Modelling the Steady-State Thermal Performance of Air Conditioning Systems - the ZEBRA Approach.

P.G. MARSHALLSAY and R.E. LUXTON.
Department of Mechanical Engineering, The University of Adelaide, South Australia 5005.

1. Introduction.

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 above factors, 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 more widely-used load calculation packages predicts coil-on and coil-off conditions required to achieve specified zone conditions. This information is commonly used to select coils of sufficient capacity for the task at hand. It is implicitly, and naively, assumed that the selected coils will also provide the desired degree of dehumidification. 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 which has been called “the moisture staircase” (Luxton and Shaw, 1991). For a single zone system using a simple coil, the 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 benefits of automating the design process have received some recognition in recent years, and a number of computer-based air conditioning design packages are currently being developed (Madjidi and Bauer, 1994). The ZEBRA software package has been developed against a background of more than twenty years of research into the fundamentals of air conditioning system operation and design methodology, and has been conceived as a tool which will facilitate the activities of system selection and control system design and optimization by adopting a holistic approach to the entire design specification and design process. In essence, the software package is based on a system simulation engine which has been established on a firm physical basis, augmented where appropriate with experimental design data. The software has been structured strictly according to object-oriented design principles, providing maximum flexibility in terms of the system configurations and components which can be modelled now and in the future. The software system seeks to integrate various aspects of current design practice, which would otherwise be performed either manually or by using a disparate collection of computational design aids. As such, it provides a basis for further automation of the design process through proposed linkages to other software components, including expert systems, databases, load calculation packages and CAD/CAM software.

The ZEBRA package has been under development for some years, and has been used in a number of consulting projects and design studies. The ability to model efficiently the performance of an air conditioning system over its entire operating range has provided the basis for the development of a systematic and rational air conditioning design methodology for life cycle operation. In addition, the process of designing and implementing the computational model has led to the identification of shortcomings in our understanding of certain aspects of system performance, and of ambiguities in the available experimental data. These are being addressed and studies are proposed to compare the predictions of the model with the operating characteristics of installed plant.

2. Overall Structure.

A number of HVAC simulation tools have been developed over the last twenty years, for the most part as components of comprehensive building energy analysis codes, such as ESP (Clarke, 1985), BLAST (BLAST Support Office, 1991) and BUNYIP (Moller and Wooldridge, 1985). While each is appropriate for the purpose for which they were designed, it is the authors' contention that these software packages do not adequately address the needs of the audience identified above. There are three major reasons for this:

1. It has been the authors' experience with some such codes that the degree of rigour applied to modelling plant components does not match that applied to modelling the building envelope, and while the plant model may be adequate to estimate overall energy consumption, it seldom provides other than a poor estimate of plant psychrometric performance.

2. Even if we assume that the plant simulation has been based upon a physically correct model, the close coupling between the building and plant simulation components of the model requires that both components be simulated simultaneously. This limits the

user's flexibility in specifying test load sequences, and renders objective comparison of plant design solutions difficult and tedious.

3. Many of the older codes limit the plant configuration options available to the user. While the scope of many of the newer codes such as SPARK (Buhl et al., 1993), and EKS (Tang and Clarke, 1993) do not suffer from this shortcoming, as *dynamic* models they are still inherently tied to a building simulation model.

In contrast to the above, ZEBRA models the *steady-state* performance of a HVAC system[1], and thus effects a decoupling between the plant model and the building envelope model. This decoupling is justified on the grounds that under normal operating conditions the time scale describing the plant dynamics will be considerably shorter than that associated with the building envelope[2]. Having effected this decoupling, it becomes possible to reduce the load space over which a system configuration is to be evaluated to a set of critical loads, such as may be determined by the design locus (Koptchev and Luxton, 1996), rather than requiring evaluation over a comprehensive time sequence.

ZEBRA is also distinguished from most other extant codes in that it has been structured from the point of view of the HVAC system designer rather than that of the building energy analyst. Accordingly, considerable emphasis has been placed upon modelling the performance of the air-side of the system, and particularly of the all-important but frequently neglected dehumidifier coil[3]. The model *in its present form* is not all-inclusive. For instance there is no chiller model and the choice of coolants is restricted to chilled water. The software is however structured in such a way as to facilitate extension of the range of applicability of the model, and work is in progress to develop and integrate a number of new features.

The projected structure of the ZEBRA package in its entirety is as shown in figure 1. At the heart of the package is the Zebra Kernel, which is in essence the engine of the simulation system. It contains the data structures and associated algorithms required to represent the various plant components and to simulate plant response to changes in operating conditions. The primary channel for user interaction with the Zebra Kernel is by means of a closely-coupled graphical user interface which provides access to a range of system services, the most important of these being:

---

1 The ASHRAE Secondary Systems Toolkit also models the steady-state performance of systems components such as fans and heat exchangers. However, this toolkit is configured as a discrete collection of FORTRAN subroutines modelling the performance characteristics of specific components; the user is left with the task of incorporating the components into a *system* model.

2 This contention is borne out by the authors' laboratory and professional experience.

3 The model used has been derived upon the basis of a rigorous analysis of the processes occurring within the cooling coil as determined by comprehensive laboratory testing.

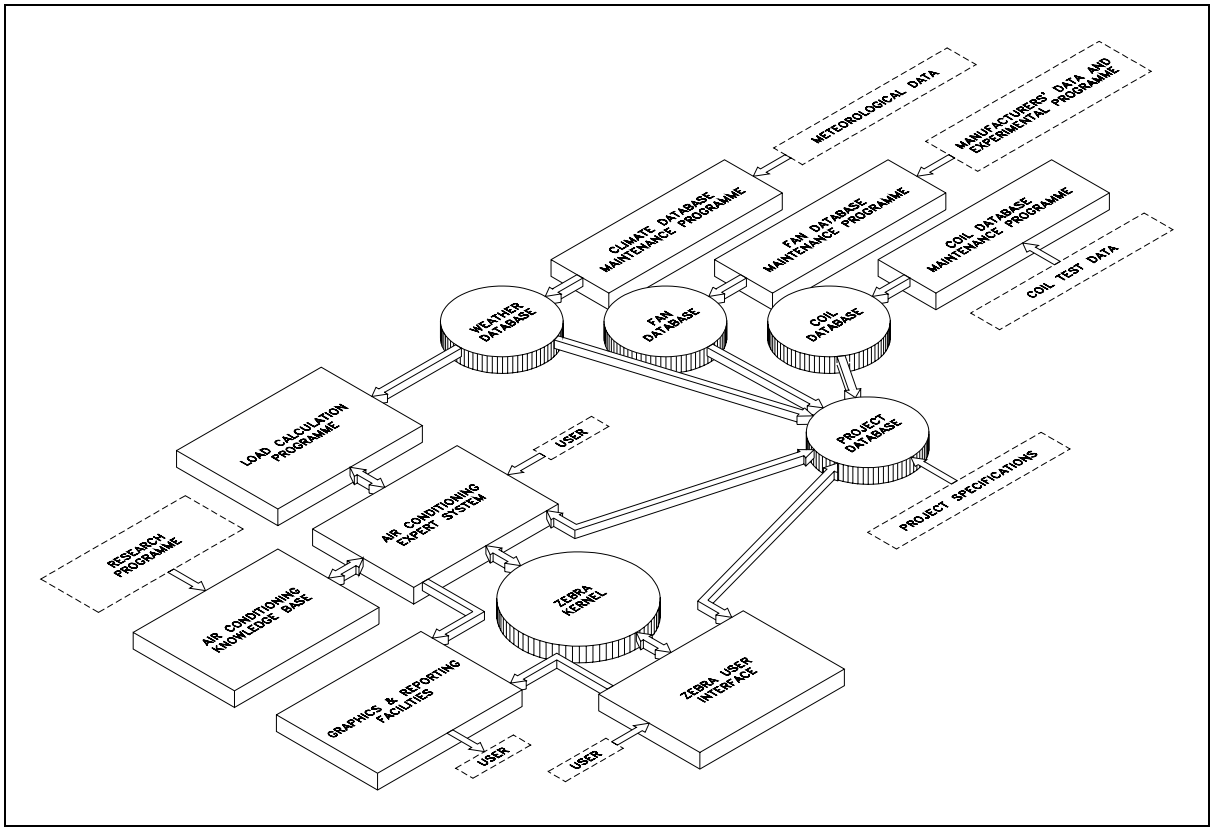

Figure 1. Architecture of the ZEBRA software package.

i. Specification and editing of the model of the HVAC system and its associated control strategies. The system model may be archived to disc between sessions.
ii. Specification of zone loads and ambient conditions.
iii. Initiation of a system simulation run using the currently specified system configuration and load conditions.
iv. Generation and printing of reports using a report generator and printing facilities hosted within the Zebra Kernel itself.

The capabilities listed above provide a basic level of functionality, which is augmented by two further complementary features:

i. The ability to run a system simulation sequence using a pre-programmed sequence of zone loads and ambient conditions as specified in a *metafile*. The sequence might for instance represent the load conditions pertaining at regular intervals over a period of operation, or might alternatively represent a standard test sequence devised by the user to compare candidate design solutions for a particular problem.
ii. The ability to store selected data items from consecutive runs to a *log file* for further processing, which might include off-line report generation, graphical display, or conversion to formats suitable for input to spreadsheet and statistical analysis packages.

The Zebra Kernel is supported by a number of coordinating components, which either exist at the present time or are projected for future implementation. These include:

i. Databases. The Zebra system interacts with a number of databases, which essentially fall into two categories:

   a. The *project* database. This is currently restricted in scope, but in the longer term will provide an indexed repository for a wide range of data relating to each project.

   b. *Global* databases contain information which is available to all projects based on need. These include a set of databases containing the characteristics of various equipment items, such as fans, filters and coils, and a climate database which is currently being developed as a collaborative venture between the University of Adelaide and the Bureau of Meteorology.

ii. Expert System. The Zebra Kernel in itself is a system simulation tool which may be used in conjunction with an expert system to automate much of the HVAC system design process. The eventual aim is to provide a tool which will guide the user through the initial choice of a system configuration to the detailed selection of equipment components and control strategies to produce a near-optimal design solution. This is necessarily a long-term aim since we do not believe the requisite expertise currently exists; it will be developed by systematic exploration of design spaces using tools such as Zebra. The shorter term aim is to automate specific facets of the design process on an incremental basis.

iii. Load Calculation Programme. Interaction with a load calculation programme is desirable both as a convenience when dealing with time sequences of load conditions, and more importantly, to investigate system performance when zone dry-bulb temperature conditions vary, either by design or as a result of systems with inadequate capacity being specified. Few if any of the load calculation codes in common use provide an interactive capability. The possibility of interfacing with one or more available codes is under investigation.

The discussion which follows will focus on the structure of the system model used by the Zebra Kernel.

3. Design Philosophy.

The Zebra Kernel is a complex software system designed to serve a diverse body of users over a comparatively long life cycle. With this in mind, factors which have been of major importance in determining the philosophy underpinning the design of the package are:

i. User access to the model. In particular, it must be borne in mind that the design process in most cases proceeds from the general to the specific. Thus, it is appropriate that the designer should be able as far as is appropriate to specify generic equipment items having default operating behaviour during the early stages of the design cycle. The specificity of the model can subsequently be refined as concrete design decisions are made.

ii. Maintainability of the programme structure and software components.

iii. Extensibility of the software system to incorporate additional features, and of the HVAC system model to represent additional configurations and equipment items.

The package has been designed to meet the above criteria using object-oriented design techniques (Booch, 1994; Jacobson et al., 1992) and is programmed in the C++ language

(Stroustrup, 1991) which provides support for object-oriented programming. In this approach, components of the model are regarded as *objects*, each of which is an instance of a specific *class*. A class contains data members which specify the *state* of objects of that class, and member functions which specify object *behaviour*. Thus, the components in a model of a refrigeration cycle might be represented by a *base* class *Component* which, in its simplest form specifies the state of the corresponding objects by data members representing:

The state of the working fluid at entry to the component;
The state of the working fluid leaving the component;
The mass flow rate of the working fluid through the component;
The working fluid itself (an object of another class)

and specifies the behaviour of the objects by an *abstract* function *Process* which takes the object through an (as yet undefined) thermodynamic process. The inheritance mechanism provides a means to augment and refine the behaviour of derived classes, while retaining access to the attributes of the *base* class. Thus, to model an ideal vapour refrigeration cycle, one needs to derive four classes from base class *Component*, each of which redefines member function *Process* to model the appropriate thermodynamic process:

Class *Compressor*: function *Process* performs isentropic compression;
Class *Condenser*: function *Process* performs isobaric condensation;
Class *ExpansionDevice*: function *Process* performs isentropic expansion;
Class *Evaporator*: function *Process* performs isobaric evaporation.

The process may be further refined by deriving an additional layer of classes from the above. Thus, classes *ReciprocatingCompressor*, *ScrewCompressor* and *CentrifugalCompressor* might be derived from class *Compressor*, each redefining function *Process* to account for departures from ideal behaviour, and providing additional data members to describe the characteristics of a specific compressor type. Structuring software entities in a hierarchy as described offers benefits for both the user and the programmer. From the point of view of the user, it provides an opportunity to start from a set of generic components, and to refine the design incrementally while concentrating on one component at a time. From the programmer's point of view, it provides a structure which is readily extended as the need arises, and which facilitates re-use of existing code; member function *Process* of class *Compressor* remains accessible to classes higher in the hierarchy, and may be invoked to provide an initial estimate in an iterative procedure to account for departures from ideal behaviour.

4. Programme Structure.

Within an object-oriented framework, a complex system may be modelled by selecting a suitable set of coordinating classes, and defining the operations whereby they interact. The core class categories in the Zebra model are shown in figure 2, and essentially perform the following functions:

i. An object of class *HVACProject* provides the means by which all other components of the package access the components of the HVAC system model. There is one and only one object of this class for each invocation of ZEBRA. This object maintains indexes to the zones and systems in the model, and processes all requests to edit and query the state of the components, and to run simulations.

ii. Each zone within the project is represented by an object of class *Zone*, which specifies the zone loads, and the status of the thermostat and humidistat serving the zone (Enabled/Disabled; On/Off; set point). Zones may or may not be conditioned.

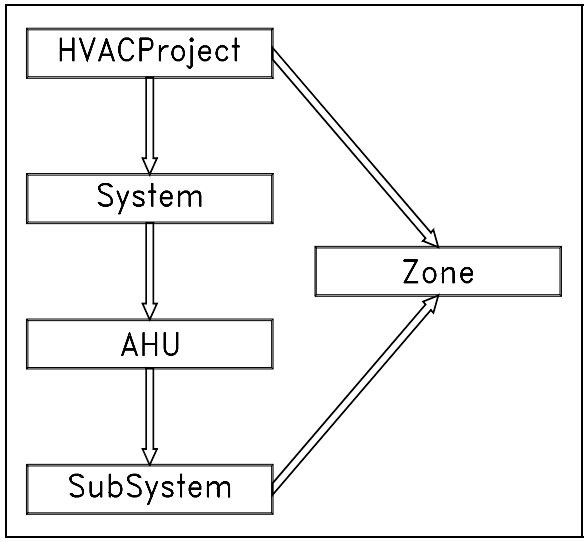

Figure 2. Core class categories required to imlement the air conditioning system model.

iii. Objects of class *System*, or more properly one of its subclasses, each of which maintains a one-to-one relationship with an object of class *AHU*, which provides a model representation of a *primary* air-handling unit, each of which serves one or more zones or, in the case of central outdoor air treatment plants, one or more *secondary* units (themselves objects of class *AHU*). Subclasses of class *System* provide a common interface, specifying a range of operations including initiation of a simulation run, generation of a report describing current system state, and creation and editing of a system model of the appropriate type. Specialisation of the base class to a specific system type is determined by the implementation of the derived class. Similarly, class *AHU* provides an abstract representation of an air handling unit which may be adapted to provide a concrete representation of a specific type by appropriate specification of equipment subsystems and control strategies. This aspect is obviously of central importance to our modelling strategy, and will be taken up again at a later stage.

The above classes thus provide a generalised framework which will accommodate a wide range of equipment types and system configurations.

5. Algorithms for System Simulation.

Luxton and Shaw (1991) focussed attention on the often overlooked "*thermodynamic requirement 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 in 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 a controlled 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 the *moisture staircase* phenomenon mentioned in the Introduction. This process has been graphically illustrated by Sekhar (1990), and by Luxton and Shaw (1991). It can be demonstrated for a single-zone system of conventional configuration 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 basic scheme thus defined is capable of modelling a wide range of system configurations, with appropriate perturbations. Some of the considerations in programming the algorithm in its more basic forms will be considered below.

5.1. The Outer Loop.

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. 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 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

$$\tilde{W}_a < W < \tilde{W}_b \quad . \tag{1}$$

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

$$f(\tilde{W}) = \tilde{W} - \tilde{W}' = 0 \quad . \tag{2}$$

Provided we can write a function which will take $\tilde{W}$ as input, and return $f(\tilde{W})$, equation (2) can be solved efficiently using a root-finding procedure. The method generalises to the multizone case if we observe that at equilibrium equation (2) must be satisfied at every point in the system. In the more general case it is convenient to take $\tilde{W}$ to be the supply-air humidity ratio.

5.2. The Inner Loop.

The purpose of the inner loop is to calculate the objective function $f(\tilde{W})$, as defined in equation (2), for some estimate of the supply-air humidity ratio. In other words, we aim to take a parcel of air at a specified supply-air condition (t, $\tilde{W}$) through one circuit of the air conditioning process, and find the condition of the parcel on completion of the circuit. For a multizone all-air system of conventional configuration, the procedure is, in outline, as follows:

i. Assuming that each zone will operate at its thermostat set-point, the humidity ratio corresponding to the supply-air humidity ratio is calculated. Two points need to be taken into consideration during this process:

   a. The supply-air duct temperature gain for each zone needs to be accounted for.

   b. For a VAV system, the supply-air temperature is controlled, and is thus known; for a CAV system it needs to be calculated at this stage. To do this, we calculate the supply-air temperature $t_{SA,i}$ required by each zone in the absence of all other zones. Then, the supply-air temperature for the system,

$$t_{SA} = \min_i t_{SA,i} \quad . \tag{3}$$

   For all zones other than those for which $t_{SA} = t_{SA}$, reheat must be supplied to achieve the desired set-point temperature.

ii. The return-air condition is found by incrementing the temperature of the return-air stream from each zone by its appropriate return-air duct temperature rise, and performing psychrometric mixing of all the return-air streams.

iii. The coil-on condition is determined by mixing the return-air and outdoor-air streams psychrometrically in the ratio of their respective mass flow rates.

iv. Let t_{target} be the target coil-off dry-bulb temperature, defined as

$$t_{target} = t_{SA} - \Delta t_{fan} \quad . \tag{4}$$

where Δt_{fan} is the fan temperature gain. Then, the coil chilled water flow rate is adjusted until the target coil-off temperature is achieved; the coil-off condition determines the revised supply-air humidity ratio ($\tilde{W}'$ in equation (2)).

5.3. Handling Humidity Constraints.

The procedure described above will provide a solution for the situation in which no 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. Having obtained a solution for the unconstrained problem, a check is carried out to determine whether any humidity constraints have been violated. An algorithm to solve the constrained problem follows if we observe:

i. The solution to the constrained problem will always require a greater flow rate of chilled water than the corresponding unconstrained problem.

ii. 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 ($\Delta W_i = 0$).

iii. There is one and only one supply-air humidity ratio which will exactly satisfy an active humidity constraint.

Having set the condition for some zone *i* in step (ii) above, the supply-air condition, and consequently the conditions for all other zones are established, and can be found for either VAV or CAV operation using an appropriate algorithm. The steps which follow are essentially those described for the inner loop of the unconstrained case above, except that now we adjust the chilled water flow rate to achieve a target coil-off humidity ratio, W_{target}. Then, the required reheat,

$$\Delta t_{reheat} = t_{SA} - \Delta t_{fan} - t_{coil-on} \quad . \tag{5}$$

5.4. Generalisation.

The preceding sections describe in outline a modelling procedure which will simulate the steady-state thermal performance of all-air systems of conventional design. The methodology may readily be extended to cover a wide range of system types. Among the additional system types for which solutions have been developed are the following:

i. Blow-through systems having multiple cooling coils, each independently controlled, but sharing a common return-air path.
ii. Systems based on the high-driving potential (HDP) principle (Shaw et al., 1993), in which the outdoor-air and return-air streams are separately treated through individual cooling coils.
iii. Induction systems.

The task of incorporating new system types will be facilitated, and reuse of existing and tested code will be maximised if close attention is paid to selecting a suitable set of classes, and distributing the desired functionality in a appropriate manner. As a general principle:

i. *Functionality should be placed in a class representing the component or subsystem which will perform the associated action*, and
ii. *Where a candidate class forms part of an inheritance hierarchy (as will generally be the case), a specific item of functionality should be placed as low in the hierarchy as is consistent with the structure of the hierarchy*. Distribution of functionality through an inheritance hierarchy will in fact normally be a major consideration in determining the architecture of the hierarchy.

These issues will be pursued further in the following section.

6. Modelling of System Components.

It will be obvious from the foregoing discussion, and from section 4, that the manner in which class *AHU* and its component classes have been structured, and functionality distributed among them, has been of central importance to the entire software design process. The structuring of these classes will be described in the following. As a preliminary step in this discussion, it is instructive to consider the range of coil configurations which are possible for a fan-powered air-handling unit. The configurations which are supported by Zebra (the range is considered to be exhaustive) are shown in figure 3. To model the desired range of configurations, it is convenient to consider an air-handling unit to be constructed from a set of subsystems:

i. The *outdoor-air subsystem*;

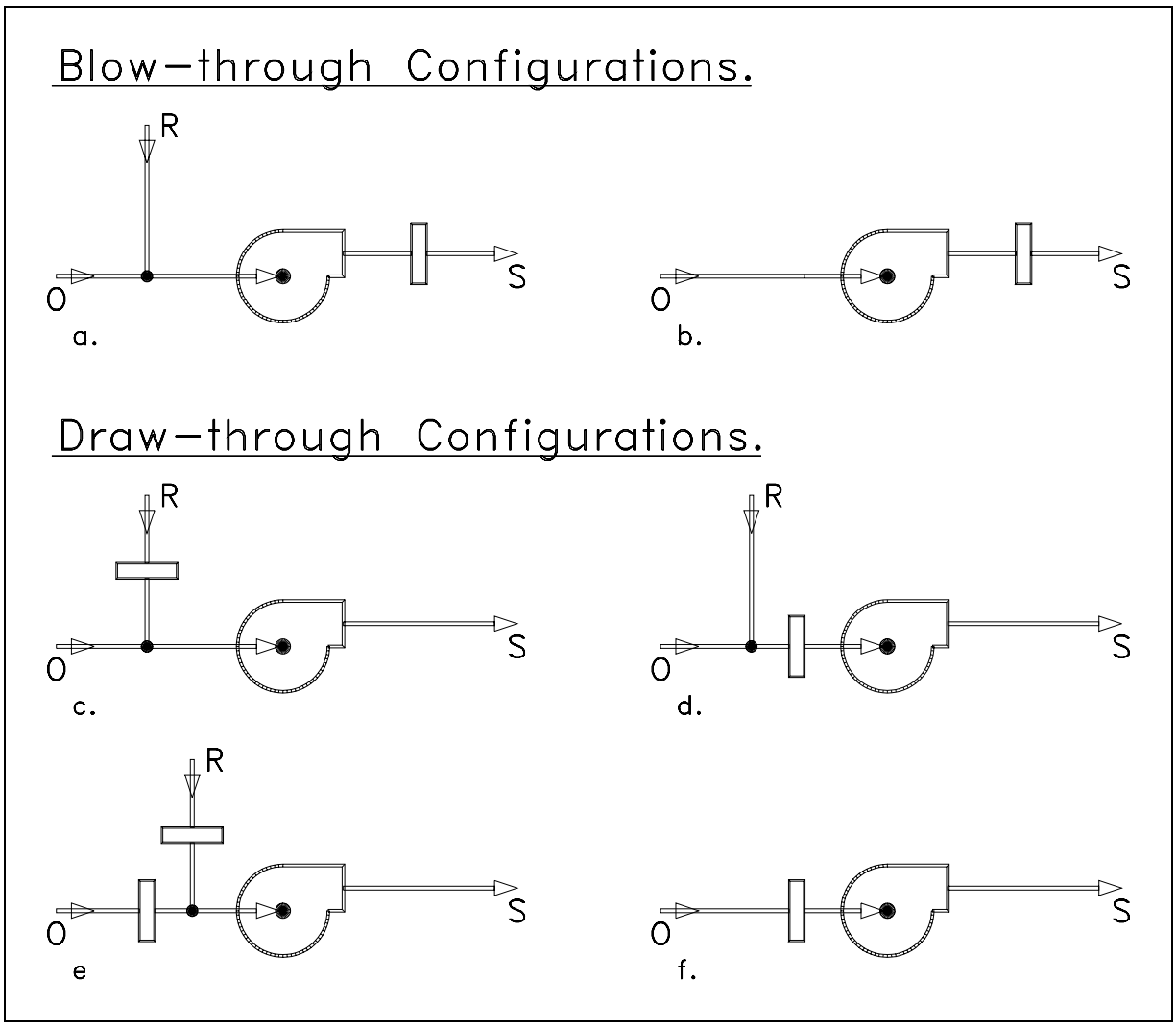

Figure 3. Available cooling coil configurations for an air handling unit. In the above; **O** signifies *outdoor air* (which may have been pre-treated); **R** signifies *return air*; and **S** signifies *supply air*.

ii. The *return-air subsystem*;
iii. The *supply-air subsystem*;
iv. The *coil subsystem*;
v. The *fan subsystem*.

Several points are apparent from figure 3:

i. Some system configurations do not use all of the above types of subsystem. In particular, ventilation only configurations ((b) and (f), and central pre-treatment units which share this general configuration) will not have a return-air subsystem.
ii. In most of the configurations shown, the coil subsystem (plural for all-air HDP units) is associated with either the outdoor-air subsystem, the return-air subsystem or the supply-air subsystem. Only for the conventional configuration (d) is it not possible to associate the coil subsystem with one of the other subsystems.

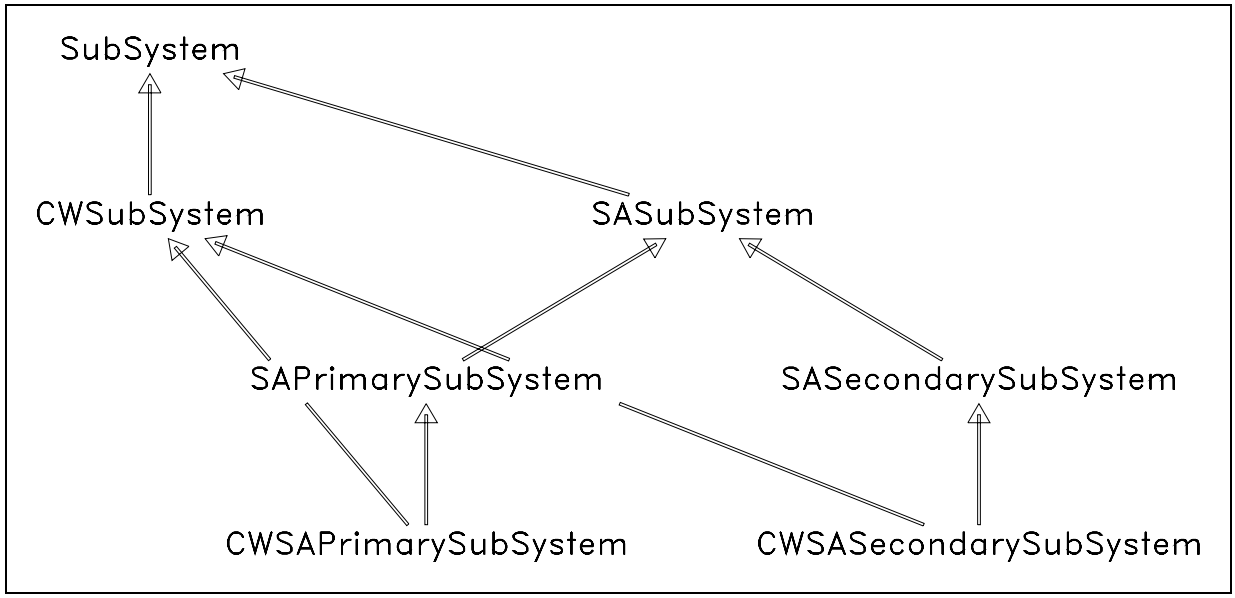

Figure 4. Directed Acyclic Graph showing the inheritance hierarchy for class *SubSystem* and its derived classes.

iii. For primary units, it is convenient for the structure representing the supply-air subsystem to maintain a list of the secondary units served by that subsystem; for secondary units, the supply-air subsystem should maintain a list of the zones served.

To provide the desired functionality, outdoor-air subsystems, return-air subsystems and supply-air subsystems are all members of an inheritance hierarchy which has class *SubSystem* as its common base class. The complete hierarchy is shown in figure 4, to which the following notes refer:

i. Class *SubSystem* provides a basic level of functionality. It provides member functions to access the air state at entry to and exit from the subsystem, the air pressure drop across the subsystem and the mass flow rate of air through the subsystem, together with (possibly null) pointers to a filter and a reheat device. The class representing reheat devices (class *ReheatDevice*) provides a member function to find the reheat necessary to provide a target exit dry-bulb temperature.

ii. Class *CWSubSystem* adds support for a chilled-water coil subsystem to the base structure. The coil subsystem is modelled by an object of class *CWCoilControlStructure*, of which more will be said later. The implementation of this class is fairly complex, and provides options for 2- and 4-pipe installations, with the possibility of shared or separate cooling and heating coils in the latter case, and the option of using a separate hot water coil for reheat.

iii. Class *SASubSystem* is a base class for supply-air subsystems, and adds a supply-air thermostat to the basic implementation of class *SubSystem*. Specialised versions of the class for primary and secondary units are provided by derived classes *SAPrimarySubSystem* and *SASecondarySubSystem* respectively, from which are derived classes *CWSAPrimarySubSystem* and *CWSASecondarySubSystem* for use in blow-through units. Note that these latter classes provide an example of *multiple inheritance*.

6.1. The Coil Subsystem.

In essence, the coil subsystem comprises two major components, a coil bank and a control valve, both of which are implemented as objects of an appropriate class. Class *CoilBank* (and *CWCoilBank*, its specialisation to chilled-water coils) provides a means of representing and simulating the performance of coils of arbitrary complexity using simple coil components as the basic building blocks, and of selectively deactivating or bypassing certain circuits in the coil. The class provides member functions to solve the following problems (full details of the algorithms used will be found in Marshallsay, 1996):

i. Sensible and latent cooling capacities and state of the working fluids leaving the coil bank, if the mass flow rates and conditions of the fluids at entry to the coil bank have been specified.
ii. Water pressure drop through the coil bank.
iii. Air pressure drop across the coil bank.

Incorporating the coil bank and control valve structures into class *CWCoilStructure* has enabled the implementation of member functions to solve two additional problems:

i. To find the valve setting (chilled water flow rate) to achieve a target coil-off temperature for the coil bank, and
ii. To find the valve setting to achieve a target coil-off humidity ratio for the coil bank.

Placement of facilities such as these at this level in the structure is a key factor in the extension of the programme to handle additional system types.

6.2. The Fan Subsystem.

The major classes required to model the fan subsystem are shown in figure 5. Class *FanUnit* is central to the modelling for this subsystem and, in the absence of more detailed information an instance of this class references an object of base class *Fan*, which specifies a default efficiency and fan temperature rise for a 'generic' fan. The user may provide more specific information, and select a fan of a particular type (axial or centrifugal), in which case a fan characteristic (stored as a set of cubic spline coefficients) will also be selected from the fan database. In this latter case, the user may also elect to specify fan system effects at the fan inlet and outlet. The actions taken within the modelling process depend on the information available:

i. If the pressure rise across the fan is not known, the default temperature rise will be used.
ii. If the pressure rise is known or can be calculated, but no fan characteristic is available, the default fan efficiency is used in calculating the fan temperature rise.
iii. If the fan characteristic is also available, the operating point if the fan is found, and the efficiency thus found is used to calculate the fan temperature rise.

A software component to model the thermal and pressure drop characteristics of a duct system is currently being developed (refer Marshallsay, 1996).

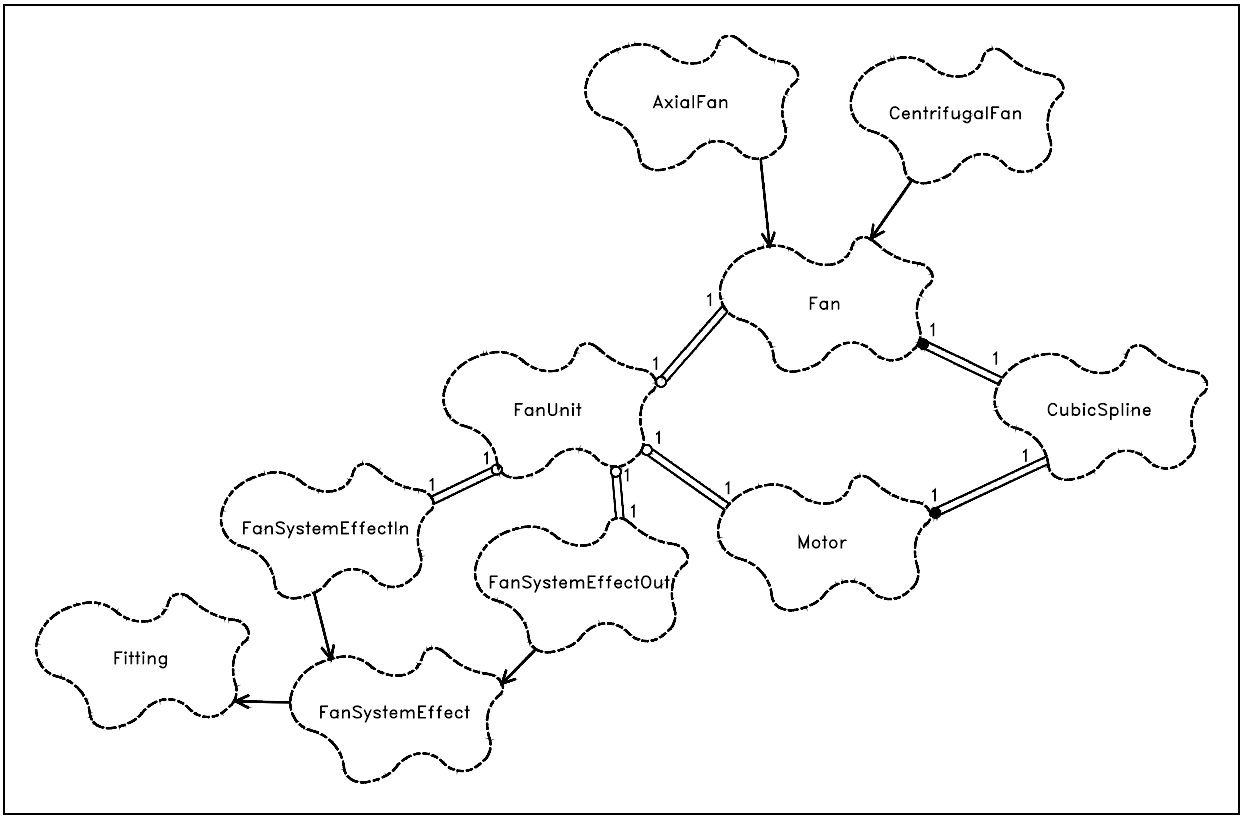

Figure 5. Major classes involved in modelling a fan sub-system.

7. Air Handling Units.

The air handling unit, represented by class *AHU* and its derived classes, provides a framework within which the system modelling algorithms described in section 5 may be implemented. Systems using chilled water as a coolant are modelled with a considerable degree of generality by the hierarchy of classes shown in figure 6. In line with the philosophy enunciated in section 5.4, most of the functionality within this hierarchy is contained within classes *AHU* and *CWAHU*, which respectively provide a general base class for all air handling units, and its specialisation to units using chilled water. Class *AHU* contains member variables representing:

a. An outdoor air sub-system (OASys), this being a pointer to a variable of class *SubSystem*, or its derived class *CWSubSystem*.
b. A return air sub-system (RASys), this being a pointer to a variable of class *SubSystem*, or its derived class *CWSubSystem*.
c. A vector of pointers (SASys) to variables of class *SASubSystem*, or classes derived from class *SASubSystem*, representing the supply air sub-systems[4].
d. A pointer (f) to a variable of class *FanUnit*.

[4] Implementing a vector of pointers rather than a single pointer for this variable provides an opportunity to model a unit having several supply air sub-systems, each with its own cooling coils, a situation which is sometimes encountered in blow-through configurations.

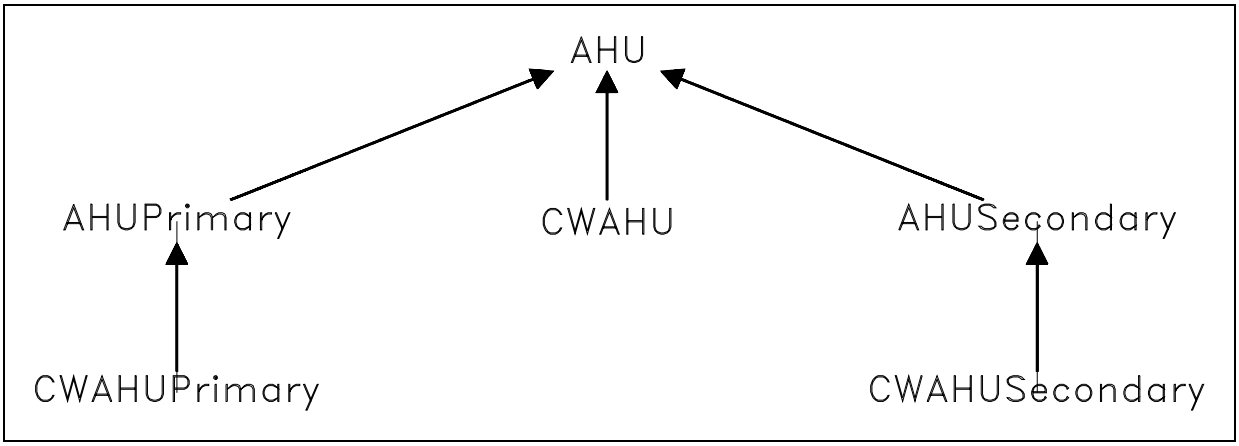

Figure 6. Directed Acyclic Graph showing the inheritance hierarchy for class *AHU* and its derived classes.

These are augmented with a set of member functions, mainly of a low-level[5] nature.

In addition to the pointer variables declared in the base class, class *CWAHU* provides a pointer variable (CoilSys) to a referencing a variable of class *CWSubSystem*. The polymorphic[6] nature of the pointer variables declared within class *CWAHU* and its base class provides a general mechanism for representing chilled water air handling units of fairly general configuration (refer to figure 3). Thus, for an all-air system, the air handling unit would be represented by:

i. For a draw-through unit of conventional configuration (figure 3d):
OASys references a variable of class *SubSystem*;
RASys references a variable of class *SubSystem* (or is null for a unit providing 100% ventilation (figure 3f));
CoilSys references a variable of class *CWSubSystem*;
SASys contains a single pointer referencing a variable of class *SASecondarySubSystem*;
f references a variable of class *FanUnit*[7]

ii. For a unit of HDP configuration (figure 3e):
OASys references a variable of class *CWSubSystem*;
RASys references a variable of class *CWSubSystem*;
CoilSys is null;
SASys contains a single pointer referencing a variable of class *SASecondarySubSystem*;
f references a variable of class *FanUnit*[8]

---

5 The main exception being a function which provides a framework for implementing the moisture staircase iteration for a derived class specifying a particular coolant and configuration.

6 This refers to the ability of a pointer to reference in a transparent manner a variable of the declared class, or of one of its derived classes. Note in this connection that any of these pointers may be null.

7 Variable f will in general only be null in the representation of a terminal unit in an induction system.

The solution algorithms described in section 5 are implemented as member functions of class *CWAHU*. Classes *CWAHUPrimary* and *CWAHUSecondary* provide relatively minor extensions of the base classes to represent the primary or secondary units respectively in an air-water system.

8. Summary.

The ZEBRA package has been conceived as a tool which will assist the HVAC designer in understanding the processes occurring within an air conditioning system, and consequently will provide direction in the search for a near-optimal design solution. In this it differs from most if not all extant plant simulation codes in that it has been designed from the outset with the aim of satisfying the specific needs of the HVAC system designer, rather than the needs of the user whose primary interest is in analysing building energy consumption. The package has been used extensively within the University of Adelaide for design studies, which in some cases have been of a conceptual nature, and in others have been associated with professional design consultancies. The understanding which has accrued from these studies has played 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. Like most of the codes cited in section 2, Zebra is constantly undergoing a process of evolution. Specific issues being addressed in the development process include:

iii. Verification of the range of applicability of the algorithms used, and development of new algorithms to further broaden the range of applicability of the model.
iv. Extension of the modelling process to cover a broader range of equipment.
v. Facilitation of user access to the model through improvements to the user interface, and in the range of tools provided to the user.
vi. Closer integration with the operating environment, and with coordinating processes, as described in section 2.

The above not withstanding, the package has now reached a certain degree of maturity, and it is intended to make it more widely available in the near future.

[8] Variable f will in general only be null in the representation of a terminal unit in an induction system.

References.

BLAST Support Office, 1991, *BLAST User Reference*, Volume 1 and 2, Department of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign, Urbana, Il.

Booch, G., 1994, *Object-Oriented Analysis and Design With Applications*, 2nd Ed., Benjamin-Cummings, Redwood City, CA.

Buhl, W.F., Erdem, A.E., Winkelmann, F.C. and Sowell, E.F., 1993, Recent improvements in SPARK: strong component decomposition, multivalued objects, and graphical interface, *Proc. IBPSA 3rd International Conf.*, Adelaide, Aug. 16-18, 1993.

Clarke, J.A., 1985, Energy Simulation in Buildings, Adam Hilger, Bristol.

Jacobson, I., Christerson, M., Jonsson, P. and Övergaard, G., 1992, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Reading, MA.

Koptchev, I.D. and Luxton, R.E., 1996, A new look at air conditioning design, *Proc. Joint Meeting of Commissions E1, E2, B1 and B2, IIR*, Melbourne, Feb. 11-14, 1996.

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.

Madjidi, M. and Bauer, M., 1995, How to overcome the HVAC simulation obstacles, *Proc. 4th International Conference, International Building Performance Simulation Association*, Madison.

Marshallsay, P.G., 1996, A Methodology for Modelling the Steady-State Thermal Performance of Air Conditioning Systems, *Ph.D. Thesis*, The University of Adelaide.

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 TR-6.

Sekhar, S.C., 1990, Life Cycle Design of Dehumidifiers in Air Conditioning, *Ph.D. Thesis*, The 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.

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.

Stoecker, W.F., 1989, *Design of Thermal Systems*, 3rd Ed., McGraw-Hill, New York.

Stroustrup, B., 1991, *The C++ Programming Language*, 2nd Ed., Addison-Wesley, Reading, MA.

Tang, D., and Clarke, J.A., 1993, Application of the object oriented programming paradigm to building plant system modelling, *Proc. IBPSA 3rd International Conf.*, Adelaide, Aug. 16-18, 1993.