a conventional CAV system for January. Adelaide. Ventilation air flow 10 L/s per person. Overcooling and reheat applied to constrain maximum zone humidity to 60%.

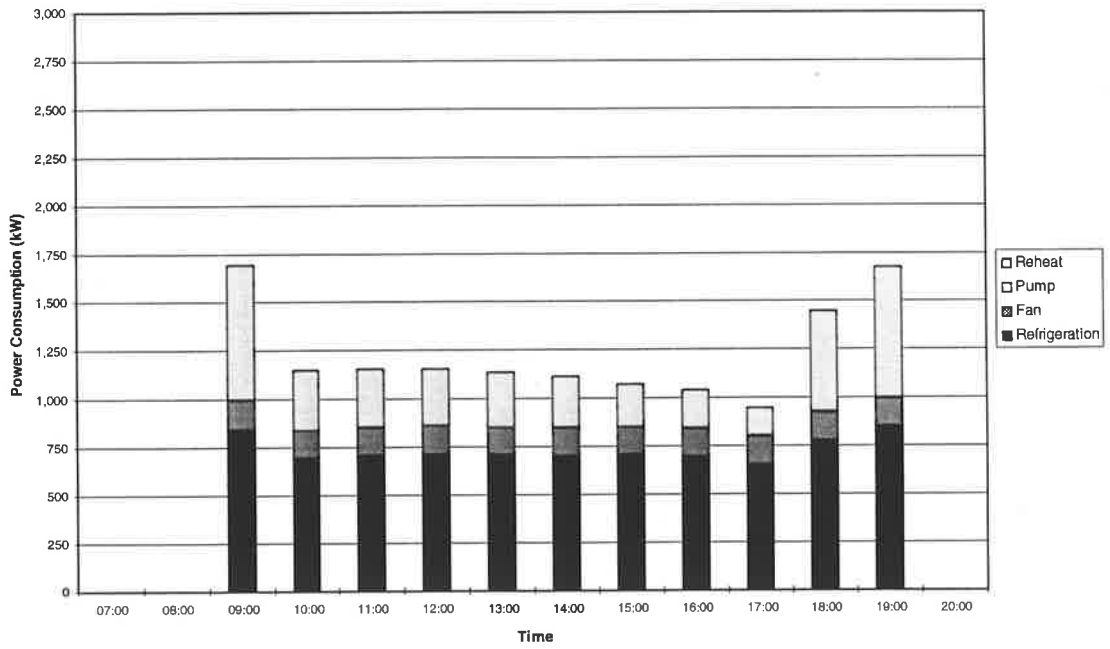


Figure D.11. Power consumption for a conventional CAV system for April. Adelaide. Ventilation air flow 2·5 L/s per person. Overcooling and reheat applied to constrain maximum zone humidity to 60%.

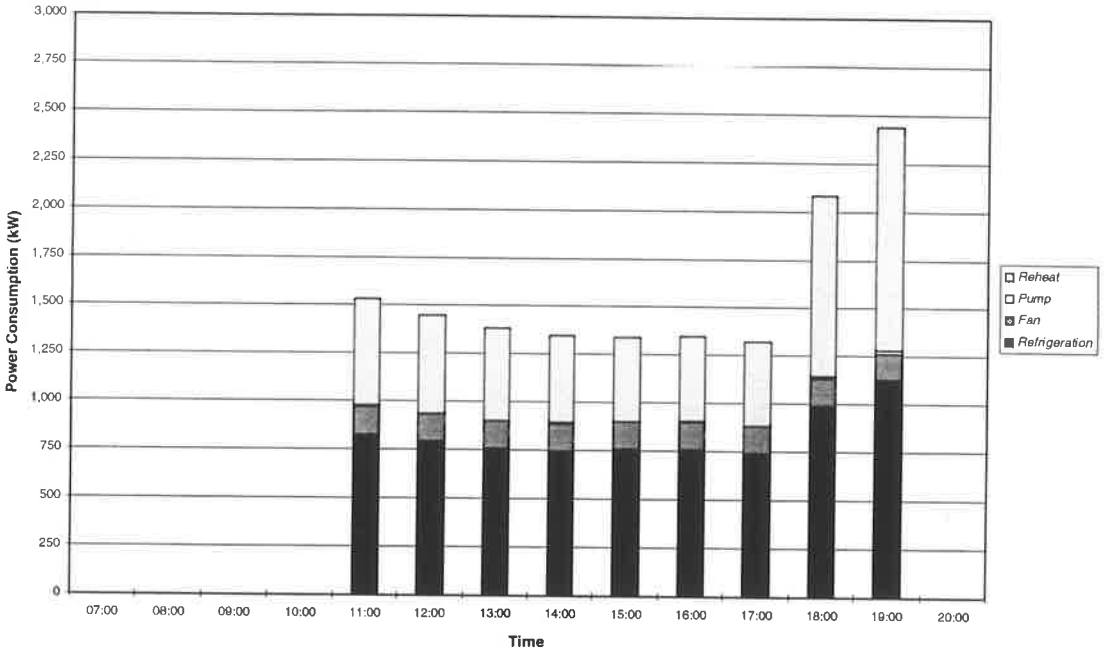


Figure D.12. Power consumption for a conventional CAV system for April. Adelaide. Ventilation air flow 10 L/s per person. Overcooling and reheat applied to constrain maximum zone hunidity to 60%.

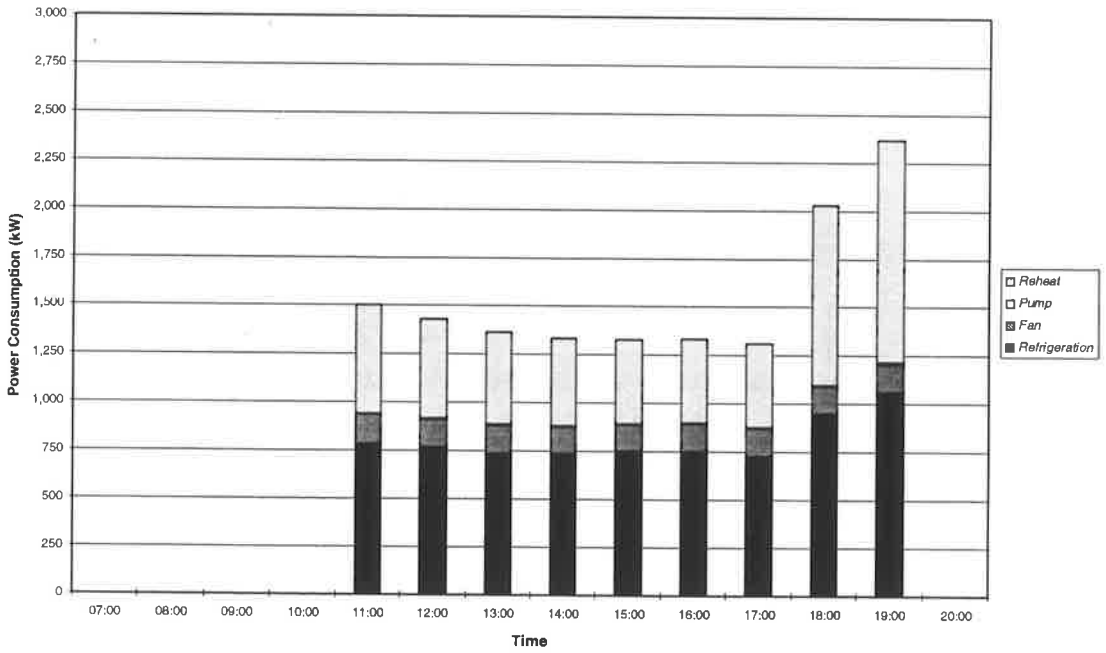


Figure D.13. Power consumption for a conventional CAV system for March. Kuala Lumpur. Ventilation air flow 2·5 L/s per person. Overcooling and reheat applied to constrain maximum zone humidity to 60%.

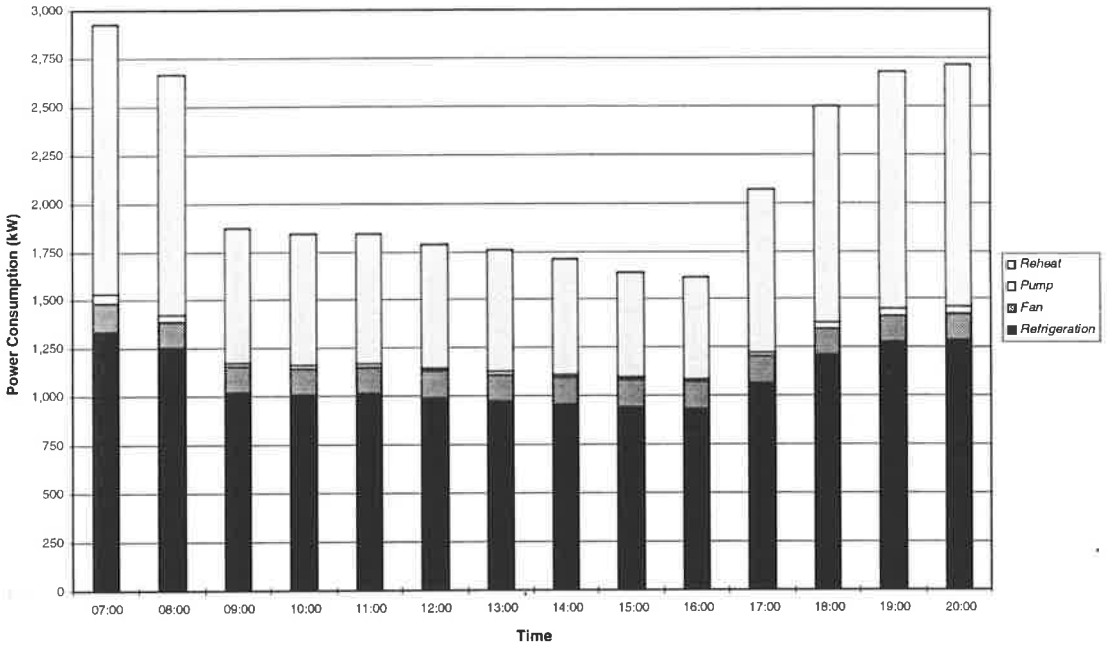


Figure D.14. Power consumption for a conventional CAV system for March. Kuala Lumpur. Ventilation air flow 10 L/s per person. Overcooling and reheat applied to constrain maximum zone humidity to 60%.

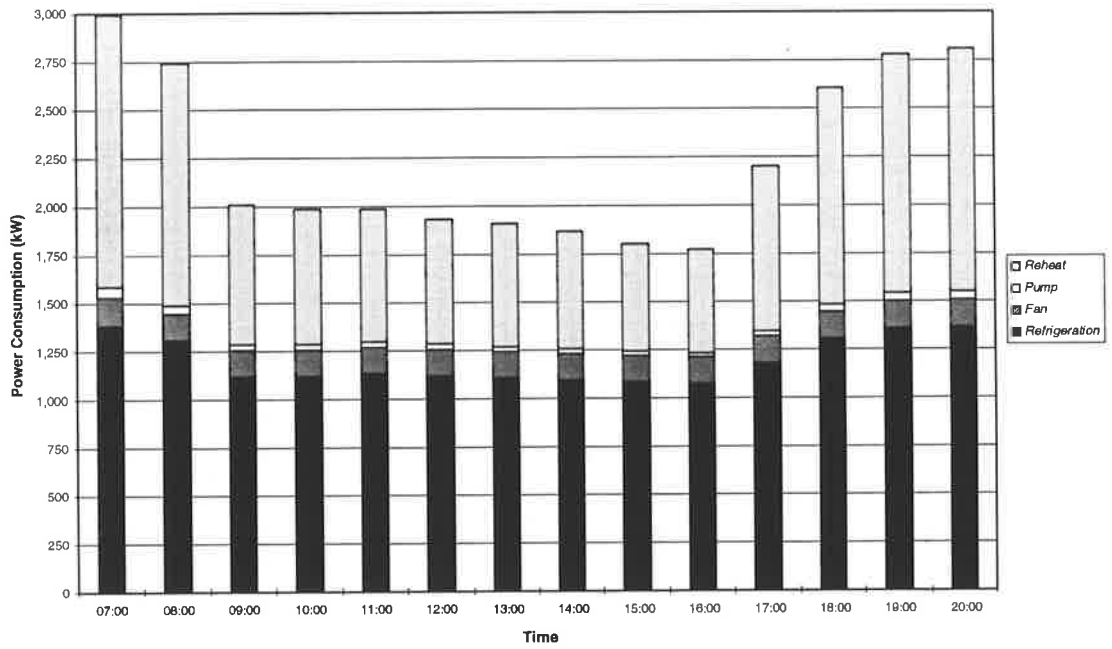


Figure D.15. Power consumption for a conventional CAV system for November. Kuala Lumpur. Ventilation air flow 2·5 L/s per person. Overcooling and reheat applied to constrain maximum zone humidity to 60%.

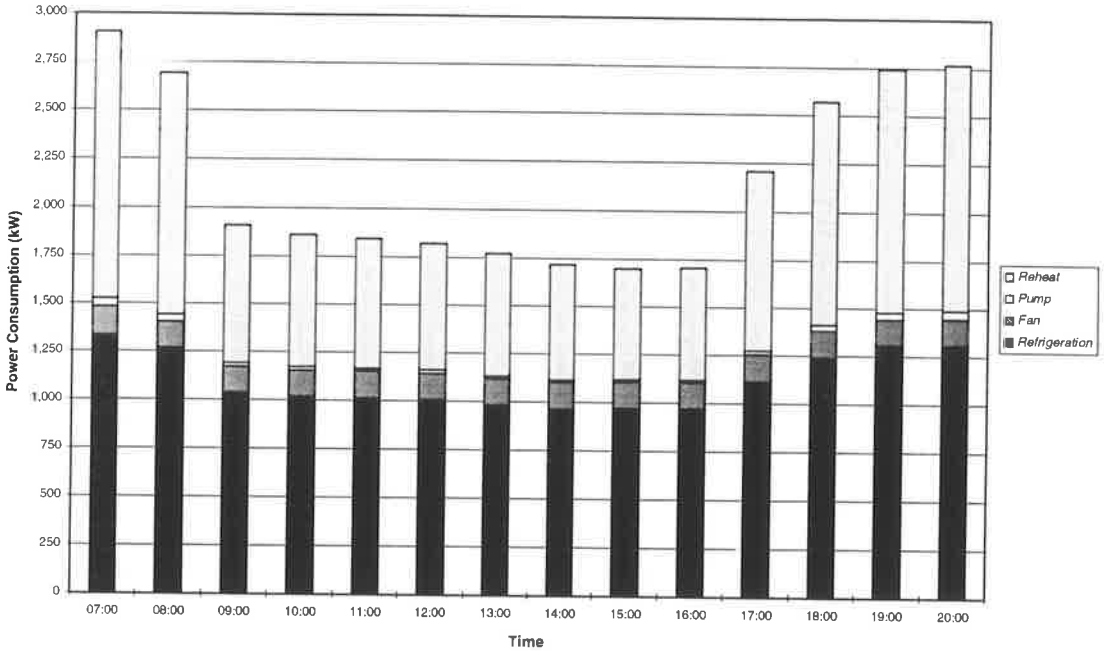


Figure D.16. Power consumption for a conventional CAV system for November. Kuala Lumpur. Ventilation air flow 10 L/s per person. Overcooling and reheat applied to constrain maximum zone humidity to 60%.

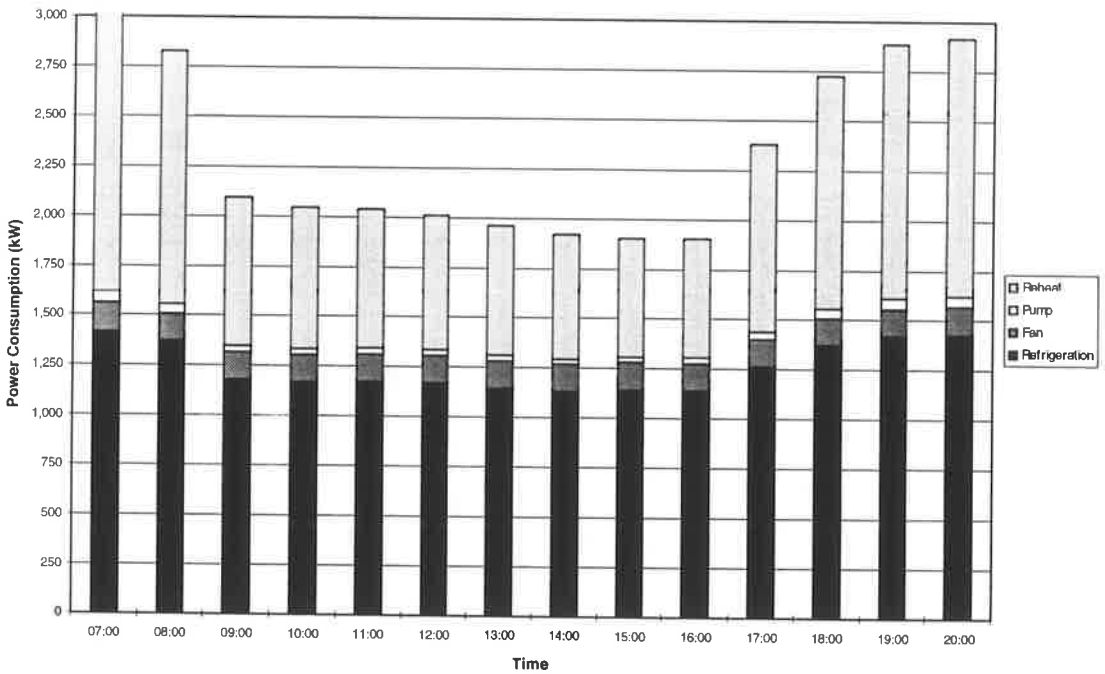


Figure D.17. Comparison of energy consumption for an HDP system without staging, and the same system with staging on zones 1 to 4. Kuala Lumpur, March. Ventilation air flow rate 2·5 L/s per person.

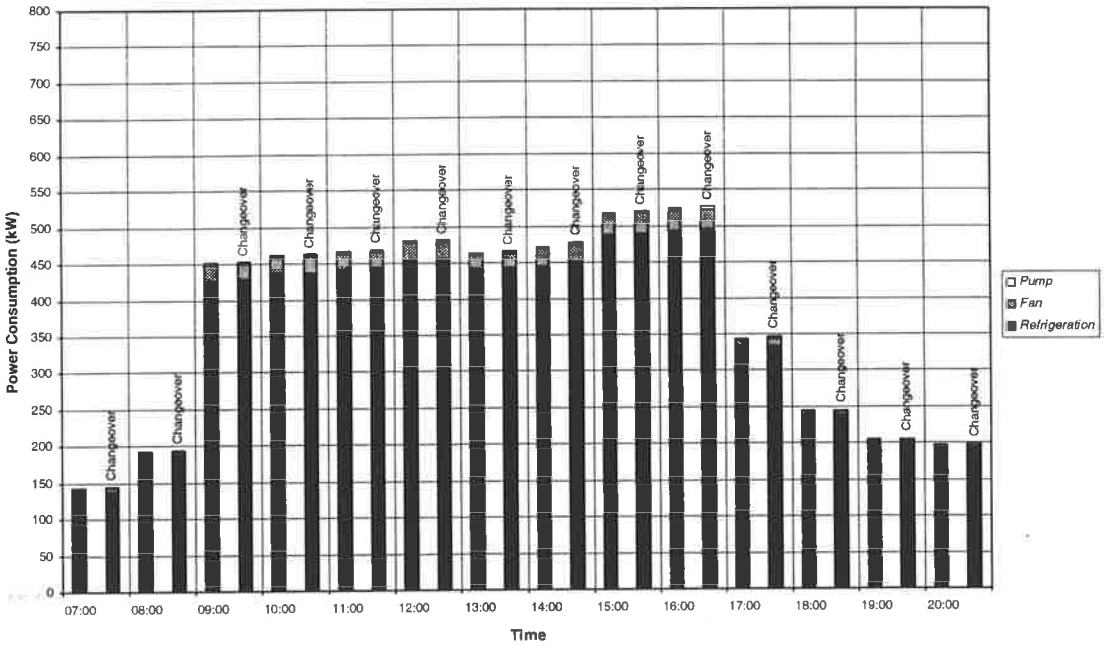
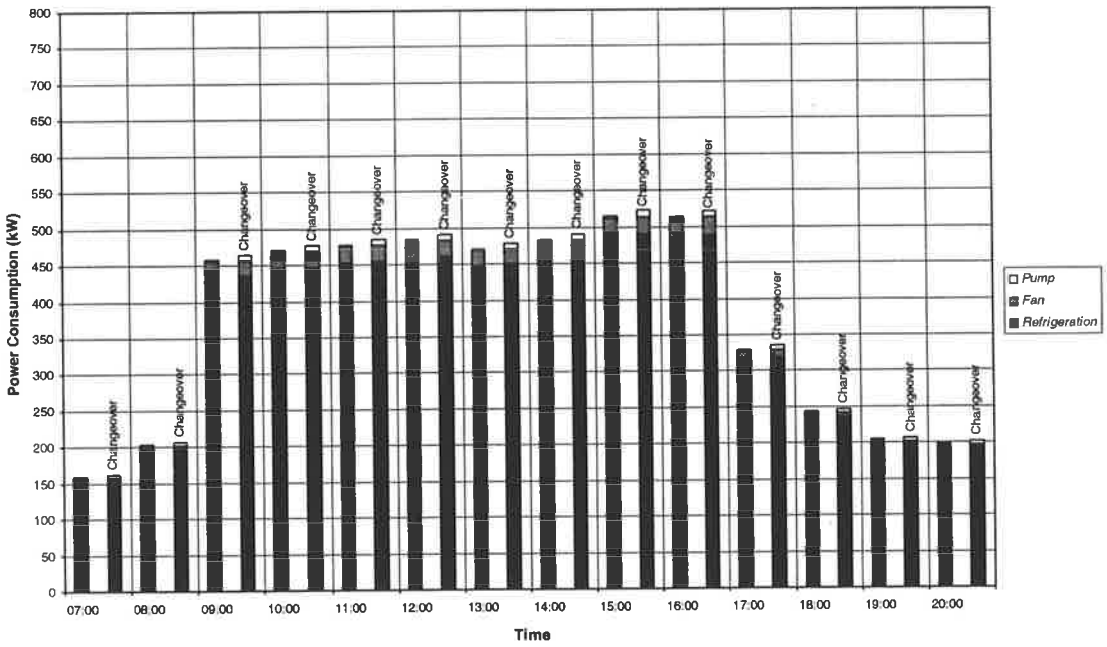


Figure D.18. Comparison of energy consumption for an HDP system without staging, and the same system with staging on zones 1 to 4. Kuala Lumpur, November. Ventilation air flow rate 10 L/s per person.



Appendix E. Water Flow Rates for Multistorey Building.

Figure E.1. Chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system (without staging). Adelaide, January. Ventilation air flow rate 2·5 L/s per person.

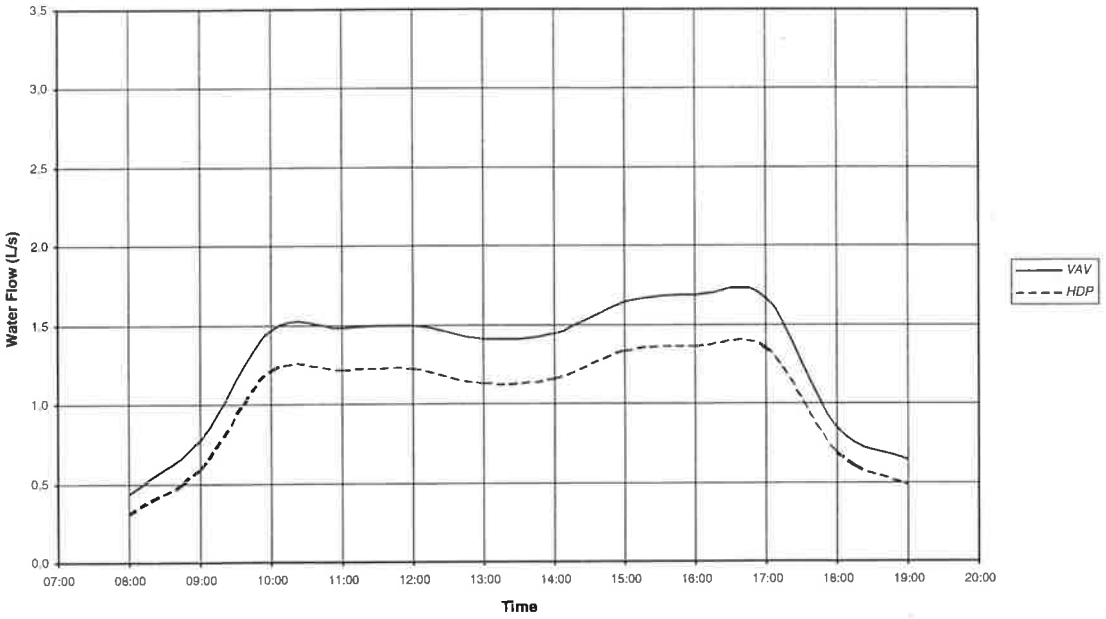


Figure E.2. Chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system (without staging). Adelaide, January. Ventilation air flow rate 10 L/s per person.

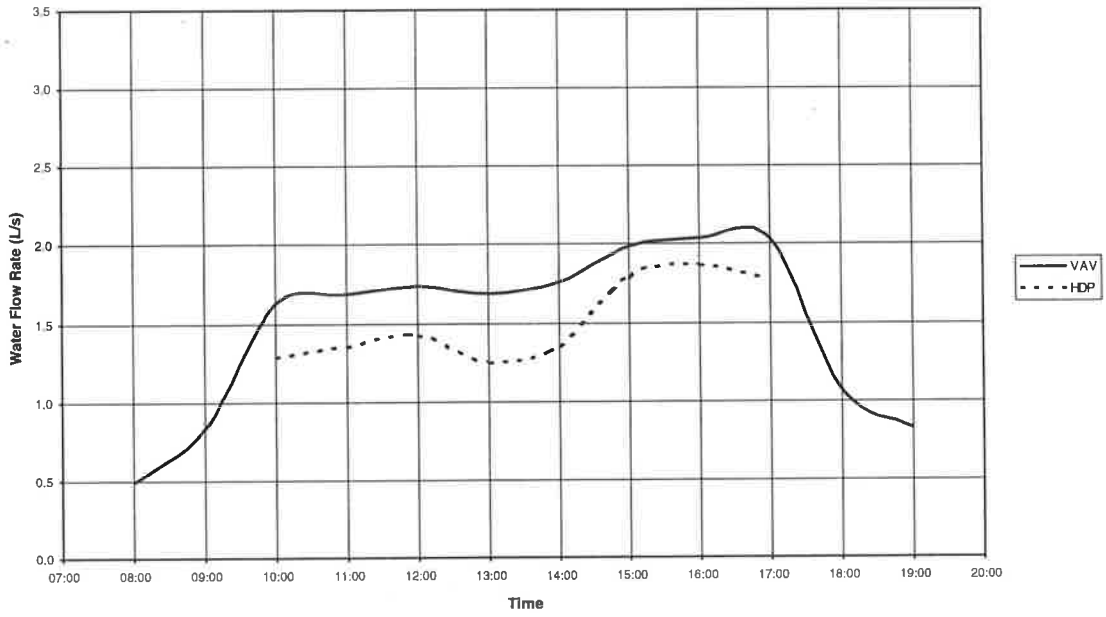


Figure E.3. Chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system (without staging). Adelaide, April. Ventilation air flow rate 2·5 L/s per person.

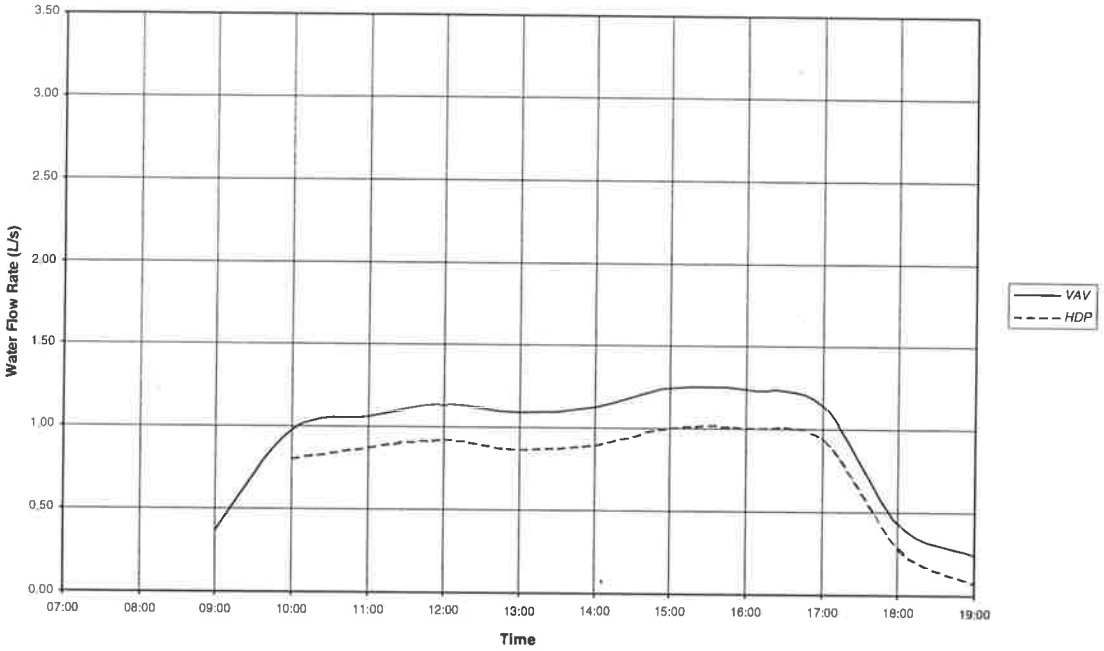


Figure E.4. Chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system (without staging). Adelaide, April. Ventilation air flow rate 10 L/s per person.

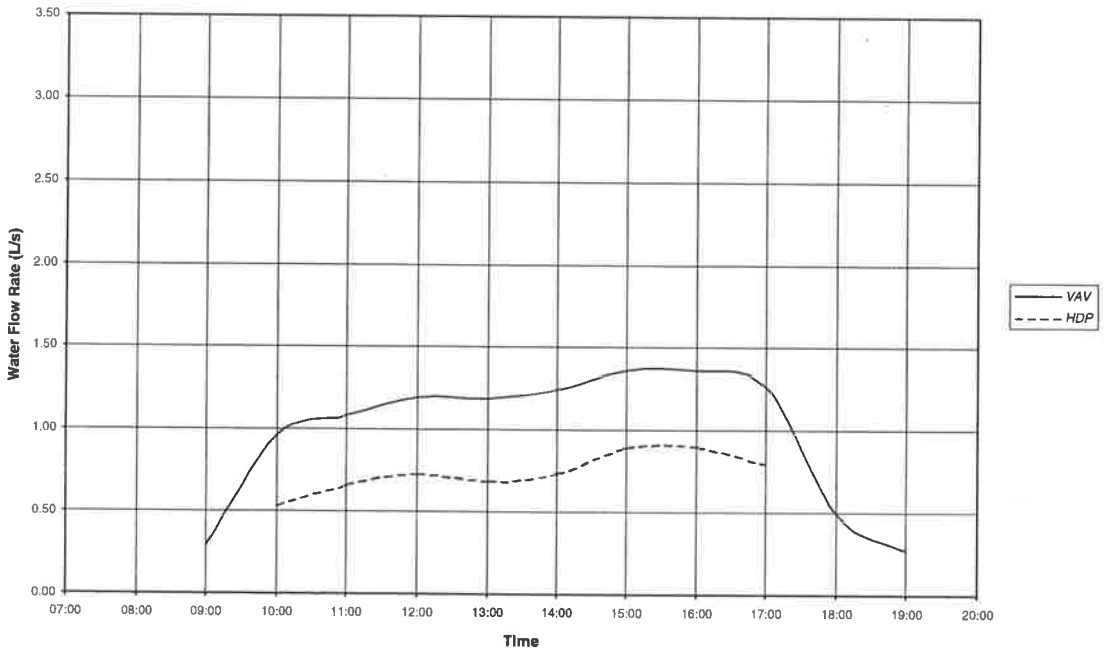


Figure E.5. Chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system (with and without staging). Kuala Lumpur, March. Ventilation air flow rate 2·5 L/s per person.

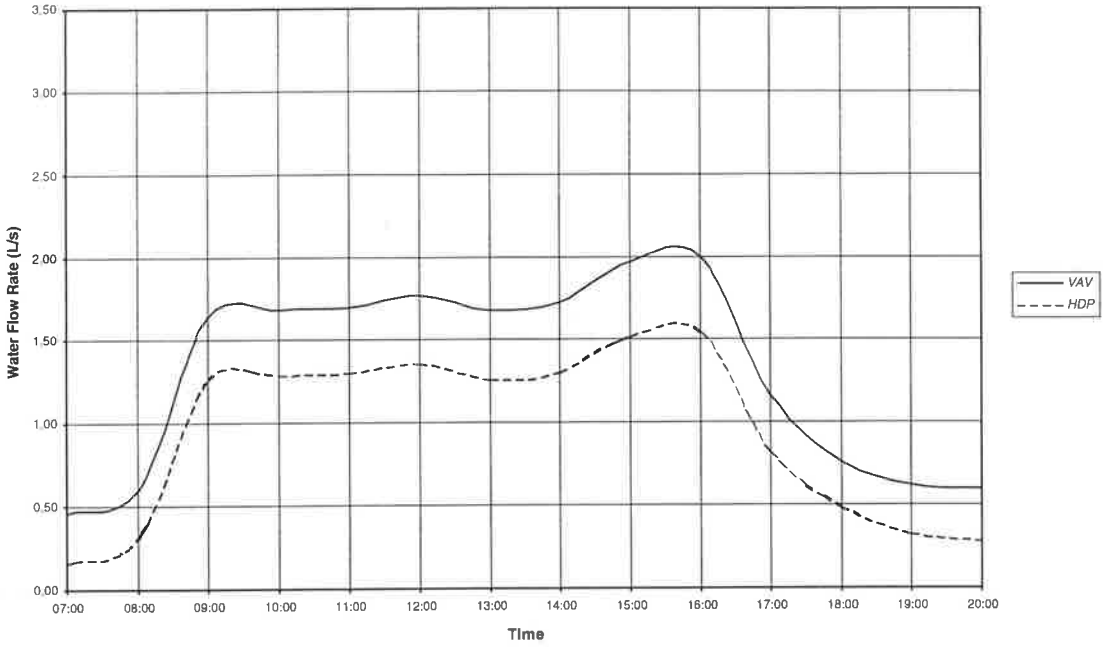


Figure E.6. Chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system (without staging). Kuala Lumpur, March. Ventilation air flow rate 10 L/s per person.

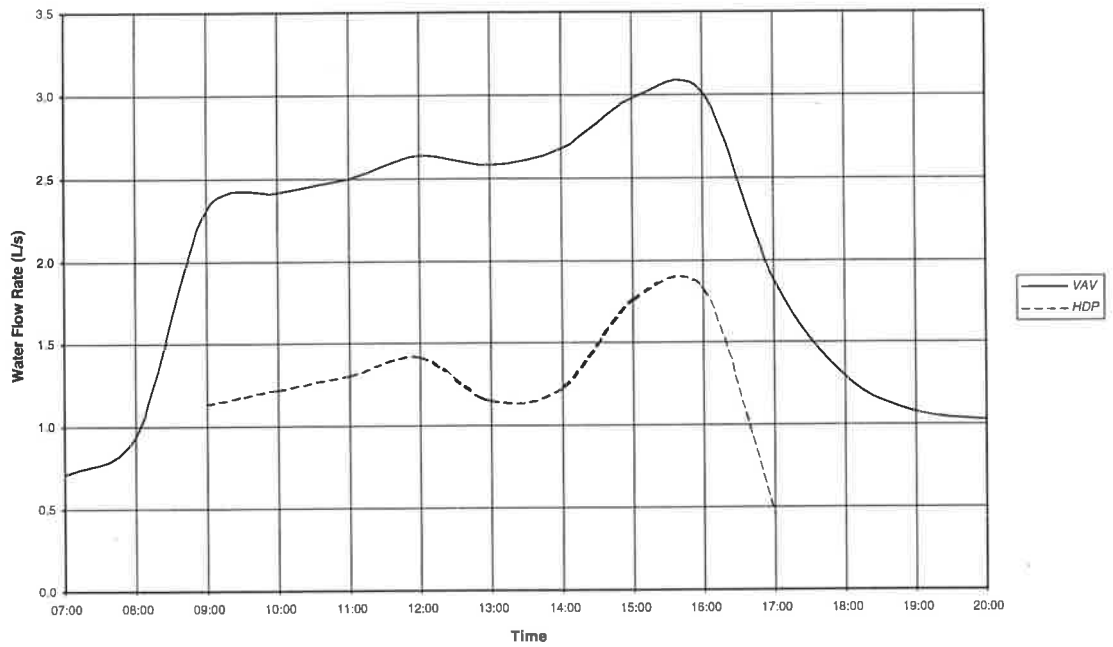


Figure E.7. Chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system (wtih and without staging). Kuala Lumpur, November. Ventilation air flow rate 2·5 L/s per person.

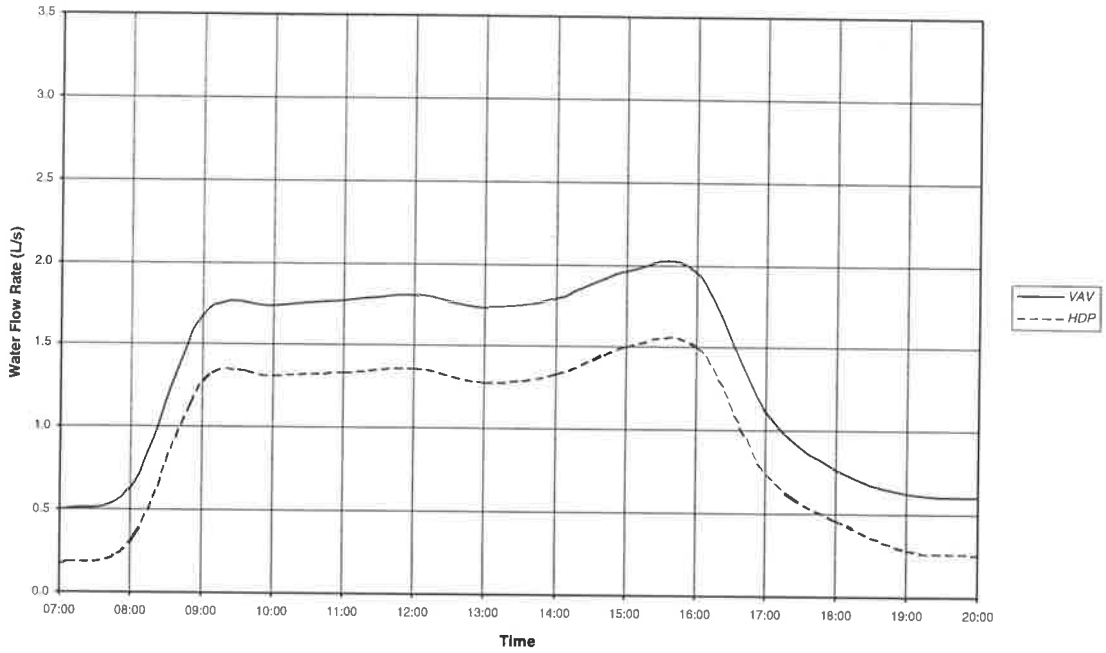
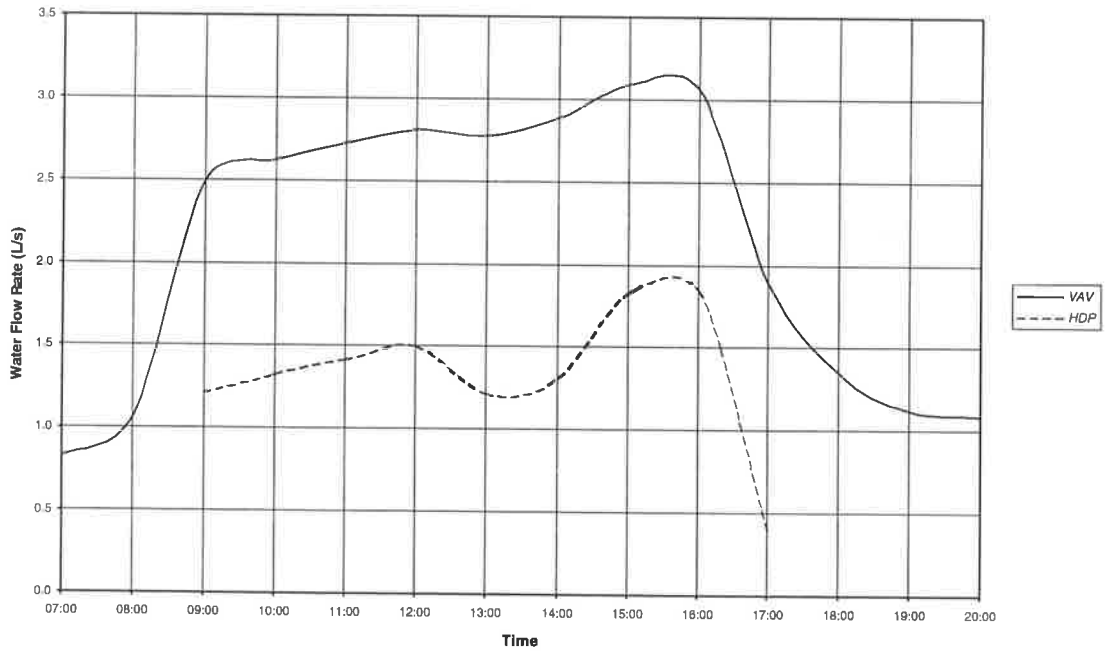


Figure E.8. Chilled water consumption *per floor* for a conventional multizone VAV system and an HDP system (without staging). Kuala Lumpur, November. Ventilation air flow rate 10 L/s per person.



Appendix F. Coil Selections and Peak Load Performance for the Multistorey Building.

This appendix tabulates the coil selections used in the air conditioning system performance simulations for the multistorey building (Section 13.3), together with details of the system performance under peak load conditions. The following abbreviations are used in the tables below:

AFV Air Face Velocity
APD Air Pressure Drop
CW Chilled Water
WPD Water Pressure Drop
WTR Water Temperature Rise
WV Water Velocity

F.1. VAV Systems.

F.1.1. Adelaide.

Table F.1. Cooling Coil Selection.

Width mm	Height mm	Height tubes	Fins/Inch	Rows	Passes per Circuit	Circuits
1800	1371·6	36	9	4	12	12

Table F.2. Cooling Coil Performance at Simultaneous Peak Load.

Ventilation L/s per person	WV m/s	CW Flow L/s	Refrigeration Cap (kW)			AFV m/s	APD Pa	WPD kPa	WTR °C
			Sens.	Lat.	Total				
2·5	1·009	2·115	86·52	7·82	94·34	2·511	149·1	22·38	10·64
10	1·114	2·398	96·64	8·45	105·1	2·511	149·5	27·83	10·455

Table F.3. Zone Relative Humidity at Simultaneous Peak Load.

Ventilation L/s per person	Zone 1 NW	Zone 2 NE	Zone 3 SE	Zone 4 SW	Zone 5 Interior
2·5	49·33	49·92	49·92	49·29	51·74
10	48·63	49·22	49·22	48·59	51·05

F.1.2. Kuala Lumpur.

Table F.4. Cooling Coil Selection.

Width mm	Height mm	Height tubes	Fins/Inch	Rows	Passes per Circuit	Circuits
1800	1371·6	36	9	4	8	18

Table F.5. Cooling Coil Performance at Simultaneous Peak Load.

Ventilation L/s per person	WV m/s	CW Flow L/s	Refrigeration Cap (kW)			AFV m/s	APD Pa	WPD kPa	WTR °C
			Sens.	Lat.	Total				
2·5	0·797	2·506	77·73	15·88	93·61	2·311	124·3	10·21	8·908
10	1·270	3·995	84·87	40·00	124·87	2·310	111·2	25·97	7·452

Table F.6. Zone Relative Humidity at Simultaneous Peak Load.

Ventilation L/s per person	Zone 1 NW	Zone 2 NE	Zone 3 SE	Zone 4 SW	Zone 5 Interior
2·5	50·73	51·53	51·53	50·65	53·26
10	51·45	52·24	52·24	51·36	53·97

F.2. CAV Systems.

F.2.1. Adelaide.

Table F.7. Cooling Coil Selection.

Zone	Width mm	Height mm	Height tubes	Fins/Inch	Rows	Passes per Circuit	Circuits
1	1100	571·5	15	9	4	20	3
2	1100	571·5	15	9	4	20	3
3	1400	381	10	9	4	20	2
4	1400	381	10	9	4	20	2
5	1400	609·6	16	9	4	16	4

Table F.8. Cooling Coil Performance for Each Coil at its Individual Peak Load.

Zone	Ventilation L/s per person	WV m/s	CW Flow L/s	Refrigeration Cap (kW)			AFV m/s	APD Pa	WPD kPa	WTR °C
				Sens.	Lat.	Total				
1	2·5	1·160	0·608	21·44	5·63	27·07	2·482	138·4	29·03	10·617
	10	1·126	0·590	23·18	4·21	27·39	2·482	142·7	27·51	11·070
2	2·5	1·113	0·584	19·80	6·16	25·96	2·331	124·2	27·02	10·613
	10	1·031	0·541	20·68	4·54	25·22	2·331	128·3	23·62	11·130
3	2·5	1·055	0·369	14·78	3·82	18·60	2·030	104·2	30·90	12·037
	10	1·054	0·368	15·45	3·45	18·90	2·030	105·3	30·85	12·240
4	2·5	1·196	0·418	17·32	3·50	20·82	2·350	131·5	41·11	11·891
	10	1·280	0·447	19·01	3·37	22·38	2·350	132·5	46·38	11·938
5	2·5	1·230	0·859	29·12	8·53	37·65	2·500	138·3	35·53	10·451
	10	1·238	0·866	30·43	7·87	38·31	2·500	139·9	35·97	10·558

Table F.9. Zone Relative Humidity at Individual and Simultaneous Peak Load.

Peak Zone	Ventilation L/s per person	Zone 1 NW	Zone 2 NE	Zone 3 SE	Zone 4 SW	Zone 5 Interior
1	2·5	**52·64**	58·88	59·23	59·05	56·09
	10	**51·85**	56·53	56·85	56·67	55·29
2	2·5	60·43	**53·27**	60·81	60·65	56·52
	10	57·93	**52·69**	58·35	58·15	55·91
3	2·5	59·63	59·58	**53·98**	59·83	55·44
	10	58·72	58·67	**53·65**	58·92	55·06
4	2·5	57·42	57·48	57·83	**52·85**	55·07
	10	56·60	56·68	57·02	**52·23**	54·42
5	2·5	59·63	59·58	53·98	59·83	**55·44**
	10	58·72	58·67	53·65	58·92	**55·06**
Overall	2·5	57·42	57·48	57·83	52·85	55·06
	10	56·60	56·68	57·02	52·22	54·41

F.2.1. Kuala Lumpur.

Table F.10. Cooling Coil Selection.

Zone	Width mm	Height mm	Height tubes	Fins/Inch	Rows	Passes per Circuit	Circuits
1	1000	609·6	16	9	4	16	4
2	1000	609·6	16	9	4	16	4
3	1000	609·6	16	9	4	16	4
4	1000	609·6	16	9	4	16	4
5	1350	609·6	16	9	4	8	8

Table F.11. Cooling Coil Performance for Each Coil at its Individual Peak Load.

Zone	Ventilation L/s per person	WV m/s	CW Flow L/s	Refrigeration Cap (kW)			AFV m/s	APD Pa	WPD kPa	WTR °C
				Sens.	Lat.	Total				
1	2·5 10	0·988 1·339	0·690 0·936	19·42 20·82	7·60 12·40	27·02 33·22	2·343 2·343	120·7 111·8	16·29 31·08	9·332 8·463
2	2·5 10	0·914 1·088	0·639 0·761	17·12 17·41	8·24 11·40	25·36 28·81	2·105 2·105	100·6 95·4	14·24 19·28	9·468 9·035
3	2·5 10	0·973 1·162	0·680 0·812	18·05 18·32	8·48 11·79	26·53 30·12	2·220 2·220	108·8 103·0	15·88 21·62	9·300 8·841
4	2·5 10	1·030 1·422	0·720 0·994	19·99 21·43	7·75 13·14	27·74 34·57	2·414 2·414	125·4 116·0	17·52 34·60	9·191 8·294
5	2·5 10	0·827 1·128	1·156 1·577	27·47 29·43	10·72 18·11	38·18 47·55	2·456 2·456	126·6 119·0	8·37 14·30	7·874 7·188

Table F.12. Zone Relative Humidity at Individual and Simultaneous Peak Load.

Peak Zone	Ventilation L/s per person	Zone 1 NW	Zone 2 NE	Zone 3 SE	Zone 4 SW	Zone 5 Interior
1	2·5 10	**53·54** **54·08**	62·76 69·07	62·73 69·02	62·62 68·93	56·26 56·81
2	2·5 10	65·37 70·14	**54·50** **55·04**	65·43 70·17	65·35 70·11	57·90 58·51
3	2·5 10	64·87 69·65	64·98 69·74	**54·20** **54·78**	64·86 69·63	57·69 58·32
4	2·5 10	62·44 69·30	62·54 69·43	62·49 69·38	**53·39** **54·03**	56·13 56·77
5	2·5 10	62·44 69·30	62·54 69·43	62·49 69·38	53·39 54·03	**56·13** **56·77**
Overall	2·5 10	59·86 60·14	61·89 66·78	61·86 66·73	58·91 59·20	56·10 56·45

F.3. HDP Systems.

F.3.1. Adelaide.

Table F.13. Cooling Coil Selection - Ventilation Air Flow Rate 2·5 L/s per person.

Zone	Width mm	Height mm	Height tubes	Fins/Inch	Rows	Passes per Circuit	Circuits
O/A	1500	1828·8	48	9	6	3	96
1	1550	381	10	9	4	20	2
2	1550	381	10	9	4	20	2
3	1200	381	10	9	4	20	2
4	1200	381	10	9	4	20	2
5	1350	571·5	15	9	4	20	3

Table F.14. Cooling Coil Selection - Ventilation Air Flow Rate 10 L/s per person.

Zone	Width mm	Height mm	Height tubes	Fins/Inch	Rows	Passes per Circuit	Circuits
O/A	3600	2286	60	9	6	3	120
1	1200	457·2	12	9	4	16	3
2	1200	457·2	12	9	4	16	3
3	1000	457·2	12	9	4	16	3
4	1000	457·2	12	9	4	16	3
5	1350	609·6	16	6·18	3	16	3

Table F.15. Outdoor Air Cooling Coil Performance at Simultaneous Peak Load.

Ventilation L/s per person	WV m/s	CW Flow L/s	Refrigeration Cap (kW)			AFV m/s	APD Pa	WPD kPa	WTR °C
			Sens.	Lat.	Total				
2·5	1·491	25	88·63	23·92	112·55	0·942	36·7	17·62	1·072
10	1·192	25	338·85	78·77	417·61	1·255	58·9	16·49	3·980

Table F.16. Cooling Coil Performance for Each Return Air Coil at its Individual Peak Load.

Zone	Ventilation L/s per person	WV m/s	CW Flow L/s	Refrigeration Cap (kW)			AFV m/s	APD Pa	WPD kPa	WTR °C
				Sens.	Lat.	Total				
1	2·5 10	1·161 1·108	0·406 0·581	18·73 17·17	0·56 0·01	19·29 17·18	2·500 2·543	152·9 164·2	42·56 23·20	11·350 7·062
2	2·5 10	1·043 0·888	0·365 0·465	17·49 15·90	0·54 0·01	18·03 15·91	2·345 2·376	138·8 148·5	33·22 15·79	11·803 8·163
3	2·5 10	0·810 0·684	0·283 0·359	12·82 11·26	0·53 0·01	13·35 11·27	2·222 2·043	127·4 117·8	16·75 8·47	11·259 7·501
4	2·5 10	1·064 1·047	0·372 0·549	14·89 13·37	0·57 0·01	15·46 13·37	2·582 2·402	159·5 148·6	26·87 17·64	9·927 5·821
5	2·5 10	1·121 1·329	0·588 0·697	23·58 15·02	2·70 0·00	26·28 15·02	2·504 1·812	148·3 24·6	33·02 36·43	10·670 5·146

Table F.17. Zone Relative Humidity at Individual and Simultaneous Peak Load.

Peak Zone	Ventilation L/s per person	Zone 1 NW	Zone 2 NE	Zone 3 SE	Zone 4 SW	Zone 5 Interior
1	2·5 10	**51·13** **48·28**	56·30 48·29	55·73 48·45	55·69 48·44	54·75 48·52
2	2·5 10	57·02 48·18	**51·66** **47·99**	56·55 48·15	56·56 48·14	54·80 48·16
3	2·5 10	56·63 49·90	53·65 48·62	**52·57** **49·89**	55·71 49·95	54·40 50·03
4	2·5 10	53·14 50·36	55·81 50·48	54·74 50·51	**51·41** **50·39**	54·39 50·60
5	2·5 10	56·63 49·90	53·65 48·62	52·57 49·89	55·71 49·95	**54·40** **50·03**
Overall	2·5 10	53·19 50·80	55·81 50·48	54·75 50·51	51·41 50·39	54·65 50·60

F.3.1. Kuala Lumpur.

Table F.18. Cooling Coil Selection - Ventilation Air Flow Rate 2·5 L/s per person.

Zone	Width mm	Height mm	Height tubes	Fins/Inch	Rows	Passes per Circuit	Circuits
O/A	1500	1828·8	48	9	6	3	96
1	1200	457·2	12	6·18	4	12	4
2	1200	457·2	12	6·18	4	12	4
3	1200	457·2	12	6·18	4	12	4
4	1200	457·2	12	6·18	4	12	4
5	1350	571·5	15	9	4	20	3

Table F.19. Cooling Coil Selection - Ventilation Air Flow Rate 10 L/s per person.

Zone	Width mm	Height mm	Height tubes	Fins/Inch	Rows	Passes per Circuit	Circuits
O/A	3600	2743·2	72	9	6	3	144
1	1200	457·2	12	9	4	16	3
2	1200	457·2	12	9	4	16	3
3	1200	457·2	12	9	4	16	3
4	1200	457·2	12	9	4	16	3
5	1350	457·2	12	6·18	3	6	6

Table F.20. Outdoor Air Cooling Coil Performance at Simultaneous Peak Load.

Ventilation L/s per person	WV m/s	CW Flow L/s	Refrigeration Cap (kW)			AFV m/s	APD Pa	WPD kPa	WTR °C
			Sens.	Lat.	Total				
2·5	1·789	30	68·76	94·00	162·76	0·942	36·5	24·66	1·444
10	1·789	45	301·27	408·75	710·03	1·046	42·7	42·10	3·759

Table F.21. Cooling Coil Performance for Each Return Air Coil at its Individual Peak Load.

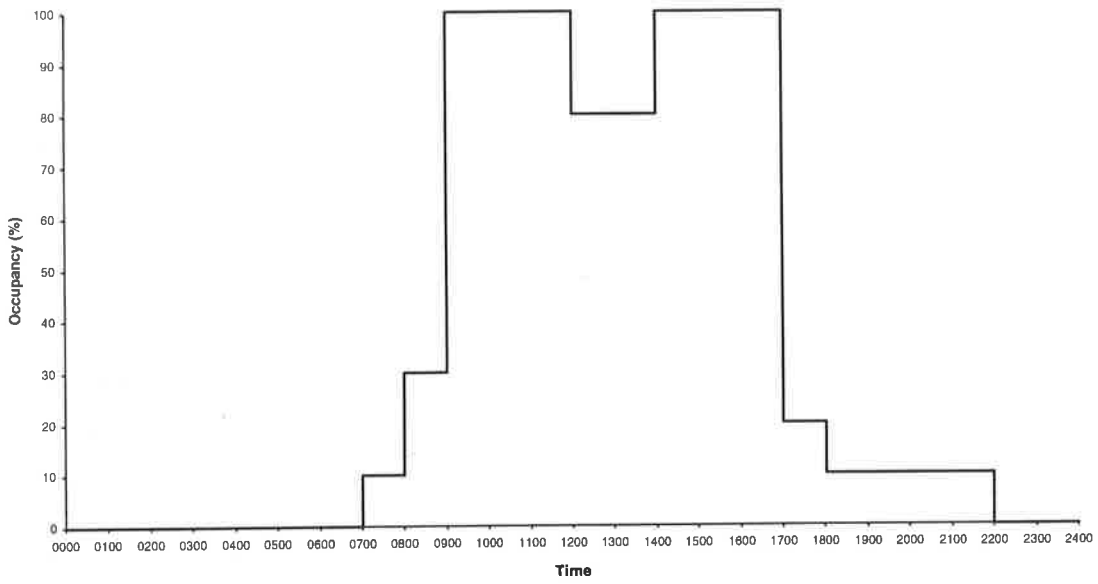
Peak Zone	Ventilation L/s per person	WV m/s	CW Flow L/s	Refrigeration Cap (kW)			AFV m/s	APD Pa	WPD kPa	WTR °C
				Sens.	Lat.	Total				
1	2·5 10	0·839 1·073	0·586 0·563	16·85 15·36	0·55 0·01	17·39 15·37	2·457 2·308	117·9 138·7	11·06 21·89	7·078 6·524
2	2·5 10	0·684 0·805	0·478 0·422	15·01 13·49	0·54 0·01	15·55 13·50	2·202 2·054	100·1 117·5	7·78 13·28	7·762 7·644
3	2·5 10	0·743 0·829	0·519 0·434	15·90 14·34	0·54 0·01	16·43 14·35	2·326 2·178	108·6 129·0	8·98 13·99	7·551 7·887
4	2·5 10	0·884 1·136	0·618 0·595	17·40 15·91	0·55 0·01	17·95 15·92	2·533 2·384	123·4 145·1	12·11 24·16	6·933 6·387
5	2·5 10	1·166 1·330	0·611 1·394	22·21 13·72	2·84 0·00	25·05 13·72	2·361 2·237	134·7 34·0	37·84 15·04	9·784 2·348

Table F.22. Zone Relative Humidity at Individual and Simultaneous Peak Load.

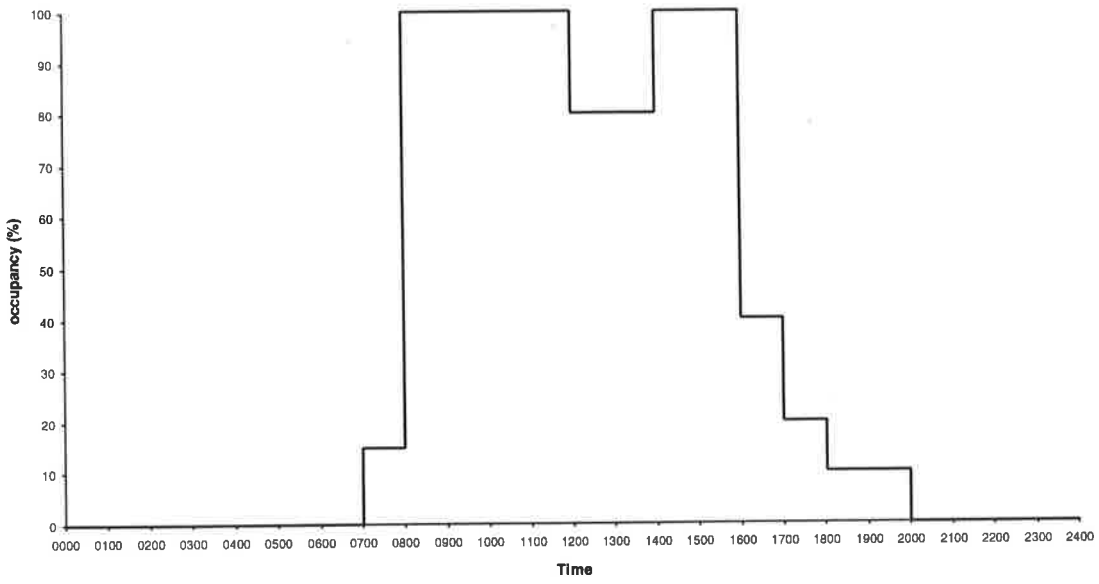
Zone	Ventilation L/s per person	Zone 1 NW	Zone 2 NE	Zone 3 SE	Zone 4 SW	Zone 5 Interior
1	2·5 10	**53·97** **50·88**	57·05 51·00	57·17 51·10	56·49 51·01	55·09 51·15
2	2·5 10	57·85 50·55	**54·45** **50·44**	57·39 50·56	57·89 50·49	55·59 50·59
3	2·5 10	57·73 50·47	57·48 49·77	**54·22** **49·71**	57·68 49·96	55·52 49·89
4	2·5 10	56·60 50·98	57·10 51·00	56·98 51·11	**53·84** **50·99**	55·04 51·10
5	2·5 10	56·60 50·98	57·10 51·00	56·98 51·11	53·84 50·99	**55·04** **51·10**
Overall	2·5 10	55·02 51·00	57·00 50·97	56·99 51·01	54·75 50·93	55·09 51·07

Appendix G. Occupancy Levels for the Multistorey Building.

Adelaide



Kuala Lumpur



Appendix H. Multistorey Building: Miscellaneous Plots.

H.1. Outdoor Air Supply Air Temperature[167].

Figure H.1. Supply air temperature for outdoor air treatment unit in HDP system. Adelaide.

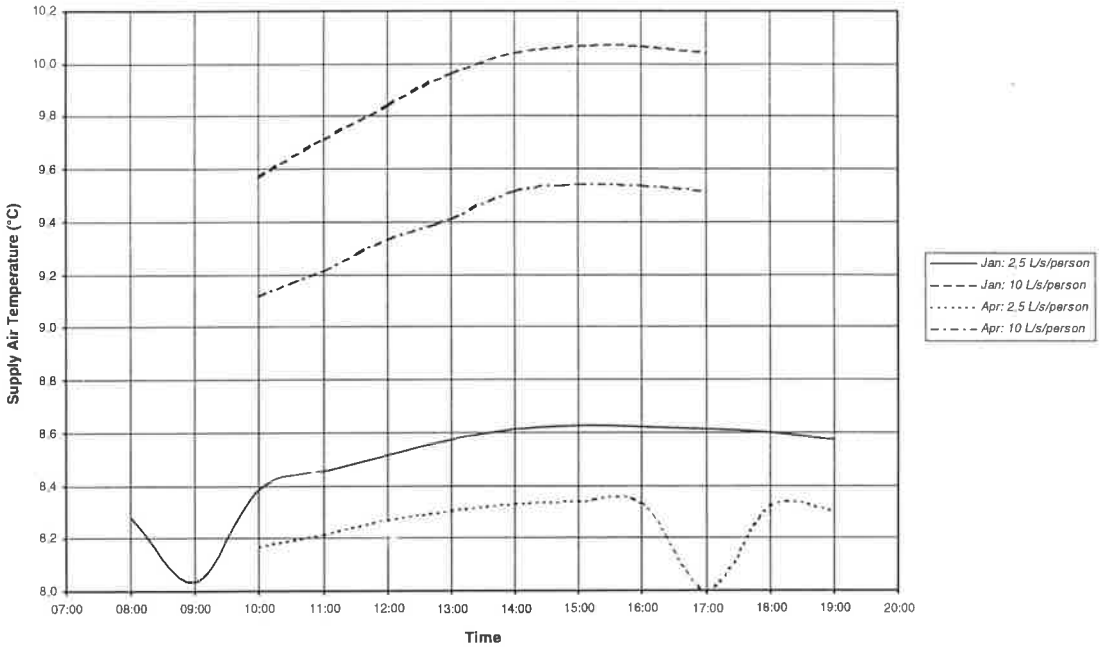
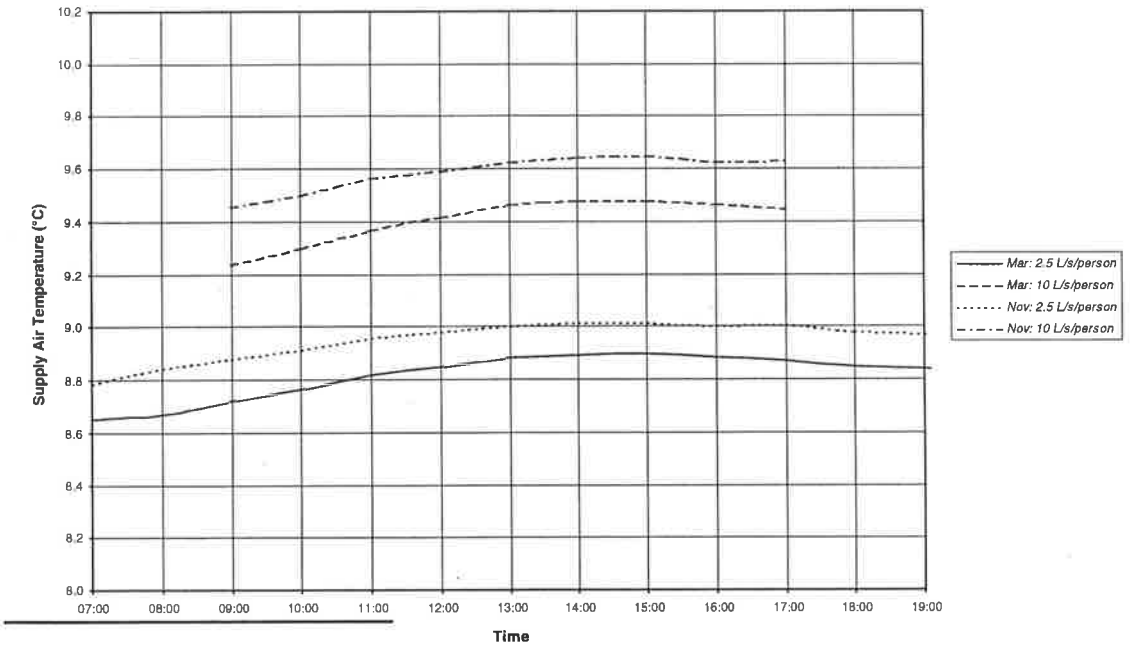


Figure H.2. Supply air temperature for outdoor air treatment unit in HDP system. Kuala Lumpur.



167 The supply air temperature plotted here is the temperature of the air supplied to the terminal units, and contains a duct temperature gain of 0·4 °C.

Appendix I. Attakarn Prasit Apartments: Performance Plots.

Figure I.1. Zone relative humidity as a function of ambient dry-bulb temperature. Outdoor air flow rate 600 L/s.

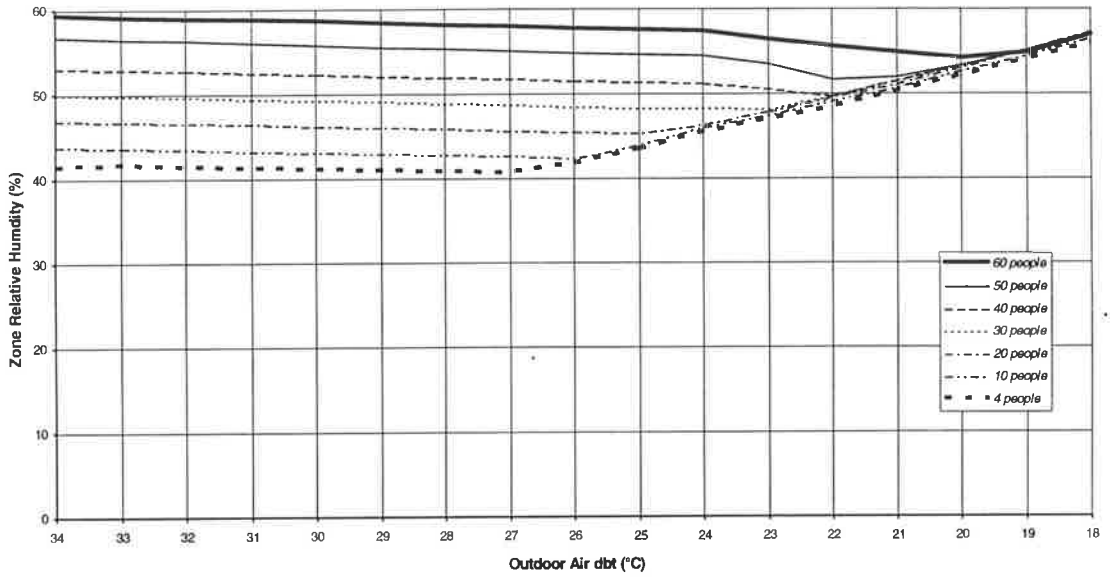


Figure I.2. Zone relative humidity as a function of ambient dry-bulb temperature. Outdoor air flow rate 300 L/s.

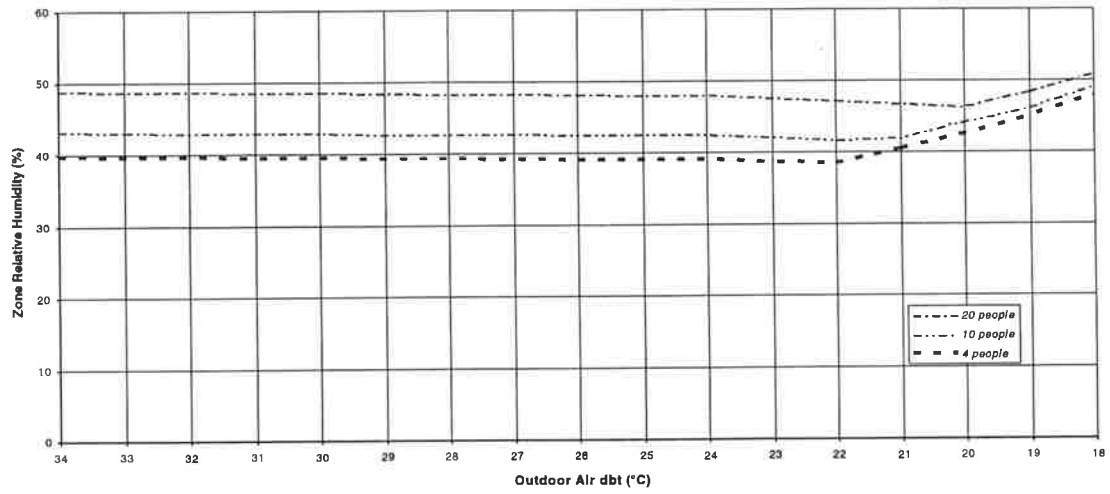
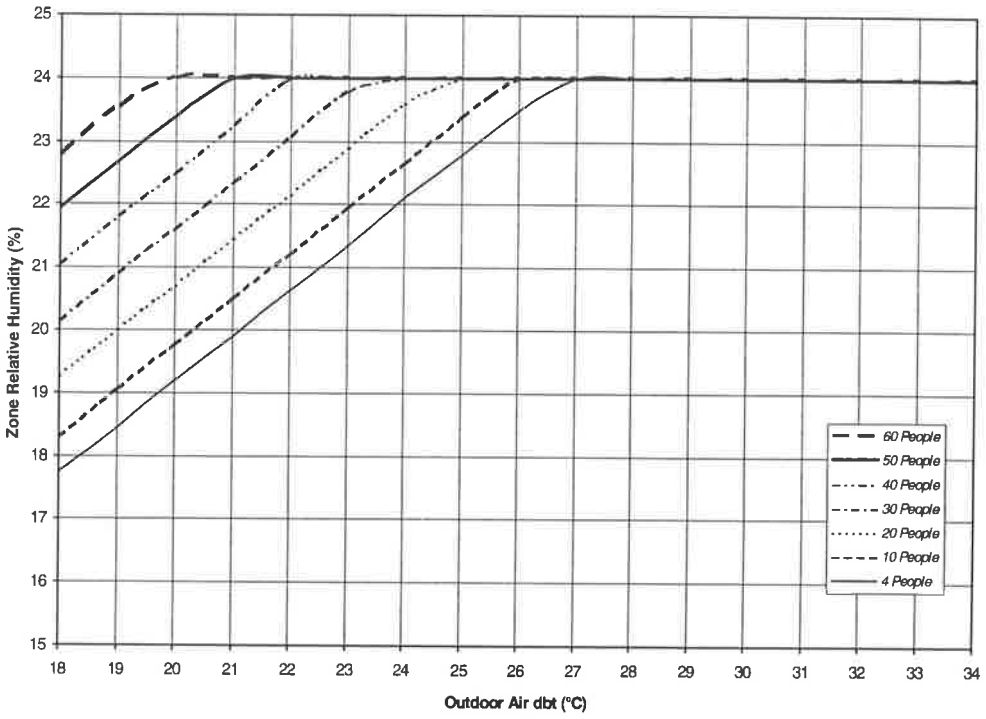


Figure I.3. Room dry-bulb temperature as a function of oudoor air dry-bulb temperature. Outdoor air flow rate maintained at 600 L/s. Room temperature maintained at 24°C at higher outdoor air temperatures by modulating chilled water flow through return air coil. At lower air temperatures, reheat is applied as necessary to maintain room temperature at 24°C up to the maximum possible when proportional control is lost, and room dry-bulb temperature decreases.



References.

Afnan, D., 1990, Condensation forming on dehumidification coils, *Honours Thesis*, Dept. Mech. Eng., University of Adelaide.

Ahuja, R.K., Magnanti, T.L. and Orlin, J.B., 1993, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, N.J.

Akiyama, M., Murakoshi, T., Sugiyama, H., Cheng, K.C. and Nishiwaki, I., 1988, Numerical solution of convective heat transfer for reverse transition in the bend tube for a low-Reynolds-number turbulent model, *JSME International Journal*, Series II, Vol. 31, pp. 289-298.

ARI, 1981, *ARI Standard 410-81, Forced circulation air-cooling and air-heating coils*, Air Conditioning and Refrigeration Institute, American National Standard.

Arora, J.S., 1989, *Introduction to Optimum Design*, McGraw-Hill, New York.

ASHRAE, 1973, *Thermophysical Properties of Refrigerants*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta.

ASHRAE, 1985, *ASHRAE Handbook - Fundamentals*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta.

ASHRAE, 1988, *ASHRAE Handbook - Equipment*, American Society of Heating, Refrgierating and Air-Conditioning Engineers, Atlanta.

ASHRAE, 1989a, *ASHRAE Standard 55-1989, Thermal environmental conditions for human occupancy*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta.

ASHRAE, 1989b, *ASHRAE Standard 62-1989, Ventilation of acceptable indoor air quantity*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta.

ASHRAE, 1993, *ASHRAE Handbook - Fundamentals*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta.

Australian Construction Services, 1985, *User Guide for the APEC Energy Simulation Computer Program ESPII*, Mechanical Engineering Services Design Aids No. DA12.

Australian Construction Services, 1987, *User Guide for Computer Program TEMPER*, Mechanical Engineering Services Design Aids No. DA14.

Australian Construction Services, 1988a, *Air conditioning systems - load estimation and associated psychrometrics*, Mechanical Engineering Services Design Aids No. DA9.

Australian Construction Services, 1988b, *User Guide for the Computer Program Donkey*, 2nd Edition, Mechanical Engineering Services Design Aids No. DA4.

Australian Construction Services, 1991, *User Guide for the Computer Program Camel*, 3rd Edition, Mechanical Engineering Services Design Aids No. DA5.

Booch, G., 1991, *Object-Oriented Design with Applications*, Benjamin/Cummings, Redwood City, Calif.

Bird, R.B., Stewart, W.E. and Lightfoot, E.N., 1960, *Transport Phenomena*, John Wiley and Sons, New York.

de Boor, C., 1977, Package for computing with B-splines, *SIAM J. Numer. Anal.*, Vol. 14, pp. 441-472.

Bowman, R.A., Mueller, A.C. and Nagle, W.M., 1940, Mean temperature difference in design, *ASME Transactions*, Vol. 62, pp. 283-294.

Brent, R.P., 1971, An algorithm with guaranteed convergence for finding the zero of a function, *Computer Journal*, Vol. 14, pp. 422-425.

Brock, J.R. and Bird, R.B., 1955, *A.I.Ch.E. J.*, Vol. 1, p. 174.

Brown, G., 1954, Theory of moist air heat exchangers, *Transactions of the Royal Institute of Technology, Stockholm, Sweden*, Nr. 77, pp. 12-15.

Brown, M.R. and Mathieson, P.J., 1991, Indoor air quality in commercial buildings, Australian Refrigeration, Air Conditioning and Heating, Vol. 45, No. 12, pp. 29-34.

Bureau of Meteorology, 1971, *Climatic Survey. Australia: Region 1 - South Australia.*

Butterfield, K.R., 1976, The computation of all the derivatives of a B-spline basis, *J. Inst. Maths. Applics.*, Vol. 17, pp. 15-25.

Carrier, W.H. and Anderson, S.W., 1942, The resistance to heat flow through finned tubing, *ASHVE Trans.*, Vol. 50, pp. 117-152.

Chae, H.B., Schmidt, J.W. and Moldover, M.R., 1990, Surface tension of refrigerants R123 and R134a, *J. Chem. Eng. Data*, Vol. 35, pp. 6-8.

Chan, C.Y. and Haselden, G.G., 1981a, Computer-based refrigerant thermodynamic properties. Part 1: Basic equations, *Int. J. Refrigeration*, Vol. 4, pp. 7-12.

Chan, C.Y. and Haselden, G.G., 1981b, Computer-based refrigerant thermodynamic properties. Part 2: Program listings, *Int. J. Refrigeration*, Vol. 4, pp. 52-60.

Chan, C.Y. and Haselden, G.G., 1981c, Computer-based refrigerant thermodynamic properties. Part 3: Use of the program in the computation of standard refrigeration cycles, *Int. J. Refrigeration*, Vol. 4, pp. 131-134.

Chilton, T.H. and Colburn, A.P., 1934, Mass transfer (absorption) coefficients, *Ind. Eng. Chem.*, Vol. 26, pp. 1183-1187.

Clarke, J.A., 1985, *Energy Simulation in Building Design*, Adam Hilger, Bristol.

Cleland, A.C., 1986, Computer sub-routines for rapid evaluation of refrigerant thermodynamic properties, *Int. J. Refrigeration*, Vol. 9, pp. 346-351.

Colburn, A.P., 1933, A method of correlating forced convection heat transfer data and a comparison with fluid friction, *AIChE Trans.*, Vol. 29, pp. 174-210.

Colebrook, C.F., 1939, Turbulent flow in pipes with particular reference to the transition region between smooth and rough pipe laws, *Journal of the Institution of Civil Engineers*, Vol. 11, pp. 133-156.

Collins, W.M. and Dennis, S.C.R., 1975, The steady motion of a viscous fluid in a curved tube, *Quart. J. Mech. Appl. Math.*, Vol. 28, pp. 133-156.

Cornish, R.J., 1928, Flow in a pipe of rectangular cross section, *Proceedings of the Royal Society*, Vol. 120(A), pp. 691-700.

Cox, J.E. and Miró, C.R., 1990, Special briefing for government officials: ASHRAE-sponsored forum covered ventilation design standard, *ASHRAE Journal*, Vol. 33, No. 1, pp. 14-16.

Cox, M.G., 1972, The numerical evaluation of B-splines, *J. Inst. Maths. Applics.*, Vol. 10, pp. 134-149.

Cox, M.G., 1988, The least squares solution of overdetermined linear equations having band or augmented band structure, *IMA Journal of Numerical Analysis*, Vol. 1, pp. 3-22.

Dierckx, P., 1977, An algorithm for least-squares fitting of cubic spline surfaces to functions on a rectilinear mesh over a rectangle, *J. Computational and Applied Mathematics*, Vol. 3, pp. 113-129.

Dierckx, P., 1981, An algorithm for surface-fitting with spline functions, *IMA Journal of Numerical Analysis*, Vol. 1, pp. 267-283.

Dijkstra, E.W., 1962, *A Primer of ALGOL 60 Programming*, A.P.I.C. Studies in Data Processing No. 2, Academic Press, London.

Dittus, F.W. and Boelter, L.M.K., 1930, *University of California Publications on Engineering*, Vol. 2, No. 13, pp. 443-461, Berkeley.

Eckels, P.W. and Rabas, T.J., 1987, Dehumidification: on the correlation of wet and dry transport processes in plate finned-tube heat exchangers, *Trans. ASME, J. Heat Transfer*, Vol. 109, pp. 575-582.

Ellis, M.A. and Stroustrup, B., 1990, *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, Mass.

Fanger, P.O., 1972, *Thermal Comfort*, McGraw-Hill, New York.

Fanger, P.O., 1988a, Hidden olfs in sick buildings, *ASHRAE Journal*, Vol. 30, No. 11, pp. 40-43.

Fanger, P.O., 1988b, Introduction of the olf and the decipol units to quantify air pollution perceived by humans indoors and outdoors, *Energy and Buildings*, Vol. 12, pp. 1-6.

Fanger, P.O., Lauridsen, J., Bluyssen, B. and Clausen, G., 1988, Air pollution sources in offices and assembly halls, quantified by the olf unit, *Energy and Buildings*, Vol. 12, pp. 1-19.

Fisk, D.J., 1981, *Thermal Control of Buildings*, Applied Science Publishers, London

Fobelets, A.P.R. and Gagge, A.P., 1988, Rationalization of the effective temperature ET* as a measure of enthalpy of the human indoor environment, *ASHARE Trans.*, Vol. 94, Part 1, pp. 12-31.

Fukui, S. and Sakamoto, M., 1968, Some experimental results on heat transfer characteristic of air cooled heat exchangers for air conditioning devices, *Bulletin of JSME*, Vol. 11, pp. 303-311.

Gagge, A.P., Stolwijk, J.A.J. and Nishi, Y., 1971, An effective temperature scale based on a simple model of human physiological regulatory response, *ASHRAE Trans.*, Vol. 77, Part 1, pp. 247-262.

Gardner, K.A., 1945, Efficiency of extended surface, *ASME Trans.*, Vol. 67, pp. 621-631.

Gay, D.M., 1983, Subroutines for unconstrained minimization using a model/trust-region approach, *Collected Algorithms from ACM*, Algorithm 611.

Gelperin, A., 1973, Humidification and upper respiratory infection incidence, *Heating, Piping and Air Conditioning*, Vol. 45, No. 3, p. 77.

Gilbert, G.P., 1987, Flow Through a Model Fin and Tube Heat Exchanger and its Influence on Mass and Heat Transfer, *M.Eng.Sc. Thesis*, University of Adelaide.

Gill, P.E., Murray, W. and Wright, M.H., 1981, *Practical Optimization*, Academic Press, New York.

Glicksman, L.R. and Hunt, A.W., 1972, Numerical simulation of dropwise condensation, *Int. J. Heat Mass Transfer*, Vol. 15, pp. 2251-2269.

Gnielinski, V., 1976, New equations for heat and mass transfer in turbulent pipe and channel flow, *Int. Chem. Engng.*, Vol. 16, pp. 359-368.

Gosney, W.B., 1982, *Principles of Refrigeration*, Cambridge University Press, Cambridge.

Graham, C. and Griffith, P., 1973, Drop size distributions and heat transfer in dropwise condensation, *Int. J. Heat Mass Transfer*, Vol. 16, pp. 337-346.

Green, G.H., 1979, The effect of indoor relative humidity on colds, *ASHRAE Trans.*, Vol. 85, Part 1.

Green, G.H., 1982, The positive and negative effects of building humidification, *ASHRAE Trans.*, Vol. 88, Part 1, p. 1049.

Guillory, J.L. and McQuiston, F.C., 1973, An experimental investigation of air dehumidification in a parallel plate exchanger, *ASHRAE Trans.*, Vol. 79, Part 2, pp. 146-151.

Gupta, V.K., Int-Hout, D., Roberts, M.M., Wessel, D.J., Brickman, H. and Waeldner, W., 1987, A forum on variable air volume, *ASHRAE Journal*, Vol. 29, No. 8, pp. 22-31.

Hallion, J.G., 1988, Nature of condensation on dehumidification coils, *Honours Thesis*, Dept. Mech. Eng., University of Adelaide.

Haltiner, G.J., and Martin, F.L., 1957, *Dynamical and Physical Meteorology*, McGraw-Hill, New York.

Hamilton, J.F. and Miller, J.L., 1990, A simulation program for modeling an air-conditioning system, *ASHRAE Transactions*, Vol. 96, Part 1, pp. 213-221.

Hansen, T.L., 1990, *The C++ Answer Book*, Addison-Wesley, Reading, Mass.

Hartnett, J.P. and Eckert, E.R.G., 1957, Mass-transfer cooling in a laminar boundary layer with constant fluid properties, *Trans. ASME*, Vol. 79, pp. 247-254.

Hayes, J.G. and Halliday, J., 1974, The least-squares fitting of cubic spline surfaces to general data sets, *J. Inst. Maths. Applics.*, Vol. 14, pp. 89-103.

Hill, J.M. and Jeter, S.M., 1991, A linear subgrid cooling and dehumidification coil model with emphasis on mass transfer, *ASHRAE Transactions*, Vol. 97, Part 2, pp. 118-128.

Hou, Y.C. and Martin, J.J., 1959, Physical and thermodynamic properties of trifluoromethane, *A.I.Ch.E. J.*, Vol. 5, pp. 125-129.

Hyland, R.W. and Wexler, A., 1983a, Formulations for the thermodynamic properties of the saturated phases of H_2O from 173.15K to 473.15K, *ASHRAE Transactions*, Vol. 89, Part 2A, pp. 500-519.

Hyland, R.W. and Wexler, A., 1983b, Formulations for the thermodynamic properties of dry air from 173.15K to 473.15K, and of saturated moist air from 173.15 to 372.15K, at pressures to 5 MPa, *ASHRAE Transactions*, Vol. 89, Part 2A, pp. 520-535.

ICI, 1991, KLEA 134a Preliminary Data Sheet.

Idem, S.A., Jacobi, A.M. and Goldschmidt, V.W., 1990, Heat transfer characterization of a finned-tube heat exchanger (with and without condensation), *Trans. ASME, J. Heat Transfer*, Vol. 112, pp. 64-70.

Incropera, F.P. and DeWitt, D.P., 1990, *Fundamentals of Heat and Mass Transfer*, 3rd Edition, John Wiley and Sons, New York.

Ito, H., 1959, Friction factors for turbulent flow in curved pipes, *Transactions of the ASME*, Series D, Vol. 81, pp. 123-134.

Ito, H., 1960, Pressure losses in smooth pipe bends, *Transactions of the ASME*, Series D, Vol. 82, pp. 131-143.

Jensen, K. and Wirth, N., 1978, *PASCAL: User Manual and Report*, 2nd Edition, Corrected Printing, Springer-Verlag, N.Y.

Jones, O.C., 1976, An improvement in the calculation of turbulent friction in rectangular ducts, *Trans. ASME, J. Fluids Engg.*, Vol. 48, pp. 173-181.

Jung, D. and Radermacher, R., 1991, Transport properties and surface tension of pure and mixed refrigerants, *ASHRAE Transactions*, Vol. 97, Part 1, pp. 90-99.

Kakac, S., Shah, R.K. and Aung, W., 1987, *Handbook of Single-Phase Convective Heat Transfer*, Wiley, New York.

Kays, W.M. and Crawford, M.E., 1993, *Convective Heat and Mass Transfer*, 3rd Edition, McGraw-Hill, New York.

Kays, W.M. and London, A.L., 1984, *Compact Heat Exchangers*, 3rd Edition, McGraw-Hill, New York.

Kernighan, B.W. and Ritchie, D.M., 1988, *The C Programming Language*, 2nd Edition, Prentice-Hall, Englewood Cliffs, New Jersey.

Kimura, K.-I., 1977, *Scientific Basis of Air Conditioning*, Applied Science Publishers, London.

Kinney, R.B. and Sparrow, E.M., 1970, Turbulent flow, heat transfer, and mass transfer in a tube with surface suction, *Trans. ASME, J. Heat Transfer*, Vol. 92, p. 117.

Kusuda, T., 1957, Graphical method simplifies determination of air coil wet surface temperature, *Refrigerating Engineering*, pp 41-45, 74-80.

Kusuda, T., 1965, Calculation of the Temperature of a Flat-Plate Surface under Adiabatic Conditions with Respect to the Lewis Relation, in *Humidity and Moisture, Volume I: Principles and Methods of Measuring Humidity in Gases*, (Ruskin, R.E., ed.), Reinhold Publ. Corp., New York.

Langley, A.J. and Whitbread, S.M., 1991, The tobacco industry and sick building syndrome: is untreated ventilation a substitute for workplace smoking practices, *Report to the South Australian Health Commission.*

LeFevre, E.J. and Rose, J.W., 1965, An experimental study of heat transfer by dropwise condensation, *Int. J. Heat Mass Transfer*, Vol. 8, p. 1117.

LeFevre, E.J. and Rose, J.W., 1966, A theory of heat transfer by dropwise condensation, *Proc. Third Int. Heat Transfer Conference*, Vol. 2, pp. 362-375, Am. Inst. Chem. Engrs., New York.

Linnhoff, B., 1993, Pinch analysis - a state-of-the-art overview, Trans. Institution of Chemical Engineers, Vol. 71, Part A, pp. 503-522.

Luxton, R.E. and Petrovic, V.M., 1995, Building classes and their influence on cooling load, *Proc. AIRAH 95 World Conference*, Melbourne, 1-3 May.

Luxton, R.E., Shaw, A. and Sekhar, S.C., 1989, Specification and achievement of part load conditions for air conditioning systems with minimum energy penalty, *Proc. CLIMA 2000*, Sarajevo, Yugoslavia, 27 August-1 September.

Luxton, R.E. and Shaw, A., 1991, Processes within a dehumidifier coil and their consequences in air-conditioning design, *Proc. 3rd ASME-JSME Thermal Engineering Joint Conference*, Reno, Nevada, 17-22 March.

McAdams, W.H., 1954, *Heat Transmission*, 3rd Edition, McGraw-Hill, New York.

McElgin, J. and Wiley, D.C., 1940, Calculation of coil surface areas for air cooling and dehumidification, *Heating, Piping and Air Conditioning*, March 1940, p. 195.

McKenzie, R.T.B., 1991, Aids to design - the bypass factor, AIRAH Journal, Vol. 45, No. 5, pp. 24-27.

McQuiston, F.C., 1975, Fin efficiency with combined heat and mass transfer, *ASHRAE Transactions*, Vol. 81, Part 1, pp. 350-355.

McQuiston, F.C., 1976, Heat, mass and momentum transfer in a parallel plate dehumidifying exchanger, *ASHRAE Transactions*, Vol. 82, Part 2, pp. 87-106.

McQuiston, F.C., 1978a, Heat, mass and momentum transfer data for five plate-fin-tube heat transfer surfaces, *ASHRAE Transactions*, Vol. 84, Part 1, pp. 266-293.

McQuiston, F.C., 1978b, Correlation of heat, mass and momentum transport coefficients for plate-fin-tube heat transfer surfaces with staggered tubes, *ASHRAE Transactions*, Vol. 84, Part 1, pp. 294-309.

McQuiston, F.C. and Parker, J.D., 1994, *Heating, Ventilating, and Air Conditioning: Analysis and Design*, 4th Edition, John Wiley and Sons, New York.

Marshallsay, P.G., Luxton, R.E. and Shaw, A., 1993, Ventilation air quantity, indoor air quality, and energy, *Proc. CLIMA 2000*, London, 1-3 November.

McLinden, M.O., Gallagher, J.S., Weber, L.A., Morrison, G., Ward, D., Goodwin, A.R.H., Moldover, M.R., Schmidt, J.W., Chae, H.B., Bruno, T.J., Ely, J.F. and Huber, M.L., 1989, Measaurement and formulation of the thermodynamic properties of refrigerants 134a (1,1,1,2-tetrafluoroethane) and 123 (1,1-dichloro-2,2,2-trifluoroethane), *ASHRAE Transactions*, Vol. 95, Part 2, pp. 263-283.

Meyer, B., 1988, *Object-oriented Software Construction*, Prentice-Hall, Englewood Cliffs, N.J.

Miller, D.S., 1978, *Internal Flow Systems*, British Hydrodynamics Research Association.

Moller, S.K. and Wooldridge, M.J., 1985, *User's Guide for the Computer Program BUNYIP: Building Energy Investigation Package (Version 2.0)*, CSIRO Division of Energy Technology, Technical Report TR6.

Moré, J.J. and Cosnard, M.Y., 1980, BRENTM, a Fortran subroutine for the numerical solution of systems of nonlinear equations, *Collected Algorithms from ACM*, Algorithm 554.

Nagaoka, K., Tanaka, Y., Kubota, H. and Makita, T., 1986, A new correlation for the viscosity of gaseous fluorocarbon refrigerants, *Int. J. Thermophysics*, Vol. 7, pp. 1023-1031.

Nusselt, W., 1916, Die Oberflachenkondensation des Wasserdampfes, *Z. Ver. Deut. Ing.*, Vol. 1, p. 53.

Phillips, T.W. and Murphy, K.P., 1970a, Liquid viscosity of halogenated refrigerants, *ASHRAE Transactions*, Vol. 76, pp. 146-156.

Phillips, T.W. and Murphy, K.P., 1970b, Liquid viscosity of halocarbons, *J. Chem. Eng. Data*, Vol. 15, pp. 304-307.

Piao, C., Sato, H. and Watanabe, K., 1991, Thermodynamic charts, tables, and equations for refrigerant HFC-134a, *ASHRAE Transactions*, Vol. 97, Part 2, pp. 268-284.

Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T., 1988, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge U.P., Cambridge.

Procell, C.J., 1990, Standard 62-189 (Letter), ASHRAE Journal, Vol. 32, No. 1, pp. 18-19.

O'Bara, J.T., 1967, Dropwise condensation of steam at atmospheric and above atmospheric pressures, *Chem. Eng. Sc.*, Vol. 22, pp. 1305-1314.

Osborne, W.C., 1977, *Fans*, 2nd Edition, Pergamon Press, London.

Rich, D.G., 1973, The effect of fin spacing on the heat transfer and friction performance of multi-row, smooth plate fin-and-tube heat exchangers, *ASHRAE Transactions*, Vol. 79, Part 2, pp. 137-145.

Rich, D.G., 1975, The effect of the number of tube rows on heat transfer performance of smooth plate fin-and-tube heat exchangers, *ASHRAE Transactions*, Vol. 81, Part 1, pp. 307-319.

Richardson Pacific Ltd., 1989, *Centrifugal Fans Series C Type CP/CY.*

Rose, J.W., 1972, Dropwise condensation of mercury, *Int. J. Heat Mass Transfer*, Vol. 15, pp. 1431-1434.

Rose, J.W., 1976, Further aspects of dropwise condensation theory, *Int. J. Heat Mass Transfer*, Vol. 19, pp. 1363-1370.

Rose, J.W., 1988, Some aspects of condensation heat transfer theory, *Int. Comm. Heat Mass Transfer*, Vol. 15, pp. 449-471.

Rose, J.W. and Glicksman, L.R., 1973, Dropwise condensation - the distribution of drop sizes, *Int. J. Heat Mass Transfer*, Vol. 16, pp. 411-425.

Saboya, F.E.M. and Sparrow, E.M., 1974, Local and average transfer coefficients for one-row plate fin and tube heat exchanger configurations, *Trans. ASME, J. Heat Transfer*, Vol. 96, pp. 265-272.

Saboya, F.E.M. and Sparrow, E.M., 1976a, Transfer characteristics of two-row plate fin and tube heat exchanger configurations, *Int. J. Heat Mass Transfer*, Vol. 19, pp. 41-49.

Saboya, F.E.M. and Sparrow, E.M., 1976b, Experiments on a three-row fin and tube heat exchanger, *Trans. ASME, J. Heat Transfer*, Vol. 98, pp. 520-522.

Schmidt, E., Schurig, W. and Sellschop, W., 1930, Versuche über die Kondensation von Wasserdampf an Film- und Tropfenform, *Tech. Mech. Thermo-Dynam.*, Berlin, Vol. 1, p. 53.

Schroeder-Lanz, A.-K., 1989, Investigation of the heat transfer coefficient of the condensate layer on dehumidifying coils, *Unpublished Report*, Department of Mechanical Engineering, The University of Adelaide.

Sekhar, S.C., 1990, Life Cycle Design of Dehumidifiers in Air Conditioning, *Ph.D. Thesis*, University of Adelaide.

Sekhar, S.C., Luxton, R.E. and Shaw, A., 1988, Some problems in the rating of cooling coils - ARI Standard 410 revisited, *Proc. International Congress of Commissioners of the International Institute of Refrigeration*, Brisbane, September.

Sekhar, S.C., Luxton, R.E. and Shaw, A., 1989, Design methodology for cost-effective air conditioning in humid climates, *Proc. Far East Conference on Air Conditioning in Hot Climates*, Kuala Lumpur, Malaysia, 25-28 October.

Sekhar, S.C., Shaw, A., Luxton, R.E., Browne, E.D. and DeIeso, S., 1991a, A control system upgrade for the closed cycle thermal environmental wind tunnel/coil test facility, *Proc. AIRAH Annual Conference*, Melbourne, Australia, 21-24 April.

Sekhar, S.C., Shaw, A. and Luxton, R.E., 1991b, A computer-based analytical method of data reduction for wetted surface dehumidifiers, *Proc. 4th International Symposium on Transport Phenomena in Heat and Mass Transfer*, Sydney, Australia, 14-19 July.

Sieder, E.N. and Tate, G.E., 1936, Ind. Eng. Chem., Vol. 28, pp. 1429-1436.

Sham Sani, 1979, *Aspects of Air Pollution Climatology in a Tropical City: A Case Study of Kuala Lumpur - Petaling Jaya*, Universiti Kebangsaan Malaysia Press.

Sham Sani, 1980, *The Climate of Kuala Lumpur - Petaling Jaya Area Malaysia: A Study of the Impact of Urbanization on Local Climate within the Humid Tropics*, Universiti Kebangsaan Malaysia Press.

Shaw, A., 1979, A unified approach to the design of climate simulators, *Ph.D. Thesis*, The University of Adelaide.

Shaw, A., 1982a, Airstream velocity across dehumidifiers, *Proc. 7th International Heat Transfer Conference*, Munich, FDR, Vol. 6, August 1982.

Shaw, A., 1982b, Exploration of air velocity across air-conditioning dehumidifiers - an energy conservation project, *ASHRAE Transactions*, Vol. 88, Part 2.

Shaw, A., 1985, Latest findings on airstream velocity effects in heat and mass transfer through dehumidifier coils, *Proc. 2nd Australasian Conference on Heat and Mass Transfer*, Melbourne, Australia, 13-15 May.

Shaw, A. and Luxton, R.E., 1988a, A comprehensive method of improving part-load air-conditioning performance, *ASHRAE Transactions*, Vol. 94, Part 1.

Shaw, A. and Luxton, R.E., 1988b, An overview of low face velocity air conditioning, *AIRAH Journal*, Vol. 42, No. 7, pp. 37-43.

Shaw, A. and Luxton, R.E., 1990, Combined heat and mass transfer in a dehumidifier - the basis of low energy tailored for human comfort, *Proc. 9th International Heat Transfer Conference*, Jerusalem, Israel, 19-24 August.

Shaw, A., Luxton, R.E. and Marshallsay, P.G., 1992, Life cycle design methodology for low energy air conditioning, *Proc. 1st European Thermal-Sciences and 3rd UK National Heat Transfer Conference*, Birmingham, 16-18 September.

Shaw, A., Luxton, R.E. and Marshallsay, P.G., 1993, Integration of dehumidification into life-cycle system design, *Proc. ASHRAE Conference on Building Design, Technology and Occupant Well-being in Temperate Climates*, Brussels, 17-19 February.

Sherman, F.S., 1990, *Viscous Flow*, McGraw-Hill, New York.

Smith, J.M. and Van Ness, H.C., 1975, *Introduction to Chemical Engineering Thermodynamics*, 3rd Edition, McGraw-Hill, New York.

Spieksma, F.Th.M., 1970, Biological aspects of the house dust mite (*Dermatophagoides pteronyssinus*) in relation to house dust atopy, *Clinical and Experimental Immunology*, Vol. 6, pp. 61-70.

Spieksma, F.Th.M., 1990, Mite biology, *Clinical Reviews in Allergy*, Vol. 8, pp. 31-49.

Spieksma, F.Th.M., 1991, Domestic mites: their role in respiratory allergy, *Clinical and Experimental Allergy*, Vol. 21, pp. 655-660.

Spieksma, F.Th.M., 1993, Aerobiology of common environmental allergens: sizes of allergen carrying particles, *Asian Pacific Journal of Allergy and Immunology*, Vol. 11, pp. 93-94.

Spieksma, F.Th.M., Zuidema, P. and Leupen, M.J., 1971, High altitudes and house-dust mites, *British Medical Journal*, Vol. 1, pp. 82-84.

Stewart, R.B., Jacobsen, R.T. and Becker, J.H., 1983, Formulations for thermodynamic properties of moist air at low pressures as used for construction of new ASHRAE SI unit psychrometric charts, *ASHRAE Transactions*, Vol. 89, Part 2A, pp. 536-548.

Stiehl, L.I. and Thodos, G., 1962, The viscosity of polar gases at normal pressures, *A.I.Ch.E. J.*, Vol. 8, pp. 229-232.

Stiehl, L.I. and Thodos, G., 1964a, The thermal conductivity of nonpolar substances in the dense gaseous and liquid regions, *A.I.Ch.E. J.*, Vol. 10, pp. 26-29.

Stiehl, L.I. and Thodos, G., 1964b, The viscosity of polar substances in the dense gaseous and liquid regions, *A.I.Ch.E. J.*, Vol. 10, pp. 275-277.

Stoecker, W.F., 1989, *Design of Thermal Systems*, 3rd Edition, McGraw-Hill, New York.

Stroustrup, B., 1991, *The C++ Programming Language*, 2nd Edition, Addison-Wesley, Reading, Mass.

Swift, A. and Lindfield, G.A., 1978, Comparison of a continuation method with Brent's method for the numerical solution of a single nonlinear equation, *Computer J.*, Vol. 21, pp. 359-362.

Tai, G.S.F., 1991, Condensation on air conditioning dehumidification coils, *Honours Thesis*, Dept. Mech. Eng., University of Adelaide.

Tamblyn, R.T., 1983, Beating the blahs for VAV, *ASHRAE Journal*, Vol. 24, No. 9, pp. 42-45.

Threlkeld, J.L., 1970, *Thermal Environmental Engineering*, 2nd Edition, Prentice-Hall, Englewood Cliffs.

Tree, D.R. and Helmer, W.A., 1976, Experimental heat and mass transfer data for condensing flow in a parallel plate heat exchanger, *ASHRAE Transactions*, Vol. 82, Part 1, pp. 289-299.

Tremblay, J.-P. and Sorensen, P.G., 1984, *An Introduction to Data Structures with Applications*, 2nd Edition, McGraw-Hill, New York.

Tsal, R.J. and Adler, M.S., 1987, Evaluation of numerical methods for ductwork and pipeline optimization, *ASHRAE Transactions*, Vol. 93, Part 1, pp. 17-34.

Tsal, R.J., Behls, H.F. and Mangel, R., 1988a, T-method duct design. Part I: Optimization theory, *ASHRAE Transactions*, Vol. 94, Part 2, pp. 90-111.

Tsal, R.J., Behls, H.F. and Mangel, R., 1988b, T-method duct design. Part II: Calculation procedure and economic analysis, *ASHRAE Transactions*, Vol. 94, Part 2.

Tsal, R.J., Behls, H.F. and Mangel, R., 1990, T-method duct design. Part III: Simulation, *ASHRAE Transactions*, Vol. 96, Part 2, pp. 3-31.

Ungbhakorn, V., 1989, Air-conditioning expert system for buildings in Thailand, *Proc. Far East Conference on Air Conditioning in Hot Climates*, Kuala Lumpur, Malaysia, 25-28 October.

U.S. Department of Energy, 1981, *DOE-2 Reference Manual (Version L1A)*, Los Alamos National Laboratory Report LA-7689-M.

Van Aken, G.J., 1990, *Numerical methods on the properties of gases, vapours and liquids*, Australian Construction Services, Mechanical Engineering Report No. 39.

Van Aken, G.J., 1993, Transient Modelling of Finned Tube Heat Exchangers, *M.Eng.Sc. Thesis*, University of Adelaide.

Van Wylen, G.J. and Sonntag, R.E., 1985, *Fundamentals of Classical Thermodynamics*, 3rd Edition, John Wiley and Sons, New York.

Von Thun, G. and Witte, M.J., 1991, Applying building simulation tools to optimize system sizing and operation strategies, *ASHRAE Transactions*, Vol. 97, Part 1, pp. 697-703.

Ward-Smith, A.J., 1980, *Internal Fluid Flow: The Fluid Dynamics of Flow in Pipes and Ducts*, Clarendon Press, Oxford.

Ware, C.D. and Hacha, T.H., 1960, Heat transfer from humid air to fin and tube extended surface cooling coils, *ASME Paper No. 60-HT-17.*

Webb, R.L., 1990, Standard nomenclature for mass transfer processes, *International Communications in Heat and Mass Transfer*, Vol. 17, pp. 529-535.

White, C.M., 1929, Streamline flow through curved pipes, *Proc. Roy. Soc.*, Vol. A123, pp. 645-663.

Wilson, D.P. and Basu, R.S., 1988, Thermodynamic properties of a new stratospherically safe working fluid - refrigerant r134a, *ASHRAE Transactions*, Vol. 94, Part 2, pp. 2095-2118.

Worbs, P.E., 1984, A friendly microprocessor approach to the determination of the thermodynamic properties of moist air, *ASHRAE Transactions*, Vol. 90, Part 1A, pp. 46-57.

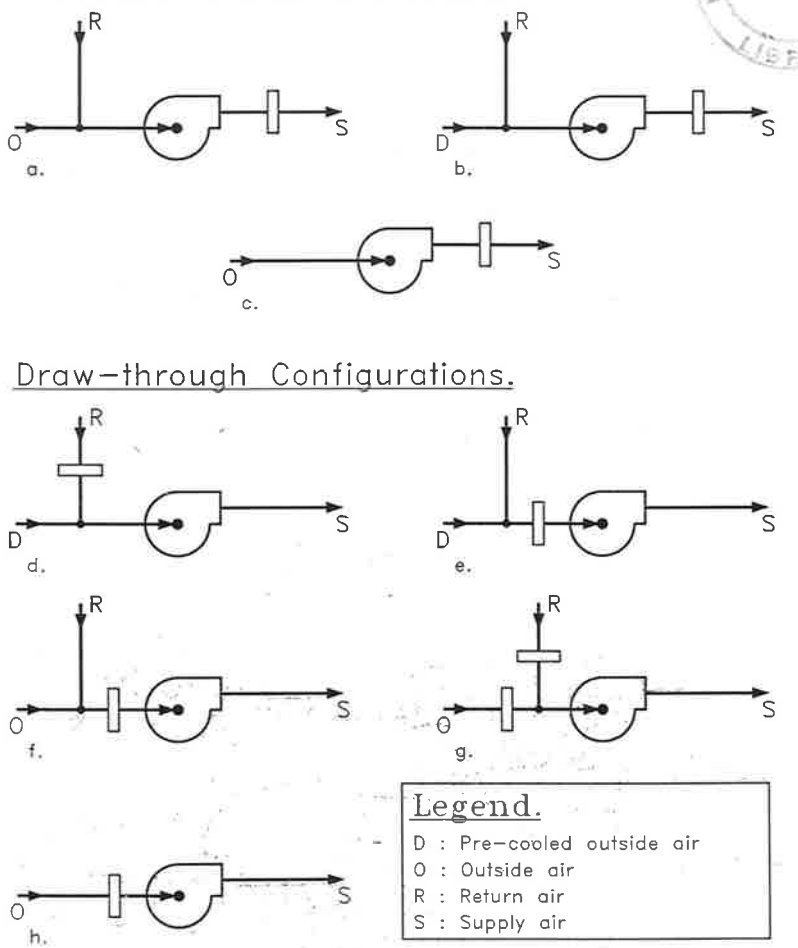
Yata, J., Minamiya, T. and Tanaka, S., 1984, Measurement of thermal conductivity of liquid fluorocarbons, *Int. J. Thermophysics*, Vol. 5, pp. 209-218.

Yoshida, H. and Terai, T., 1992, Modeling of weather data by time series analysis for air-conditioning load calculations, *ASHRAE Transactions*, Vol. 98, Part 1, pp. 328-345.

Young, T.L. and Van Woert, M.L., 1989, *PLOT88 Software Library Reference Manual*, 3rd Edition, Plotworks Inc.

Yourdon, E., 1989, *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs.

Cooling Coil Configurations for a Distributed Air Handling Unit.



The above is a reproduction of Figure 6.5.