A Dynamic Modelling Methodology for the Simulation of Industrial Refrigeration Systems

A thesis presented in partial fulfilment of the requirements for the Degree of Doctor of Philosophy in Biotechnology and Bioprocess Engineering at Massey University.

Simon James Lovatt, B.E. (Hons) (Cantuar.)

1992

Abstract

A dynamic modelling methodology has been developed for the computer simulation of industrial refrigeration systems. A computer program, RefSim, has been developed which embodies the new methodology. RefSim contains a total of 33 separate models — 11 derived from existing models, six which are substantially enhanced and 16 which are new. In general, these models were derived from thermal considerations and ignored the effect of hydrodynamic processes in the refrigeration circuit. Models can be dynamically linked together as specified by the input data in order to simulate a complete plant. The program includes a set of simulation utilities which reduce the amount of work required to develop models. The object-oriented features of inheritance, encapsulation and polymorphism are used extensively.

Substantial model development was carried out to achieve accurate predictions of the heat release profile during chilling and freezing of food product as product cooling makes the greatest contribution to both mean and peak heat loads in many industrial refrigeration plants. The new ordinary differential equation (ODE) model was tested against finite difference (FD) calculations for a range of product shapes and Biot numbers. The ODE model predicted to within ±10% of the FD calculation during almost all of the cooling process under the test conditions. The ODE model required several orders of magnitude less computation than FD while being capable of extension to shapes that could not be handled by FD.

To test the new ODE model against experimental data, a differential air temperature method to measure the cooling food product heat load profile was developed. Both the FD and ODE methods predicted the heat load profile of freezing meat cartons to within the experimental margin for error (±10%). The ODE model also predicted the heat load profile of freezing lamb carcasses to a similar level of accuracy.

Three refrigeration plants (a laboratory water chiller, a 18500 lamb per day meat processing plant, and a 6000 lamb/1000 beef per day meat processing plant)

were surveyed to obtain data for testing the whole simulation environment. RefSim was found to predict the measured data satisfactorily in most cases. The results were superior to those from a commercial refrigeration simulation environment and comparable to an enhanced version of that environment which included the new ODE product heat load model. Differences between the measured values and those predicted by Refsim were probably more attributable to uncertainties in the simulation input data than to model deficiencies. RefSim was found to be a flexible environment which was general enough to simulate both simple and complex refrigeration systems. Unusual components could be simulated by combining existing models rather than implementing custom models.

Nevertheless, the simulation results have indicated a number of areas for further model improvement. The effects of air mixing and the thermal buffering of structural materials were shown to be modelled poorly for some refrigerated rooms. There is some scope for improving the chilling stage of the ODE product heat load model.

Acknowledgements

The author would like to thank the following persons for advice and assistance during the course of this project:

Associate Professor Andrew C Cleland, Department of Biotechnology, Massey University, Chief Supervisor.

Mr Mark P F Loeffen, Head of Electrical Engineering Section, Meat Industry Research Institute of New Zealand, Supervisor.

Dr Q Tuan Pham, formerly Head of Refrigeration and Energy Section, Meat Industry Research Institute of New Zealand.

The author would like to thank the following persons for technical assistance during the experimental phase of the project:

Mr Robert M Kemp, Senior Technical Officer, Process Technology Section, Meat Industry Research Institute of New Zealand.

Mr James Willix, Senior Technical Officer, Process Technology Section, Meat Industry Research Institute of New Zealand.

Mr David Lindsay, Senior Technician, Process Technology Section, Meat Industry Research Institute of New Zealand.

The author would like to thank the following persons for assistance in gathering plant data which has been used for simulation testing:

Mr Frank Steans, formerly Chief Engineer, Alliance Ocean Beach Plant and the engineering staff of Alliance Ocean Beach.

Mr Co Coppoolse, formerly Head Shift Engineer, AFFCO Horotiu Plant and the engineering staff of AFFCO Horotiu.

The author would like to thank the Meat Industry Research Institute of New Zealand for funding this project and for making available the advice, technical support and industry contacts acknowledged above. In particular the author would like to acknowledge the support of:

Dr A Keith Fleming, Head of Process Engineering Division, Meat Industry Research Institute of New Zealand.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Refrigeration plants in the New Zealand meat industry are characterised by their large size (1 - 6 MW), complexity, and highly variable heat load (Fleming, 1976). The capital and operating costs of the refrigeration system form a significant fraction of total plant costs, so the industry is concerned to contain and reduce these costs. Since the 1960s, the refrigeration group at the Meat Industry Research Institute of New Zealand (MIRINZ) has worked to improve refrigeration system and application design, operation and control.

The work described herein developed from two related streams of research. The Engine Room Control project of the MIRINZ refrigeration group had the goal of developing superior strategies for the control of refrigeration systems. Testing such strategies on a real plant would have required a great deal of expenditure on sophisticated automatic control systems and a degree of risk to product quality should any error be made in setting the control system parameters. Until a programme of testing was complete, this expenditure and risk would have been incurred without any guarantee that the new control system would perform any better than the old.

At this point, the MIRINZ Engine Room Control project became interested in the use of computer simulations of refrigeration systems for gaining a quantitative evaluation of control strategy performance. Of the limited range of refrigeration simulation environments available, the Refrigeration Analysis, Design and Simulation computer programs developed by Cleland (1985*b*) appeared to be most appropriate for the MIRINZ work, and so the RADS environment was evaluated for possible use as a test-bed for advanced control strategies. While RADS was considered generally satisfactory for the purpose of simulating refrigeration systems of the scale found in the meat industry, it was found to be too inflexible for extension to advanced control strategy testing. In addition, it was considered that some of the component models within RADS would require upgrading, and this was also found to be difficult to do while remaining within the existing program structure.

The refrigeration group at Massey University has been prominent in the development of general refrigeration system models with particular emphasis upon food product freezing time prediction. The importance of meat industry refrigeration in New Zealand led Massey University to carry out co-operative work with MIRINZ and several meat processing plants, for example, the work described by Loeffen *et al* (1981) and Cleland (1983) from which the projects reported by Loeffen and Carrie (1986) and Cleland (1985*a*) were partially derived.

At about the time that the MIRINZ group found a need for a flexible refrigeration system simulation environment, the Massey University group perceived a need for similar development and it was discovered that a commonality of objectives existed between the two. The present project was therefore an attempt to develop a dynamic simulation environment with a level of flexibility sufficient to allow its use as a control system test-bed, and to improve the component models which most required enhancement so that the simulation more accurately represented the dynamics of the simulated system.

Chapter 2: Literature Review

The literature relevant to the present work may be divided into four principle areas: refrigeration system and application component models, refrigeration system simulation, and simulation methodologies applied in related fields. The first three areas are covered in detail in this review. The fourth is very broad and only the most important and relevant literature is discussed.

Dynamic modelling and simulation are comparatively recent approaches to the study and operation of refrigeration systems. While individual components have been modelled for some time, the complexity of a plant consisting of many components was beyond the resources of researchers until the early 1970s. Since then, the increasing availability of digital computers has allowed researchers to develop dynamic mathematical models which are sophisticated enough to simulate any refrigeration system. James *et al* (1986*c*), and Cleland (1990, Chapter 10) have conducted comprehensive surveys of the field.

2.1 Refrigeration System Components

2.1.1 Models

Component models are the models of individual system elements which may be linked together to simulate a complete refrigeration system. Refrigeration system components are the hardware which implements a refrigerating heat engine cycle. For a vapour compression refrigeration system, these components would include compressors, condensers, expansion valves, evaporators, refrigerant vessels and pipelines. Control systems may also be considered as refrigeration system components.

This survey of component models emphasises those which have been developed for or used in dynamic refrigeration system simulations. Certain other

models have been included where they provided a basis for the later development of simulation models.

The majority of models discussed below are of the lumped parameter type. Lumped parameter models are formulated by dividing the modelled component into a number of zones, each of which is characterised by a set of dynamic variables. A differential equation for each dynamic variable can be derived from a conserved quantity balance. Algebraic equations are derived to calculate other variable values from the dynamic variables and system input data. The conserved quantities which are most often considered in refrigeration system component models are mass and energy, so a maximum of two differential equations is usually required to characterise each zone. In the case of what Cleland (1985*a*) called "thermal" models, it is assumed that there is little variation in the way in which mass is distributed within or between models, and so only one differential equation (derived from an energy balance) is required for each zone.

Steady state models of refrigeration system components have been used for design purposes since refrigeration systems were first constructed but dynamic models have only been used within the last 50 years. According to James *et al* (1986*c*), in their survey of dynamic mathematical models of refrigeration systems and heat pumps, the first attempts at dynamic modelling in the field were hydraulic analogues (Hrones, 1942), electronic analogues (Hasegawa *et al*, 1965, Finlay and Smith, 1967) and simplified mathematical models capable of solution by Laplace transformation. These models were specific to individual components.

Improvements in digital computer technology allowed attempts at more detailed component models. The vegetable freezing plant simulation of Marshall and James (1975) contained a number of significant new models. All of the refrigeration system component models considered both mass and energy flows. Evaporators were modelled using three zones for the metal (each with a differential equation for temperature) and three zones for the refrigerant (each with energy and mass ODEs). A liquid separator was modelled as two zones, each with energy and mass ODEs.

The compressor suction line was modelled using ODEs to describe refrigerant mass and energy and with a mass flow-dependent pressure drop.

Compressors were modelled using an algebraic equation derived from thermodynamic theory together with an empirical correlation which related volumetric efficiency to compression ratio. A combined intercooler and liquid sub-cooler was modelled in four zones (liquid, vapour bubble, sub-cooling coil and vapour space), each with mass and energy ODEs. The evaporative condenser was modelled in four zones (refrigerant vapour space, boundary layer, refrigerant liquid space, and tubes). Refrigerant state equations were evaluated using correlations fitted to thermodynamic table data. Model predictions were compared with data measured from the real plant and were found to be of sufficient quality to permit the evaluation of alternative capacity control strategies.

Blankespoor *et al* (1976) developed an interactive compressor simulation for use in design optimization. This utilised a hybrid digital/analogue electronic computer in order to take advantage of analogue computation in the solution of the differential equations. The reciprocating compressor was divided into three zones with ordinary differential equations to describe refrigerant pressure, refrigerant mass, refrigerant mass flow rate and compressor valve motion. The model was validated with some success, but it was specific to a particular variety of reciprocating compressor which made general application of its results difficult.

The direct expansion R22 heat pump simulation of Chi and Didion (1982) contained component models for an hermetic compressor and motor, condenser, expansion valve, evaporator, fan and accumulator. Several lumped parametric ordinary differential equations were used to describe each model, each derived directly from energy balance and thermodynamic considerations. Testing showed good correspondence between the simulation and measured results. Model parameters were measured directly from the simulated plant.

Higuchi and Hayando (1982) developed a dynamic model of a thermostatic expansion valve with the objective of establishing a testing method for a TEV in dynamic flow conditions. Having developed the means to measure the response of

the TEV, they derived a transfer function for the valve and generated Bode diagrams to show the effect of changing the parameters of the transfer function. A first-order lag was found to be an insufficient model for a TEV, and it was shown that additional factors for bulb temperature and pressure, and for heat transfer to the sensor bulb were necessary. With these details included, calculated frequency responses for the tested expansion valve demonstrated good agreement with the measured response.

The first simulation of a large New Zealand meat processing plant refrigeration system was presented by Cleland (1983). Many of the component models followed those of Marshall and James (1975), but they were simplified by assuming that the refrigerant mass distribution through the plant remained constant during the simulation. This removed the need to evaluate equations representing refrigerant mass balances and made it practical to implement such a large simulation. The refrigeration plant used the pump-circulation method, so the models could be simplified without substantial loss of accuracy by assuming that refrigerant streams were saturated (with the exception of compressor outlets), and by describing evaporators and pipelines with algebraic rather than differential equations.

Refrigerant vessels and condensers were described using a single refrigerant zone each rather than the two or more used by Marshall and James (1975). Model testing was described by Cleland (1985*a*), and performance was found to be adequate for time scales of several hours and longer, while short term transients were predicted less well.

Yasuda *et al* (1983) developed a dynamic simulation of a laboratory-scale R12 refrigeration system consisting of a single cylinder reciprocating compressor, a shell and tube condenser, a thermostatic expansion valve, and a dry evaporator. The compressor model followed that of Blankespoor *et al* (1976), with highly detailed descriptions of each portion of the compressor and a combination of ordinary differential and algebraic equations which described both the energy and mass balances in all zones. The simulation was tested using step changes in the evaporator outlet static superheat to generate dynamic responses in both the

simulation and the real plant. Comparisons between simulated and measured results showed good agreement.

Bonte and Veldhoven (1983) presented a series of input/output transfer function models for a thermostatic expansion valve, an evaporator and a compressor. Validation was carried out for the TEV and evaporator, showing an adequate simulation of the refrigerant and pipe temperature, but less accurate prediction of the air temperature passing off the evaporator. The simulation was tested in an open loop manner (i.e. without a product or environment model) and so no interaction was possible with the environment.

The simulation system of Glockner and Findeisen (1984) was used by Gunther *et al* (1984) in a very detailed investigation of the transient behaviour of an hermetic compressor during start-up. The model was compared with a real compressor and good agreement was obtained for the discharge temperature and power input during the first 20 minutes of operation. This contrasted with other models which had done well over longer time spans but failed to produce good transient results.

MacArthur (1984) presented a wholly theoretical model of transient heat pump behaviour. This model simulated refrigerant mass flows within the system as well as energy flows. The compressor model used a polytropic compression model with ODEs for refrigerant enthalpy, cylinder and shell temperatures. A fixed expansion orifice was modelled using a conventional orifice equation. An accumulator was modelled in two zones (liquid and vapour), with mass and energy ODEs for each. The condenser and evaporator models were also split into liquid-filled and vapour-filled zones. The model was not validated against a real heat pump, although validation was planned at the time of the initial paper. The model was run in an open-loop manner only, with fixed application conditions.

Beckey (1986) described a dynamic simulation of a heat pump which included consideration of refrigerant flow around the system. The evaporator, compressor, condenser, expansion valve and accumulator were all modelled in detail comparable to that of MacArthur (1984). The simulation was verified against

measurements of the real plant with good results during the first twelve minutes after start-up.

James and James (1986*a*) developed a simulation for the dynamic analysis of a heat pump. This included a six-zone model of an hermetic compressor and its cooling system, a twelve-zone condenser model (three zones for each of refrigerant vapour, boundary layer, tube metal and cooling water), a six-zone liquid receiver (vapour, liquid, metal shell and boundary layer in contact with vapour, and metal shell and boundary layer in contact with liquid), and a two-zone (refrigerant, tube metal) evaporator model. All of the refrigeration component model zones had ODEs derived from energy balance considerations, while those zones which contained refrigerant also included ODEs to describe the refrigerant mass balance. A total of 26 ODEs and 95 algebraic equations were used in the whole simulation. The responses of the model were assessed by James and James (1986*a*) as comparing well with the expected behaviour.

A group of models for simulating severe transients in compression refrigeration systems was described by Rajendran and Pate (1986). The models used were similar to those of MacArthur (1984) and Murphy and Goldschmidt (1984), but additionally included a pressure-volume energy term in the refrigerant energy balance equations. An unusually detailed thermostatic expansion valve model was included which was capable of dealing with liquid, vapour or mixed phase refrigerant flow. No testing was carried out against measured data.

Vidmar and Gaspersic (1987) modelled a natural draft condenser of the type found on domestic refrigerators using a set of partial differential equations (PDEs) with zones at each node for refrigerant, tube metal and air. The PDEs were solved numerically to obtain a plot of refrigerant temperature as a function of time and tube length. Favourable comparisons were drawn with previous verified models. The simulation was entirely independent of any real plant, requiring no experimental values to be input prior to a simulation. Heat transfer coefficients and pressure drops were calculated using built-in correlations. The model was used to study start-up and

shut-down transients as well as the effects of design changes on the operation of the condenser.

Xiao and Yu (1987) developed a mathematical model of a screw compressor with an economizer. Previous detailed compressor models had predominantly dealt with reciprocating compressors, so this model provided an interesting variation. The model was thermodynamic in nature and comparisons with experimental measurements revealed deviations of less than 10% from the empirical variation of performance with operating temperature. No dynamic results were presented.

James (1988) developed a number of component models for use with a flexible modelling environment (FME). All of James' models included ODEs derived from both energy and mass balances. A water-cooled condenser was modelled in two zones (refrigerant vapour and water) with refrigerant liquid modelled by algebraic equations as a part of the refrigerant-water boundary. The heat transfer coefficient was not measured but was predicted from heat transfer theory. A liquid refrigerant separator was modelled in two zones (vapour and liquid) with checks to ensure that the separator did not became full of liquid, or empty (at which point the equations used would have broken down).

Two flooded evaporators were modelled, each in two zones (refrigerant and metal). Energy and mass balance ODEs were given for the refrigerant zones, and energy balance ODEs were given for the metal zones. The pressure drop in each evaporator was estimated using a factor proportional to the square of mass flow and another factor proportional to the change in head. Evaporator metal temperature was measured over time for use as input to the evaporator model during testing, making application models unnecessary. A float valve was modelled with algebraic equations for vapour and liquid flow with valve setting provided by measured input data.

The FME also included a rotary compressor model which used a pseudo-thermodynamic algebraic equation with fitted constants based upon experimental data. Volumetric efficiency was fitted to a linear function of operating speed, but it was assumed not to vary with pressure ratio. The suction line model had one zone

with ODEs for energy and mass balances and a similar pressure drop calculation as had been used for the evaporator model, though it was assumed that there was no change in head along the suction line. The oil separator and the compressor discharge line were modelled together as one zone with energy and mass balances, but no pressure drop calculation.

A number of other components were modelled by James (1988), but the models were not validated, and were only used for testing the FME simulation environment. The models described above were compared with a previously validated set of models. The FME models were found to produce very similar results to the pre-FME models, the plant dynamics being predicted with an error of approximately 15%.

Wong and James (1988) applied the methods of Cleland (1983) to the simulation of a liquid chilling plant. By considering only thermal variation within the plant and by having a constant heat load provided by flowing water, the plant was simulated using only two ODEs: one for condensing and one for evaporating temperature. Wong and James noted that this simplification increased solution speed by a factor of about 100 while reducing the number of equations by ⅔ compared with a full hydrodynamic model. They reported that predictions made by the simplified model compared well with both measured data and an earlier more detailed model.

Wang and Touber (1991) modelled a dry expansion air cooler using a very sophisticated model which included consideration of momentum, mass and heat transfer in the two-phase flow region. The model was made tractable by decoupling the momentum equation from those for heat and mass transfer using similar techniques to those used for refrigerated rooms by Wang and Touber (1990). The model was validated against an evaporator in a test rig with satisfactory results over periods of up to 2500 seconds. Computing requirements were substantial for this model as with the room model, with about 10 hours of computation required on a Sun 3/60 workstation per hour of simulated time. For this reason, Wang and Touber recommended that a combination of models with different levels of complexity would be better for longer term simulations.

2.1.2 Refrigerant thermodynamic and transport property correlations

A substantial step forward in the digital computer modelling of refrigeration system components was taken by Chan and Haselden (1981*a*,*b*,*c*) who published a set of thermodynamic property correlations for the refrigerants R11, R12, R13, R13B1, R14, R22, R113, R114, R502 and R717. The correlations were derived from thermodynamic theory and fitted to the best available experimental data, generally predicting within the experimental margin of error. The correlations were conveniently presented as subroutines in the FORTRAN programming language (ANSI, 1978) but several of the equations used were implicit and required iterative solution, thereby making their direct use impractical for applications where high rates of computation are required, such as dynamic simulation. In the present author's experience, implementation of the correlations exactly as presented by Chan and Haselden (1981*a*,*b*,*c*) required high precision calculation to avoid over- and under-flows in intermediate calculations, although minor modifications allowed them to be implemented with no loss of accuracy.

Cleland (1986*a*) developed a set of computer subroutines to calculate thermodynamic properties for the refrigerants R12, R22, R114, R502 and R717. These FORTRAN (ANSI, 1978) subroutines implemented polynomial equations fitted to the results of Chan and Haselden (1981*a*,*b*,*c*), and were therefore much faster to evaluate than the originals while generally agreeing with the results of Chan and Haselden to within 0.25%. This made the routines much more appropriate for use in dynamic simulations than those of Chan and Haselden, though care was required to avoid exceeding the range of data to which the polynomials were fitted.

2.1.3 Summary

Refrigeration system component model development has been the subject of considerable research, with the result that many of the models discussed in the literature are capable of predicting the thermodynamic behaviour of individual

refrigeration system components with an accuracy close to that of experimental data.

One difficulty in using the more sophisticated models described above was in obtaining the large quantities of data required to evaluate their parameters. Having done this, it was often hard for model developers to fully validate their models as some of the predicted values were difficult to measure. Measurement of refrigerant distribution within a refrigeration system, for instance, has been shown to be a very complex problem (Belth and Tree, 1986). Cleland (1990, p.237) has pointed out that there is a need for modellers to have guidelines to indicate what complexity level would be appropriate to a given modelling problem. Wang and Touber (1991) have shown that even with modern computing resources, it may be necessary to choose models which have reduced levels of complexity in order to make model evaluation practical. Despite these considerations, some workers have continued to develop very complex models with many parameters, demanding validation requirements and long evaluation times.

2.2 Application Components

The behaviour of a refrigeration plant often depends upon factors external to the refrigeration cycle, such as the refrigerated product and its surroundings. These items may be grouped as application component models.

2.2.1 Food Product Models

The purpose of a food product model within a refrigeration system simulation is to characterise any behaviour of the product which may affect the behaviour of the refrigeration system. The most important effect of food product on the refrigeration system is commonly the heat load that it releases as a function of time and ambient conditions during a cooling or freezing process. Other important behaviours to be predicted in specific applications may include heat of respiration, moisture loss (or sometimes moisture gain), extent of freezing at any time during the process, and total

freezing time. As with the discussion of refrigeration system component models above, the present survey is limited to those models which have been used in dynamic refrigeration system simulations and those which provided a basis for the later development of simulation models.

The earliest food product freezing model was that of Plank (1913, 1941), who developed a model for freezing of food product. Plank considered the motion of a notional freezing front passing from the surface to the centre of the product and solved the differential equations governing this motion by analytical methods for one, two and three-dimensional rectilinear product shapes. The resulting equations predicted freezing time if constant external conditions were assumed, but they were not able to predict the heat load profile due to the method of integration.

Marshall and James (1975) modelled freezing vegetable pieces by assuming an effective specific heat which varied depending upon the refrigerated space zone in which the product rested at any given time. The effective specific heat was adjusted so that the product temperature reduction through the tunnel corresponded with experimental measurements. This provided a partially experimental heat load for the simulated refrigeration system.

Cleland and Earle (1982*a*) developed a model for the heating and cooling of solids without phase change which built upon the work of Pflug *et al* (1965). Exact series solutions were calculated for each of the three shapes for which the transient heat conduction differential equation could be solved analytically. These solutions were then used as reference shapes to which the chilling times of other shapes could be related using the equivalent heat transfer dimensionality shape factor, E. The model was found to predict experimentally measured heating and cooling times with an accuracy of ±12%.

Wade (1984) developed a model for the cooling of horticultural produce under moderate Biot number conditions. The model was largely based upon the work of Cleland and Earle (1982*a*) and it used a cooling coefficient and a lag factor, along with a simple solid interpretation of the shape of the cooled material to produce an estimate of the required heat duty. Application of the model

demonstrated that the required refrigeration capacity for a given task could be over-estimated if an average cooling rate was assumed. Predicted heat loads were not tested, although predicted cooling times were compared favourably with previously measured data.

Loeffen *et al* (1981) presented two methods for predicting the freezing times of slab, cylinder and sphere shapes with time-variable boundary conditions. An ordinary differential equation was derived from the work of Plank (1941) to predict the motion of a freezing front from the surface of the freezing body to its centre. Instead of assuming constant boundary conditions so that the equation could be integrated analytically, Loeffen *et al* (1981) integrated the equation numerically so that both heat transfer coefficient and ambient temperature boundary conditions could be varied during the process. The model predictions were compared with the results of finite difference calculations and were found to predict freezing time with time-variable boundary conditions to the same level of accuracy as previous analytically-integrated methods had predicted freezing times with constant boundary conditions.

Cleland and Earle (1982*b*) developed a model for predicting the freezing time of food products under constant ambient conditions which was based upon the work of Plank (1941). The shape factor concept of Cleland and Earle (1982*a*) was used to allow the freezing product to be of any shape. As with earlier work, the emphasis was on chilling and freezing times in constant ambient conditions rather than upon heat load prediction. Experimentally measured freezing times were predicted with an accuracy of ±10%.

The time-variable product heat load models used by Cleland (1983) were derived from the chilling and freezing models of Cleland (1982*a*,*b*). The freezing model calculated the freezing front motion during the process by numerically integrating the governing ODE after the fashion of Loeffen *et al* (1981). The rate at which frozen volume changed with frozen depth was determined individually for each shape to be frozen by assuming that the freezing front maintained its original shape throughout freezing. The rate of volume change with depth was multiplied by the rate of freezing front motion to predict the rate at which frozen volume changed

with time. This was in turn multiplied by the volumetric latent heat content of the product to estimate the freezing heat load at any time during the process.

The chilling model of Cleland (1983) used a numerically-integrated form of Newton's law of cooling to predict the temperature of the product and the product heat load profile during chilling. Heat load prediction was made more accurate by choosing the UA (heat transfer coefficient by surface area) factor so that predicted cooling times corresponded with experimental measurements.

When the work of Cleland (1983) was tested (Cleland, 1985*a*), the freezing product model was further enhanced as described by Cleland (1986*b*). The model was improved to remove the assumption that the freezing front shape remained constant and Cleland (1986*b*) showed that this resulted in significant improvements in prediction accuracy by testing against finite difference freezing heat load predictions. Use of the new model was also shown to improve the prediction of a freezer room air temperature compared with the earlier model.

2.2.2 Food product thermophysical properties

To use any of the food product models described above, one must have know the values of several physical properties for the foods in question. Some of these properties are difficult to measure so it may be necessary to rely upon methods for predicting them from more accessible data.

Jowitt (1983) presented a comprehensive survey of food physical properties. Of the contributions to that survey, those of Miles *et al* (1983) and Mellor (1983) were of most interest to workers modelling food cooling and freezing processes. Miles *et al* (1983) presented a list of published equations for predicting enthalpy, specific heat, ice content, thermal conductivity, thermal diffusivity and density, reviewed the accuracy of some of these equations and selected several for use in a property estimation computer program. Mellor (1983) presented comparisons of predicted and measured properties, along with methods for measuring some properties.

2.2.3 Non-product Models

Food product may not be the only application component to significantly affect the behaviour of a refrigeration system. Heat flows between the product and the refrigeration system are often transferred through some intervening fluid, usually air. The space in which the product is processed may be surrounded by walls which conduct heat. Walls may be pierced by doors and ventilation ducts through which heat and water vapour may pass. The refrigerated space may contain machines, people, and objects with significant thermal mass. It may be necessary to model some or all of these non-product application components in order to accurately predict the behaviour of the application, and thus its effect upon the refrigeration system.

In the vegetable freezing plant simulated by Marshall and James (1975), the refrigerated air space was modelled as eight "stirred-tank" zones, with an ODE representing the enthalpy of air in each zone. Heat flowed from the freezing product through this air space and into the refrigerant evaporators.

Cleland (1983) described each refrigerated room in a large plant simulation using a single mixed zone, with ODEs for air temperature and humidity. This model was also used within the RADS simulation environment (Cleland, 1985*a*,*b*). Cleland (1990) and Cornelius (1991) have detailed the other application models available within RADS simulations. Rooms could have a number of doors, for each of which the heat and water vapour flows from a fixed outside environment were modelled after the method of Tamm (1965), as modified to reflect the measured results of Pham and Oliver (1983).

Room walls in RADS were modelled using steady-state heat conduction theory with surface heat transfer coefficients estimated from air velocities, a thermal conductivity provided by the user and a fixed outside environment. The heat capacity of all structures within the room was represented by one thermal mass with a single ODE to describe its temperature. In air-conditioned applications, heat and water vapour loads could be provided by surfaces of hot water modelled using the

Lewis relationship (Treybal, 1984, p.240). Finally, there could be a fixed sensible heat load on the room at all times, or just during working hours to represent active machinery, lighting and working personnel.

Murphy and Goldschmidt (1984) devised and verified a model of an air-conditioning duct to discover the effect of the duct upon the performance of the air-conditioning plant. The model comprised a finite difference description of heat conduction through, along the duct walls and in the air along the length of the duct. Models for the temperature measurement-thermocouples and air blower blades were based upon first-order response characteristics with time constants calculated from heat transfer relationships. Some difficulties were identified in obtaining the data required for the model to function correctly.

Szczechowiak and Rainczak (1987) developed a dynamic simulation of a cooling chamber and air cooler system which included consideration of the change in air temperature as it passed across the chamber. The chamber was modelled by a system of hyperbolic partial differential equations. The partial differential equations were solved analytically. The solution was verified against a real cooling chamber with excellent results for the air temperature distribution and adequate results for the product surface temperatures.

Wang and Touber (1987) demonstrated the use of a finite difference model for temperature and air velocity distribution in a cold store. By omitting consideration of momentum, the system of equations for macro-scale energy and mass flows in the store was made tractable, while a separate set of partial differential equations described the micro-climate within boxes of product. The temperatures predicted by the model were found to be very similar to those measured in the store itself.

Reynoso and De Michelis (1988) simulated a batch freezer which used direct injection of cryogen into the freezing chamber. Uniform internal product temperature and perfect mixing of the refrigerant medium were assumed during the freezing process. Product and structural material in the freezing chamber were modelled

using finite difference methods. Validation showed a good correlation between the simulated and experimental results.

Pala and Devres (1988) presented a long-term (32 weeks of simulated time) simulation of a cool store which included some detailed application models. Wall gain load was calculated using measured temperatures and a calculated overall heat transfer coefficient. Product and packaging heat load was evaluated by assuming that the product and packaging temperatures would be the same as that of the room 24 hours after loading. Other models described product respiration, product weight loss, air change heat load, door heat load, refrigerant evaporator operation and defrosting using algebraic equations. The model was not tested against experimental measurements.

Wang and Touber (1988) developed three alternative refrigerated room models. One utilized the Navier-Stokes equation to predict mass flow rates around the room. Another model took an experimental approach by recognising the analogy between heat and mass transfer, and using temperature measurements in the room to indicate energy and mass flows after the manner of Wang and Touber (1987). This second approach required that there be a substantial heat source in the room. A third approach simplified the Navier-Stokes equation by using a resistance network concept and ignoring natural convection and external force effects. This third approach was shown to be more useful than the second, and more practical to deal with than the first approach. The third model predicted temperature and humidity distributions within the room. The temperature distribution predicted by the third model was found to be in good agreement with measured data.

Wang and Touber (1990) described a sophisticated model of a refrigerated room which used the three-dimensional Navier-Stokes equation decoupled from expressions for energy and mass transfer within the room. Decoupling allowed the air flows in the room to be calculated once while the energy and mass flows were calculated repeatedly for the changing conditions in the room. This substantially reduced the computer resources that were required to solve the model equations over a period of simulated time. The energy and mass transfer equations were solved by

finite difference methods. Despite the economies made by decoupling, Wang and Touber noted that predicting the air flow patterns in the room still required about 100 hours of computation on their Sun 3/60 workstation. The model was validated against experimental measurements and found to produce satisfactory results over periods of up to 24 hours.

2.2.4 Summary

Application models have received less attention than models of refrigeration hardware and are consequently less well developed. Most food product models were unsophisticated in comparison with the models used for refrigeration system components, despite refrigeration of food product frequently being the main purpose of the refrigeration system. A lack of accurate food property correlations has limited the complexity of cooling and freezing models to some extent, but this is becoming less of a problem.

Refrigerated rooms have usually been described either as single stirred tanks, or with highly complicated three-dimensional partial differential equations. Other application components (such as doors, walls, ventilation and structures) have received less attention, with the sometimes important dynamic effect of thermal mass in the refrigerated space being treated simply or not at all.

2.3 Refrigeration system simulation environments

Component models may provide valuable insight into the characteristics and behaviour of individual elements in a refrigeration system. If the task is to predict the behaviour of the whole system, or to predict the effect of the refrigeration system upon other systems (such as ambient air, food product, or condenser cooling water, for instance), then component models representing each element of the refrigeration system must be linked together and solved in parallel. This is the task of a simulation environment.

Several types of simulation environment may be identified (partially following Kreutzer, 1986, p.91):

A. Specialised models custom-built using general purpose programming languages.

B. Specialised models custom-built using general purpose programming languages with extension libraries which include simulation features and utilities which are commonly required by models.

C. Specialised models described in a general simulation programming language.

D. General models linked together and parameterised to describe a particular system, using a simulation environment specific to the sort of system to be simulated.

E. Model generators which produce source code in a general programming language.

Each of these approaches has its advantages and disadvantages. Freedom in the choice of model representation decreases from type A to type E, while the ease of simulation preparation increases from type A to type E as the descriptive level becomes higher. Refrigeration system simulations which use each of the above environment types are described below. Many reports of refrigeration system simulations have concentrated upon the models used and provided little detail on the simulation environment. Such cases have not been included in this discussion.

2.3.1 Custom-built simulations in general purpose programming languages

The first dynamic mathematical model of an entire refrigeration system was that of Marshall and James (1975). This was also the first recorded work in which a digital computer was used to solve refrigeration system component model equations and so must be seen as the progenitor of most of the subsequent work in the refrigeration system simulation field. The models were expressed as a set of

equations in a general purpose programming language with simulation utilities included along with the models. Linkages between models were implicit due to the nature of the environment. Little information was provided on the structure of the simulation system or on the available simulation utilities as the report concentrated mainly on the models used.

The system of Chi and Didion (1982) was described in more detail. Their heat pump model was expressed as a FORTRAN (ANSI, 1978) computer program which comprised over 2500 statements. The program included functions to calculate thermodynamic and transport properties for air and refrigerant, functions for calculating heat and mass transfer rates under various conditions, and subroutines for evaluating the dynamic responses of each component model. Euler's method was used to solve the ordinary differential equations in the model with time steps of 0.005 seconds required to maintain close error tolerances. The small time step did not cause substantial problems because the model was developed for the prediction of short-term startup transients and runs typically proceeded for only a few minutes of simulated time. Linkages between models were expressed in the main program body where linking variables were maintained. A variable naming convention assisted in making clear the nature of each variable and which model output it represented.

James and James (1986*a*) modelled a heat pump both to study its dynamic behaviour and to study the modelling techniques used. The simulation program was written in FORTRAN, and a complete listing was provided. Linkages between models were unclear and model boundaries were indistinct. Dynamic variables were stored in an array with only an index number to identify them. Despite liberal comments in the program and variable names which were as informative as was possible within the six-character length limit imposed by the language, the models as expressed in the program were difficult to understand.

James and James drew a number of conclusions about the approach that they (and most of the other workers mentioned here) had taken in their simulations:

1 Most of the simulations were "custom built" and specific to the problem in hand.
2 The computer programs were not "user-friendly".
3 The programs were not easy to modify, maintain and correct.
4 The programs were difficult to understand.
5 It was difficult to link new component models into existing models.
6 An extensive knowledge of mathematical modelling, thermodynamics, refrigeration systems and the derivation of the model was required in order to use current simulation environments.
7 Component models were not necessarily compatible with each other.

James and James (1986*a*) decided on this basis that a new type of refrigeration system simulation environment was necessary (see section 2.3.5).

2.3.2 Custom-built simulations making use of simulation libraries

Nowotny (1983) described the STASAN/DYSAN system which included both steady state simulation of refrigeration systems with statistical sampling of variations (STASAN) and dynamic simulation (DYSAN). Nowotny reported that DYSAN solved ODEs using an adaptive time step Runge-Kutta method, but little further information was available on other simulation utilities, availability of property correlations, input or output techniques. Both systems were written in the FORTRAN IV programming language.

Bullock *et al* (1983) modelled a residential air-to-water heat pump using a modular system called TRNSYS. No models were described and little detail was provided on the simulation system itself. According to James (1988), TRNSYS solved ODEs using a modified Euler method.

Glockner and Findeisen (1984) developed the LF74 state-space and heat balance network modelling environment as a basis for a complex system simulation, including simulation of refrigeration systems. This environment was implemented in FORTRAN, with subroutines written by the user to implement specific model types. LF74 supported steady-state and dynamic simulation with ODEs solved by a variable step length method. A number of workers subsequently reported using this system for a variety of simulation models, for instance that of Gunther *et al* (1984). It appears that some refrigerant property subroutines were available.

2.3.3 Custom-built simulations in specialised simulation programming languages

It may be expected that a number of refrigeration system models have been implemented in specialised simulation programming languages, but few have been reported as such or described in detail. One use was presented by Darrow (1990), who used the ISIM simulation language (Hay and Crosbie, 1984) to describe a laboratory water chiller plant. ISIM provided a range of facilities for solving ODEs using high-order Runge-Kutta methods which included integration error control, but program structuring, input and output flexibility were limited when compared with general purpose programming languages. As a continuous system simulation language, ISIM had no features for dealing with discrete events. The results of this work were reported by Darrow *et al* (1991) (of which a copy is included in this thesis as Appendix 4.3), though the simulation environment was not discussed in that paper.

2.3.4 Refrigeration system simulation environments with generalised models

Thorbergsen (1985) reported the development of the PROSIM steady-state heat pump simulation environment. PROSIM used linear input-output models for all refrigeration components and, although it was not a dynamic simulation environment, it was worthy of note because of the degree of flexibility available to the user. The

system to be simulated was described in a declarative input language which indicated the parameters of the component models and showed the linkages between them as numbered streams. This language was understandable both to the user and to PROSIM, removing the need for a separate data input program. The output produced by PROSIM was also superior to most dynamic simulation systems in that it was capable of plotting diagrams of the simulated system and of producing graphical summaries of the simulation results.

Cleland (1985*a*,*b*) made the first attempt at a generalized dynamic refrigeration system simulation environment when the RADS system was developed. RADS comprised a suite of computer programs which allowed the dynamic simulation of almost any refrigeration system for which the required parameters could be obtained using the models described in the component models section above. The original papers contained little detail about the simulation system itself, so the following discussion is largely based upon the information provided by Cornelius (1991), which described a later version.

The numerical methods used in RADS were superior to most of the simulations mentioned here, with fourth order ODE solution techniques being used exclusively. Time step length was decreased automatically during periods when temperature variables were changing rapidly. This allowed simulation of large plants for extended periods with reasonable levels of accuracy and without excessive computational cost. Model initialisation was handled by assuming that the plant operations cycled every 24 or 48 hours. The simulation was run for several cycles and the last cycle was considered to be reliable if it repeated the output of the penultimate cycle.

Discrete events were handled by checking conditions at the start of each timestep and changing each discrete state if its condition was satisfied. This meant that it was always possible for any event to be up to one time step late, although this was not a major problem as time steps were usually about 30 seconds long (Cornelius, 1991). RADS was the first refrigeration system simulation environment which was reported to deal explicitly with discrete events.

Refrigerant property calculations were available for the range of refrigerants correlated by Cleland (1986*a*). The RADS environment included models for application components as well as refrigeration system components, so that complete plants could be simulated without the need to provide the application heat load as input data. RADS was written in portable FORTRAN-77 (ANSI, 1978) and it was available on a range of computers from IBM-compatible personal computers to mainframes.

Compatibility with powerful computers, advanced numerical methods, and the use of thermal models which were less computationally-intensive than many of the others discussed above meant that RADS had the capability to simulate large plants such as that described by Cleland (1985*a*). The ability to deal with large systems allowed this package to present results of value to industrial users of refrigeration rather than (as with many previous dynamic simulations) being of value only to researchers and designers of small systems. This advantage was enhanced by the flexibility of the package which allowed the simulation of many types of plant without the need to re-write the program itself. On the other hand, if the large variety of models within the system proved insufficient then, in the present author's experience, it was difficult to add new models or control systems.

2.3.5 Refrigeration system simulation generators

Following the conclusions drawn by James and James (1986*a*), James (1988) developed a "Flexible Modelling Environment" (FME) by applying the techniques of "structured programming" to the dynamic refrigeration system simulation problem. The FME comprised a group of programs written in a DEC VAX dialect of Pascal (Jensen and Wirth, 1985). The programs included menu-driven interfaces to the Vax system editor, Pascal compiler, linker, and command processor, and the simulation program itself. A main Pascal source file, a set of component modules and a refrigerant properties module were linked together after compilation using the system object file linker to produce the executable simulation program. Separate files

described the model instances and listed numerical indices to indicate the linkages between models which existed in each simulation. In the categorisation above, the FME could best be described as a combination of types B and E because simulation programs were not generated entirely automatically. Significant features of the FME included:

- clear definition of model boundaries.
- the concept of a standard model template into which component models were fitted.
- the concept of model compatibility.
- the use of unbounded data structures so that the number of component models in a simulation was not arbitrarily restricted.
- the identification and implementation of a methodical modelling technique for application to a wide range of models.

Several difficulties remained unresolved, some of which were identified by James (1988). No general ODE solver was available within the FME, so models which required the solution of ODEs had to include the code to do this amongst their model evaluation code. In practice, the time required to implement a sophisticated high-order ODE solver within every dynamic model would have been prohibitive and FME models would have been restricted to the use of low-order ODE solution methods. Low order methods require very small time steps to retain integration accuracy and stability while having no easy way to check that accuracy unless the simulation is repeated with an even smaller time step.

No explicit provision was made for dealing with discrete events. If such events had occurred, the results produced by the FME simulation would have been poor for some time after the event.

Generation of a specific plant simulation required the user to be conversant with the Pascal programming language, and to have access to the main program source code so that it could be modified to correspond with the new simulation.

This risked the possibility of the user inadvertently altering something important to the correct operation of the simulation system.

While most of the model templates were well-suited to the test cases used by James (1988), they were inappropriate to models which had links to many other models. James struck this difficulty with pipeline and tee-junction models but it was not satisfactorily resolved.

2.4 General System Simulation

Substantial progress has been made since the early 1960s in the development of general modelling and simulation methodologies which may be applied to a wide range of problems in the physical, social, and management sciences. The literature in this area is extensive and much is not directly useful to the simulation practitioner. This discussion therefore concentrates upon a few major surveys which were especially beneficial to the author in developing an understanding of the field.

A review by Zeigler *et al* (1979) included many papers which cast light upon the problem of applying a methodology to systems modelling and simulation. Among these, Zeigler (1979) in the introduction identified the similarities and differences between the concepts of design and modelling. Design has a definite objective, a tangible end product, and a definite life span. On the other hand, the objectives of a model are often indefinite, the end product is usually something as intangible as "understanding", and a model invariably has a life cycle which involves repeated cycles of formulation, implementation and validation.

Oren (1979) discussed the application of computers to the modelling problem, the classification of model types, the acceptability of models and their representation as computer programs. Elzas (1979) constructed a definition of "robust simulation" which covered computational algorithms, assuring model description validity, program integrity, program structure, program readability, and bounds for permissible experimentation. A robust simulation, in Elzas' view, ought to:

- free the user from artificial constraints which would otherwise restrict use of the simulation.
- ensure that the models are appropriately combined and used.
- protect against numerical pitfalls and attempts to use the models outside their bounds of applicability.

Cellier (1979) demonstrated the features required of simulation languages which are to be capable of combined continuous and discrete event simulation. In this case, the simulated system is defined by a set of differential equations but one or more variables may be discontinuous during the simulation run. For such systems, the efficient execution of a simulation requires consideration of both the discrete and continuous aspects of the problem.

The proceedings edited by Oren *et al* (1984) continued in the vein of Zeigler *et al* (1979). Elzas (1984) considered the concept of looking at a simulation as a reality mapping. The methodologies of bottom-up (feature extraction) modelling and top-down (hypothesis verification) modelling were compared and shown to be largely compatible within a larger simulation framework. It pointed out that it is difficult to establish the degree of realism of a model with reference to behaviour and structure for large systems where one cannot influence the design of the experiment from which data is obtained.

Crosbie (1984) looked at the difficulties encountered by numerical integration techniques in a combined continuous and discrete event simulation where it must be ensured that integration proceeds through discontinuities efficiently and accurately. Methods were discussed by which discontinuities could be identified so that the numerical integrator could step exactly up to each discontinuity and then proceed from the other side, rather than attempting to step over the discrete event.

Kreutzer (1986) presented a comprehensive survey of simulation languages and methodologies, including discrete event, continuous, and combined simulations, ranging from the use of general purpose programming languages to model generators specialised for simulating particular systems. For the most complicated simulation

problems, and particularly for combined simulations, Kreutzer recommended the use of object-oriented design and programming methods. Examples of object-oriented simulation applied to various problems were compared with other methodologies.

The object-oriented system which has received the most attention in the literature is the Smalltalk programming language and its environment. Lalonde and Pugh (1990) presented a detailed discussion of both object-oriented design and programming in the context of the Smalltalk environment. One way in which they characterised the object-oriented methodology was as "programming by simulation" (Lalonde and Pugh, 1990, p.1).

2.4.1 Summary

Many features and concepts which have long been utilized in general system simulation had not been applied to refrigeration system simulation. James (1988) was the first to apply structured programming techniques and a modern programming language (Pascal) to refrigeration system simulation, while object-oriented simulation systems (which are in many respects a generation ahead of structured techniques) had been developed in other fields since 1967 (Kreutzer, 1986, p.86).

In the area of numerical methods, most workers in refrigeration system simulation have continued to use elementary techniques (e.g. Euler's method) for ODE integration, despite a lack of error control and the extensive computer resources required to solve ODEs by first-order methods. Only one worker (Cleland, 1985*a*) has explicitly dealt with refrigeration systems as combined rather than as exclusively continuous simulations.

Chapter 3: Preliminary Considerations

Models of refrigeration system components have been developed by researchers for many years. Less attention has been given to application components such as food product, the rooms in which they are processed and the structural components within those rooms. Chilling and freezing of food product produces a large fraction of the heat load which is extracted by a meat processing plant refrigeration system and it has a dominant effect upon the dynamic behaviour of the refrigeration system in such a plant (Cleland, 1985*a*). Food product was therefore identified as the component model which required the most improvement over existing models.

For all models, there was a need to define the boundaries and content of each component more clearly than had been done by many researchers in the past. If models were more clearly defined, then similarities between models could be recognised. Model development effort which had been spread over several models could be concentrated upon common features. For example, a given model may, in fact, be better described as a combination of two or more models of greater generality. Parallel development of structurally similar models would economise on model development and testing effort, and also raise model reliability due to the increased level of testing to which fewer individual model types would be put.

It was important to develop a model format into which a wide variety of models of varying complexity could fit. If a particular simulation were to require a very detailed model of one component, it should be possible to apply a detailed model to that component without necessarily using a high level of detail in other component models. Thus models originating from different sources and with different complexity levels could be "mixed and matched" to make simulations appropriate to each task.

A simulation environment must allow a set of general models to be linked together to form a simulation of an actual or hypothetical plant. If a collection of very general models were available, then the complexity of simulation that could be

performed would increase rapidly with the number of models available. It may not make sense to link certain types of models together, but within that constraint, a model which is flexible enough to be linked to a large variety of other models is clearly superior to one which can be linked to a only a few other models. A group of general models in a simulation environment may then be used for purposes well beyond those originally considered by the environment and model designers.

A simulation environment powerful enough to deal with complex systems requires a similarly powerful method by which the user may describe the simulated system. This descriptive method must be capable of handling systems which may have a large number of interactions between models and be intelligible both to the user and to the simulation environment. Similarly, there must be a method through which the models can communicate the values of their state variables back to the user.

The simulation environment must ensure that model evaluation is both accurate and efficient. Methods of model evaluation (numerical integration of differential equations, for example) should not introduce errors which compromise the accuracy of the models.

The refrigeration system behaviour of interest to this project extends over periods of hours or days. It was desirable to implement the simulation environment in such a way that this behaviour could be accurately simulated at a rate significantly greater than that of the real phenomena while using a commonly-available personal computer.

This work had the following objectives:

(a) Develop a new model for the heat load produced by chilling and freezing food product.

(b) Develop an experimental method to measure the heat load released during the freezing of actual product items.

(c) Test the product heat load model against existing numerical methods and experimental data.

(d) Design a simulation environment for the combined continuous/discrete event simulation of industrial refrigeration systems.

(e) Implement the simulation environment together with a set of component models sufficient for testing purposes.

(f) Compare overall simulated results with data measured for real refrigeration plants and the results predicted by earlier simulation environments.

(g) Compare the strengths and weaknesses of the new simulation environment with those of earlier simulation environments.

Chapter 4: Product Heat Load Model Development

This chapter describes the development of a model for predicting the heat released by chilling and freezing food product. Much of this work has previously been reported in a manuscript by Lovatt *et al* (1992*a*) which is reproduced in Appendix 4.1.

4.1 Objectives

The review of simulation component models in Chapter 2 showed that application component models have received less attention from previous researchers than have refrigeration system component models. One area which has been studied by only a few workers is the modelling of heat load arising from food product cooling and freezing. This was not a serious deficiency for those simulations which were concerned only with the behaviour of refrigeration plant under tightly-specified operating conditions because those operating conditions were provided as independent input variables. The lack of accurate food product models was more important where simulations were concerned with the behaviour of refrigeration applications and their effects on the refrigeration plant behaviour. For these simulations, a poor product heat release model could significantly restrict the accuracy of the overall simulation.

Cooling and freezing of food product is the principle purpose of the meat industry refrigeration plants of interest to this project. Steady state analysis (using the RADS APPLICS application analysis program of Cleland, 1985*b*) of the large meat processing plants used for simulation testing in Chapter 10 showed that 44% of the daily refrigeration load at Alliance Ocean Beach and 43% of the daily refrigeration load at AFFCO Horotiu resulted from food product cooling. As well as comprising a large part of the total load, the variation in heat load during the cooling process is an important contributor to the peak heat load sustained by the plant.

One would expect more heat to be released during the early part of the process than at the end due to differences in temperature driving force but there is little quantitative information available on the shape of the heat load profile. This lack of information is of concern when implementing a dynamic refrigeration system simulation which involves cooling product.

4.2 The food cooling process

The food product cooling process can be considered to consist of three stages. First, there is a "chilling", or pre-cooling, stage during which the product cools from its initial temperature, releasing sensible heat. This stage comes to an end and "phase change" starts when the surface temperature drops below the initial freezing temperature and ice formation commences, although there is no sharp transition between the stages. At the end of the chilling stage, the mass average temperature is still above the freezing temperature, and the remaining sensible heat is removed in the phase change period. The rate of cooling can be characterized by the Biot number, Bi:

$$Bi = \frac{h\,X}{k} \qquad (4\text{-}1)$$

where:

h	is the surface heat transfer coefficient /	W m^{-2} K^{-1}
X	is the critical depth /	m
k	is the thermal conductivity of the body /	W m^{-1} K^{-1}

At slow rates of cooling ($Bi \rightarrow 0$), the difference between the mass average and surface temperatures is small, so that little sensible heat remains at the end of chilling. In contrast, at high rates of cooling ($Bi \rightarrow \infty$), the surface is much colder than the mass average, with the result that the chilling stage is short, and ends with much of the material still significantly superheated.

Once the phase change period commences, the dominant influence on heat load is the latent heat of freezing. Initially, freezing commences at the surface, and a

distinct (but not sharp) "freezing front" moves into the object. For this front to pass through a region, both the latent heat of freezing and the residual superheat must be removed, so it is common to treat any residual superheat and latent heat as a lump sum.

Even after the freezing front has passed, the material is not completely frozen and has not yet cooled to the ambient temperature. Further ice formation and sensible cooling will occur as the region progressively drops in temperature. Hence a third stage, that of "sub-cooling", can be distinguished. Sub-cooling commences once the further cooling of the region through which the phase change front has already passed dominates the heat release from the phase change front surrounding the shrinking unfrozen region. Again, the transition is not clear-cut. When $Bi \rightarrow 0$, the surface temperature remains higher than when $Bi \rightarrow \infty$, so in the former case, the transition occurs when the unfrozen region is very small, while for the latter there may be a significant fraction of the whole object still completely unfrozen when the transition to sub-cooling occurs.

The three stages of the process have quite different heat load vs. time profiles, so to be able to predict heat load accurately for the whole process, a model needs to take the characteristics of each into account. Of the various available methods that estimate the cooling heat load profile, finite difference (FD) and finite element (FE) methods have the strongest theoretical foundation, are known to predict freezing times within the margin of error for experimental measurements and may be expected to predict heat loads with similar precision. Unfortunately, they both have significant disadvantages which make them inappropriate for use in a dynamic simulation program:

1 FD may be conveniently used for only a small range of regular shapes. FE can deal with a wider range of shapes, but is still limited to shapes that can be described using only a few parameters.

2 It can be time-consuming to prepare an FD calculation, and more so for an FE calculation.

3 FD calculation is computationally intensive and FE is even more so. In the author's experience, both FD and FE require too much computer memory and execution time for these methods to be used as part of a complete refrigeration system simulation on a personal computer.

An alternative group of methods includes those which utilize a small number of ordinary differential equations (ODEs) to describe the problem. These may sometimes be integrated analytically if simplifying assumptions can be made, but are still much quicker to prepare and run than FE or FD methods when integrated numerically. Empirical factors are easily included when required. ODE methods were therefore the focus of this study.

4.3 Previous work

4.3.1 Freezing heat load

The first detailed dynamic food refrigeration system simulations were carried out by Marshall and James (1975). In that work, the product in a continuous vegetable freezing tunnel was characterized using a lumped parameter model with an apparent product specific heat capacity which varied through the process to account for latent heat. The product heat load was described by:

$$\phi = V\,C\,\frac{dT_{ma}}{dt} = hA_s(T_a - T_{ma}) \qquad (4\text{-}2)$$

where:

- ϕ is the rate of heat release from the product item / W
- V is the volume of the product item / m^3
- C is the volumetric specific heat capacity of the food material / $J\,m^{-3}\,K^{-1}$
- T_{ma} is the mass average temperature of the product item / °C
- t is the time / s
- A_s is the surface area of the product item / m^2

T_a is the ambient temperature around the product / °C

The apparent volumetric specific heat capacity of the product, C, at any given time, was estimated from the amount of time that a product item had been in the freezer rather than from the amount the product was frozen, or its temperature. The model was accurate as long as the refrigeration plant was working correctly, but it was difficult to extend to other situations. Further, equation (4-2) implicitly requires that $T_s = T_{ma}$, (where T_s is the product surface temperature) which is true only if $Bi = 0$. Models such as this which do not take the internal temperature profile into account may be expected to be accurate only under low Biot number conditions (typically less than 5% error below $Bi = 0.1$, according to Welty, 1978, p.119). This approach is therefore inaccurate for most food freezing situations (where the Biot number is more commonly between 1 and 10) unless h is adjusted somehow to reflect the internal temperature profile.

Reynoso and De Michelis (1988) used a similar approach, but more correctly made the apparent specific heat capacity a function of temperature and position within the product, with the consequent drawback that they had to use finite difference methods to solve the equation.

Cleland (1986*b*) presented a method which was used in the RADS (Cleland, 1985*a*,*b*) simulation environment and comprised a modified form of Plank's (1941) ordinary differential equation model. Cleland's method simplified the prediction problem by separating it into (a) a technique for predicting the position of the freezing front, and (b) a technique for estimating the volume of product enclosed by that freezing front; that is:

$$\frac{dx_f}{dt} = \frac{(T_f - T_a)\,E}{\Delta H_{10}\,E_{ref}\left[\frac{2P_{Cl}}{h}\left(\frac{x_f}{X}\right)^{E_w-1} + \frac{8R_{Cl}x_f}{k_s}\left(\frac{x_f}{x_{mCl}}\right)^{E_w-1}\right]} \qquad (4\text{-}3)$$

$$\frac{dV}{dx} = K_{Cl} E x^{E-1} \qquad (4\text{-}4)$$

(and hence: $V = K_{Cl} X^{E}$)

$$\phi = \frac{dV}{dx}\frac{dx_f}{dt}(\Delta H_{10} + C_l(T_i - T_f)) \qquad (4\text{-}5)$$

where:

x_f	is the freezing front position (between X and 0) / m
T_f	is the freezing temperature of the food material / °C
E	is the Equivalent Heat Transfer Dimensionality
ΔH_{10}	is the change in enthalpy between T_f and -10°C / J m^{-3}
E_{ref}	is the E value of a reference shape
P_{Cl},R_{Cl}	constants (Cleland, 1986*b*)
k_s	thermal conductivity of the frozen material / W m^{-1} K^{-1}
x_{mCl}	"mean radius" (Cleland, 1986*b*)
K_{Cl}	a constant (Cleland, 1986*b*)
C_l	the unfrozen volumetric specific heat capacity / J m^{-3} K^{-1}
T_i	the initial temperature / °C

The Equivalent Heat Transfer Dimensionality (E) was used to deal with irregular shapes. Empirically, E may be viewed as being defined by equation (4-6).

$$t_{f\,(body)} = \frac{t_{f\,(slab)}}{E} \qquad (4\text{-}6)$$

where:

t_f	is a freezing time / s

That is, a body with a given E value will freeze E times faster than an infinite slab. Until the work of McNabb *et al* (1990*a*,*b*), equation (4-6) and correlations derived by combining it with experimental data were the only available methods of determining E for complex shapes, but it may now be determined from the physical shape of the product as described by Hossain *et al* (1992*a*,*b*). While E is largely

dependent upon the geometry of the freezing body, it is also a weak function of the Biot number.

Cleland's method was found to be accurate for some simple shapes during that portion of the freezing process where latent heat release was dominant and when the Biot number was between 0.1 and 10. This corresponded with the assumptions made when the model was developed — specifically, the lumping of sensible heat with latent heat. While this was a significant improvement over the earlier models, it had the disadvantage that it relied upon an appropriate choice of reference shape, to which the predicted heat load could be very sensitive. A better method would predict chilling and sub-cooling heat loads as well as the phase change heat load and would not require a reference shape.

4.3.2 Chilling heat load

Chilling heat load models have been developed by Cleland (1983), Cleland (1985*a*), Wade (1984), and Szczechowiak and Rainczak (1987). All were of similar form to equation (4-2), except that the heat capacity was constant with product temperature. The models of Wade (1984) and Cleland (1985*a*) improved upon the basic technique by using the half-life method of Cleland and Earle (1982*a*) to correct the cooling rate for the implied $T_s = T_{ma}$ assumption when $Bi > 0$. This method was found to be satisfactory for most food product chilling applications and for a wide range of product shapes, but it was not directly compatible with any of the freezing models.

There have been no previous models specifically for sub-cooling of frozen product, although Cleland (1985*a*) included a model identical to that for chilling which ignored the possibility of residual latent heat.

4.4 The present work

A new method was sought to predict food product heat release over the whole cooling process. Its desired attributes were as follows:

1 The method had to be accurate over most (preferably all) of the process.

2 The method had to be accurate over a broad range of Biot numbers.

3 The method had to be extensible to all shapes of food product (including complex shapes such as lamb and beef carcasses).

4 The method had to be general enough to handle both chilling and freezing processes.

5 Ideally, the method should have a sound theoretical basis, and require a minimum of empirically-derived parameters.

6 The method had to be convenient to implement and practical to compute on a personal computer as part of a dynamic simulation which may include hundreds of models of this type.

4.4.1 Theoretical development

Given the discussion above, an ODE model appeared to hold the most promise. Following the discussion of section 4.2, it was appropriate that the structure of the product heat load model should mirror the structure of the food cooling process. A heat release profile calculation should commence with the chilling model, transitioning to the phase change, and thence to the sub-cooling model when appropriate. It will be seen that correct selection of the transition criteria is important.

For the chilling and sub-cooling periods, equation (4-2) was considered a reasonable beginning. The idea of tracing the freezing front movement first proposed by Plank (1941), and the relationship between that and the product heat release rate

developed by Cleland (1986*b*) provided a starting point for a model of the phase change period.

If the resulting ODEs were integrated numerically, then time-variable conditions (for instance, surface heat transfer coefficient and ambient temperature) could be taken into account after the fashion of Loeffen *et al* (1981). In the special case of constant conditions, it would be advantageous if the ODEs could be integrated analytically. Many refrigeration systems are designed for mean environmental conditions, and in such cases, an analytical solution could predict the heat release profile without the need for numerical integration.

4.5 Chilling model

The model used for the chilling stage heat load follows that of Pflug *et al* (1965), as extended by Cleland and Earle (1982*a*) to deal with a greater range of shapes. The general equation for one-dimensional transient heat conduction is given by:

$$\frac{1}{x^n}\frac{\partial}{\partial x}\left(x^n\frac{\partial T}{\partial x}\right) = \frac{1}{\alpha}\frac{\partial T}{\partial t} \qquad (4\text{-}7)$$

where: n is a shape factor. n = 0, 1, 2, for an infinite slab, infinite cylinder and sphere respectively.

α is the thermal diffusivity / $m^2\ s^{-1}$

For these three cases, there are exact analytical solutions and so the approach of Cleland and Earle (1982*a*) was to calculate solutions for a simple reference shape, and use E to estimate solutions for more complex shapes from that reference. For reasons which will be discussed below, it is generally most useful to use the sphere as a reference shape when dealing with food products. For a sphere, the solution for the fractional unaccomplished mass average temperature change, Y_{ma}, for cooling under constant conditions may be found to be:

$$Y_{ma} = \frac{T_{ma} - T_a}{T_i - T_a} = \sum_{i=1}^{\infty} \frac{6\ Bi^2}{\beta_i^2 \left[\beta_i^2 + Bi\ (Bi-1)\right]} e^{-\beta_i^2 Fo} \qquad (4\text{-}8)$$

where: Fo is the Fourier number (dimensionless time)

β_i is ith root of:

$$\beta \cot\beta + (Bi - 1) = 0 \qquad (4\text{-}9)$$

During all but the initial period of the cooling process, the terms where $i > 1$ are insignificant. In that case, the slope of $\ln(Y_{ma})$ as described by the abbreviated series solution ought to equal the slope of the integrated equation (4-2) when expressed in terms of Y_{ma}. This implies that:

$$hA_s = \frac{E}{3} \frac{V\beta_1^2 k_l}{X^2} \qquad (4\text{-}10)$$

where: k_l is the thermal conductivity of the unfrozen material / W m^{-1} K^{-1}

Substituted into equation (4-2), this gives:

$$\phi_{chill} = VC_l \frac{dT_{ma}}{dt} = \frac{E}{3} \frac{V\beta_1^2 k_l}{X^2} (T_a - T_{ma}) \qquad (4\text{-}11)$$

where: C_l is the volumetric specific heat capacity of the unfrozen material / J m^{-3} K^{-1}

This is not an exact solution, although the importance of the implied assumption that $T_s = T_{ma}$ is considerably reduced compared with equation (4-2). A corollary of equation (4-10) implies that:

$$\frac{6\ Bi^2}{\beta_1^2 \left[\beta_1^2 + Bi\ (Bi-1)\right]} = 1 \qquad (4\text{-}12)$$

This is true with less than 10% error for $Bi < 3.5$, indicating that reasonable heat load estimates may be expected from equation (4-11) for Biot number values commonly found in food chilling and freezing.

E is calculated by the method of McNabb *et al* (1990*a*,*b*). The McNabb method is strictly derived only for the phase change case and E values for chilling may be somewhat different from those for freezing. In the absence of better information, the phase change E values are used for both chilling and freezing. Experience has shown that freezing food products are generally best represented by equivalent ellipsoids when E is found using the McNabb method. Geometrically, the sphere shape is a special case of the ellipsoid so the sphere was considered to be a better reference shape for the purpose of food product chilling heat load prediction than either the slab or the cylinder.

As a result of the weaknesses described above, the heat load predicted by this method will underestimate the actual load at the start of the chilling process, and this error will increase as $Bi \rightarrow \infty$.

4.6 Freezing model

The freezing stage of the product heat load model has two alternative formulations which were developed independently and in parallel by the present author and Pham (1989) respectively. The work of Pham has only been reported separately in a personal communication to the author, so that document (Pham, 1989) is reproduced in this thesis as Appendix 1.

The model used for the freezing stage was developed from the following assumptions, most of which hold much better for freezing than for chilling:

1 Sensible heat effects are negligible or can be bundled with the latent heat effects.

2 The body is homogeneous.

3 It is possible to treat heat transfer at the surface of the body as varying linearly with the difference between the body surface and ambient temperatures.

4 The cross-sectional area through which heat flows within the frozen region from the freezing front to the surface is related to the distance from the product centre by equation (4-13).

$$A_x \propto x^n \tag{4-13}$$

where n corresponds with that of equation (4-7) and A_x is the cross sectional area at the distance x from the centre. When n = 0, 1, or 2, equation (4-13) describes the cases of infinite slabs, infinite cylinders and spheres respectively. For other shapes, equation (4-13) approximates the three-dimensional conduction problem using one dimension, thereby allowing the conduction problem to be solved without the need for partial differential equations.

An energy balance at the freezing front requires that the rate at which heat is conducted away equals the rate at which that heat is released at the front, hence:

$$k_s \left(\frac{dT}{dx}\right)_{x=x_f} = L \left(\frac{dx_f}{dt}\right) \tag{4-14}$$

$$\Rightarrow \frac{dx_f}{dt} = \frac{k_s}{L} \left(\frac{dT}{dx}\right)_{x=x_f} \tag{4-15}$$

The heat release rate at the surface equals the rate at which heat flows through the frozen region:

$$hA_s(T_a - T_s) = k_s A_x \left(\frac{dT}{dx}\right) \tag{4-16}$$

Substituting the relationship in equation (4-13):

$$\Rightarrow \frac{dT}{dx} = \frac{h}{k_s} X^n x^{-n} (T_a - T_s) \tag{4-17}$$

$$\Rightarrow \int_{T_s}^{T_f} dT = \frac{h}{k_s} X^n (T_a - T_s) \int_{x}^{x_f} x^{-n} dx \tag{4-18}$$

Assuming that n ≠ 1, integrating equation (4-18) gives equation (4-19):

$$T_s = \frac{(1-n)T_f - (h/k_s)X^n\left(x_f^{1-n} - X^{1-n}\right)T_a}{(1-n) - (h/k_s)X^n\left(x_f^{1-n} - X^{1-n}\right)} \tag{4-19}$$

Substituting equation (4-19) into equation (4-17) and thence into equation (4-15) gives equation (4-20).

$$\Rightarrow \frac{dx_f}{dt} = \frac{T_a - T_f}{L x_f^n \left[\frac{1}{hX^n} - \frac{\left(x_f^{1-n} - X^{1-n}\right)}{k_s(1-n)}\right]} \tag{4-20}$$

This equation can be regarded as a replacement for equation (4-3), with the advantages of greater simplicity and no need for a reference shape. To estimate the freezing heat load from this equation, it is necessary to find an expression for the rate of change in frozen volume with x_f.

It was initially speculated that the rate of change in frozen volume with x_f may be some function of the shape factor n, but testing of several alternative forms for this function showed this approach to be useful only at low values of the Biot number. In a personal communication to the author (reproduced in Appendix 1), Pham (1989) suggested that the change in frozen volume with x_f may be a function of a shape factor N which is different from n.

If it is assumed that the volumetric unfrozen fraction is related to the linear unfrozen fraction by equation (4-21),

$$\frac{V_f}{V} = \left(\frac{x_f}{X}\right)^N \tag{4-21}$$

then differentiating equation (4-21) obtains equation (4-22).

$$\frac{dV}{dx_f} = N\left(\frac{x_f}{X}\right)^{N-1}\frac{V}{X} \tag{4-22}$$

And so the freezing heat load at any given time, t, may be estimated by:

$$\phi_{freeze} = V\frac{d\hat{H}}{dt} = L\frac{dx_f}{dt}\frac{dV}{dx_f} \tag{4-23}$$

where: $\hat{H}$ is the volumetric specific enthalpy / J m^{-3}

L is the latent heat of freezing / J m^{-3}

The problem of evaluating N is considered below.

4.6.1 Product enthalpy in freezing and sub-cooling

For food products, the freezing front is not sharp, and not all of the heat is released at T_f, so when equation (4-20) is integrated numerically, experience has shown that much better results are achieved by replacing T_f with the mass average temperature, T_{ma}. T_{ma} is re-calculated at each timestep from the commonly used product enthalpy function (Schwartzberg, 1976):

$$\hat{H} = a + bT_{ma} + \frac{c}{T_{ma}} \tag{4-24}$$

where: a,b,c are constants (see below).

Equation (4-24) can be solved for T_{ma} to obtain equation (4-25):

$$T_{ma} = \frac{(\hat{H} - a) - \sqrt{(\hat{H} - a)^2 - 4bc}}{2b} \tag{4-25}$$

The value of $\hat{H}$ can be obtained at any time in the process from solving equation (4-23). The coefficients a, b and c are given for many food products by Pham (1987*a*), or they may be calculated by solving equation (4-24) at three points. For example, data for enthalpy $\hat{H}_f$ at the freezing temperature T_f, enthalpy $\hat{H}_{base}$ at the enthalpy base temperature T_{base} (usually -40°C), and the frozen heat capacity C_s at T_{base} allow one to evaluate the following equations:

$$c = \frac{\hat{H}_f - C_s(T_f - T_{base})}{\frac{T_f - T_{base}}{T_{base}^2} + \left(\frac{1}{T_f} - \frac{1}{T_{base}}\right)} \tag{4-26}$$

$$b = \frac{\hat{H}_f - \frac{c}{T_f} + \frac{c}{T_{base}}}{T_f - T_{base}} \tag{4-27}$$

$$a = -bT_{base} - \frac{c}{T_{base}} \tag{4-28}$$

4.7 Sub-cooling model

This model is similar to the chilling model, except that frozen material properties are substituted for the unfrozen material properties and T_{ma} is calculated using the enthalpy function described above so that the remaining latent heat is included in the calculation. The model which results from these modifications is described by:

$$\phi_{subcool} = V\frac{d\hat{H}}{dt} = \frac{E}{3}\frac{V\beta_1^2 k_s}{X^2}(T_a - T_{ma}) \tag{4-29}$$

T_{ma} is re-calculated from $\hat{H}$ at each time step using equation (4-25).

The chilling model has two weaknesses when it is applied to sub-cooling. Firstly, the presence of residual latent heat means that the notional heat capacity of the body changes during the sub-cooling period. This violates the assumption of constant heat capacity made in deriving the analytical series solution for chilling under constant conditions. One might expect the sub-cooling heat load to be predicted poorly until the latent heat component becomes negligible.

Secondly, the presence of the freezing front within the body during the first part of the sub-cooling period causes the shape of unfrozen region to continue changing, with the corollary that the shape of the frozen region will also change. This violates the assumption that the E geometry factor is constant during sub-cooling — a problem that does not occur during freezing as the freezing front motion is handled separately from the heat load. Again, one would expect this problem to decrease in importance during the latter part of the sub-cooling stage.

4.8 Criteria for transition between models

There is no theoretical basis for deciding when to move from one model to the next during a numerical integration of the heat load estimation method. A number of criteria were considered:

1 Move from chill to freeze when the mass average temperature of the body drops to T_f.

2 Move from freeze to sub-cool when the mass average temperature drops to some point at which freezing is considered to be complete (e.g. -10°C).

3 Move from the current model to the next when the next model produces a higher heat load than the current one.

4 Move from the current model to the next when the rate of change of heat release in the current model is equal to the rate of change of heat release in the next model.

Of these, criteria 1 and 2 are physically reasonable only at low Biot numbers. At higher Biot numbers, they generate discontinuities in the predicted heat load profile. Criterion 4 provides a continuous first derivative for the heat load result, but the equality condition is difficult to implement successfully in a computer algorithm. Criterion 3 provides a heat load result without discontinuities (although transitions may not be smooth, i.e. the first derivative is discontinuous), and realistically sites the transition points depending upon the Biot number for the process.

Criterion 3 was chosen as being the most appropriate for the chill→freeze transition. Criterion 3 was also chosen for the freeze→sub-cool transition, but some difficulties remained in this case. Firstly, the transition from freeze to sub-cool is made before the freezing front has reached the centre of the product. This is desirable as it avoids the point where $x_f = 0$, under which circumstances the freezing front moves at a nominally infinite rate for all shapes but the slab. It does introduce the problem, however, that when the sub-cooling stage commences there is still some latent heat remaining in the body.

Secondly, criterion 3 alone is inadequate at high Biot numbers, and it causes a transition to sub-cooling when there is still a substantial latent heat load, despite that fact that the sub-cooling model is physically unrealistic when the latent heat load is still dominant. In practice, the overall method has been found to perform much better if transition to sub-cooling is not allowed until the freezing object is at least 80% frozen by volume, i.e.:

$$\left(\frac{x_f}{X}\right)^{n+1} < 0.2 \qquad (4\text{-}30)$$

The factor $\beta_1^2 k_s$ in equation (4-29) is re-calculated at the time of transition to make the initial sub-cooling stage heat load equal to the final freezing stage heat load. This is a rather crude correction, but high accuracy at this point in the process is not usually critical because the total heat load is quite small.

4.9 Evaluation of freezing equation parameters

Two parameters are used by the freezing equation to define the geometry of the freezing body. The parameter n (which describes the temperature profile curvature at the phase change front) is not new, but rather, $n = E\text{-}1$, so n may be evaluated by the same methods as are used for E (Hossain *et al*, 1992*a*,*b*).

The second geometry parameter, N, describes the way in which unfrozen volume changes with freezing front position. At the start of freezing, the shape of the unfrozen region is the same as the shape of the outer surface, thus it can be seen from equation (4-21) that N = 1, 2, 3 for the slab, cylinder, and sphere cases respectively. For all shapes, N initially follows the expression:

$$N = \frac{AX}{V} \qquad (4\text{-}31)$$

When $Bi = 0$, $E = A\ X\ /\ V$, so $N = E$ under these conditions. For $Bi > 0$ and any shape but the slab, cylinder or sphere, the value of N may be expected to change during any process as the shape of the unfrozen region changes by becoming more rounded. Thus, an accurate estimate of N at any given time would depend upon the relationship between the whole body shape and that of the remaining unfrozen region. Evaluating N exactly would therefore require a great deal of data for the more irregular shapes as well as considerable computation. It could be postulated that N might change from the initial value given by equation (4-31) downwards, perhaps reaching E at the completion of freezing, but the mathematical function relating N to frozen volume may be very complicated. To simplify the problem, it may be hoped that there is some average N for any given shape that is adequately close to the real (changing) N during the period of significant heat release. In the absence of other data, equation (4-31) was used as an approximation to this average N.

The latent heat of freezing, L, is not well defined for food products. For the purposes of this heat load prediction method, L may be conveniently defined by:

$$L = \hat{H}_f - C_s (T_f - T_{base}) \tag{4-32}$$

If freezing starts when T_{ma} is still above T_f, the overall energy balance can be maintained if L is defined so that it includes the residual superheat:

$$L = \hat{H}_f + C_l (T_{ma} - T_f) - C_s (T_f - T_{base}) \tag{4-33}$$

4.10 Practical implementation

Accurate estimation of the sub-cooling heat load is strongly dependent upon a correct prediction of the mass average temperature, T_{ma}. This can cause some difficulty as the predicted temperature is very sensitive to the accuracy of equation (4-25). Fortunately, the heat load is small during this stage, and accurate estimation is less important than it is earlier in the process.

When solving the ordinary differential equations numerically, care must be taken at the transitions between chilling and phase change, between phase change and sub-cooling, and when the freezing front is close to the centre of the object. Some of the less sophisticated numerical ODE solution methods may have difficulty at these points due to discontinuities and singularities.

There is a singularity in equation (4-20) for $n = 1$. It is not worth deriving the results for this special case, however, as one may simply use some value of n as close to 1 as the precision of one's computer allows and expect that the predicted heat loads will be negligibly different from the special case itself.

4.11 Extension to non-food materials

While the model above was specifically developed for predicting the cooling heat load of food product, much of the derivation is quite general and did not require

any assumptions regarding the composition of the cooling material. In particular, the above model could be used to characterise already-frozen food product and the room structural materials (such as steel or concrete) within a refrigeration application by starting the material in the sub-cooling stage rather than in the chilling stage. One difficulty was encountered with the temperature-enthalpy function when this was done.

The hyperbolic function used above to describe the variation of enthalpy with mass-average temperature contains an implicit parameter, T_{ff}, which expresses the freezing temperature of the pure substance within the material which crystallizes during the freezing process. In the case of food materials the crystallizing substance is water (T_{ff} = 0°C) and so T_{ff} was ignored, but this may not be the case for non-food materials. The addition of T_{ff} to equation (4-25) results in equation (4-34):

$$T_{ma} = \frac{(H-a) - \sqrt{(H-a)^2 - 4bc}}{2b} + T_{ff} \qquad (4\text{-}34)$$

This extension to the heat load model made it unnecessary to develop a separate model to represent the heat load due to structural materials within applications. The heat load model of this chapter when applied to structures should also be more accurate than previously-developed structural material models because it includes compensation for the effect of Biot number on the interior temperature profile. For example, the structural material model of Cleland (1985*a*) was identified by Cornelius (1991) as being of the same form as equation (4-2).

For most non-food materials, it is not necessary to provide the actual value of T_{ff}. For some materials (e.g. concrete), there may be no real T_{ff} value at all. The most important criterion in these cases is to set T_{ff} above the range of temperatures which may be expected in the application so that simulated structural materials cannot "melt". The most simple form of the heat load equation parameters for use with structural materials is as follows:

C_s Heat capacity of the structural material / J m^{-3} K^{-1}

T_{base} = -40°C

$\hat{H}_{base}$ = 0 J m^{-3}

T_f = 99°C (well above the expected operating temperature range)

T_{ff} = 100°C (well above the expected operating temperature range)

$\hat{H}_f$ = C_s (T_f - T_{base})

If equations (4-26) to (4-28) are used to evaluate the enthalpy equation parameters, then T_{ff} should not be equal to either T_f or T_{base} to avoid division by zero.

This parameter set assumes that the specific heat capacity of the structural material can be adequately represented by a constant value. If the heat capacity was known to vary significantly with temperature, then $\hat{H}_f$ and T_f could be given values which allow the enthalpy function to represent this variation. The specific heat capacities of the structural materials modelled in the simulations of Chapter 10 did not vary substantially with temperature so this has not been done in the present work.

Chapter 5: Food Product Heat Load Measurement

Validation of the food product heat load model developed in Chapter 4 required data against which the model could be tested. An indirect approach was described by Cleland (1985*a*, 1986*b*), who used a full plant simulation to test his heat load model. In that case the product heat load models comprised only about 10% of the dynamic models in the simulation. All of these models were subject to error, and most of them interacted with the others to some greater or lesser degree. Product heat load was not measured explicitly, so the accuracy of the product heat load model could only be inferred from its effects upon secondary values such as the ambient air temperature in each refrigerated room. This limited the extent to which the conclusions drawn from this validation method could be relied upon.

More confidence could be placed in a validation procedure which tested the model against actual experimental measurements of product heat load. This presented the problem of measuring the heat load of cooling food product. The product types of most interest to this project were two principal exports of the New Zealand meat industry: frozen whole lamb and sheep carcasses (with weights up to 30 kg) and frozen boned meat cartons (of about 27 kg each).

The following methods of measuring the heat load vs. time profile of cooling food product have previously been reported in a manuscript by Lovatt *et al* (1992*b*) which is reproduced in Appendix 4.2.

5.1 Development of methods for measuring product heat load

Three techniques for measuring food cooling heat load were attempted.

5.1.1 Industrial freezer air cooler load measurement

The principle of this method was to measure the performance of one freezer in detail rather than attempting to measure overall plant performance in the fashion

of Cleland (1985*a*). This would allow a much more sensitive evaluation of the product heat load model than a full plant simulation. The minimum requirement was to measure the cooler air flow rate before or after the freezing process, and to monitor the air temperature change over the air cooler during the process using differential temperature probes. Heat infiltration could be estimated by conventional means (Cleland, 1990, p.214), and the remaining heat load would be due to items within the room.

It was hoped that if a batch freezer were instrumented in this fashion then the total heat load resulting from the product freezing in the room could be calculated. There are a number of problems which restrict the quality of any results obtained:

1 There are other heat loads in a typical freezer room besides the product. In particular, the effect of structural loads will distort the measured load (the author's experience suggests that an initially warm concrete floor may add 20% or more to the total load in a typical New Zealand batch freezer). Steady state loads such as forced convection evaporator fans may be accounted for by measuring electrical energy input, but time-variable loads such as those due to air interchange are difficult to estimate accurately.

2 Differential temperature measurements record sensible loads only so it is not practical to use this method for product where there is a significant latent load due to evaporation from the product.

3 Refrigeration conditions in the room may not remain consistent during the freezing process. Refrigerant evaporating temperatures may fluctuate considerably over the freezing period due to interactions with the rest of the plant, and in a batch freezer the air temperature will often fall during much of the process as the refrigeration load drops (Cleland, 1985*a*).

4 If the refrigerant or air flow is varied to regulate air temperature, the measured instantaneous load may not bear any resemblance to the real load under constant conditions.

When preliminary trials of this method were undertaken, all of the problems discussed above were identified and the method was found to be insufficiently precise. Errors in the energy balance were typically in the range 50-100%, and there was no indication that this level of error could be substantially reduced without a radical change of approach.

While this method was found to be impractical for product heat load model validation, it was found that for some operating conditions the measured heat load did provide some useful information on the performance of the individual room. If the differential temperature measurement was properly calibrated for the freezing conditions, it could be a useful source of data for a refrigeration plant strategy controller.

5.1.2 Electrical heat make-up

In principle, product heat load measurement required a form of calorimeter. The product and process were not suitable for placing in a conventional calorimeter, so the principles of calorimetric operation were extended to an existing controlled-environment tunnel (as shown in Figure 5-1) which was converted for the purpose.

It was intended to supply a constant refrigeration effect to the freezing tunnel throughout the process and to control an electrical heater in such a way that the temperature of the air passing over the product was held constant. If this was done, then the heat load of the product at any given time would be equal to the difference between the electrical power input to the heater and the power required to maintain the same air temperature in the absence of product. That is:

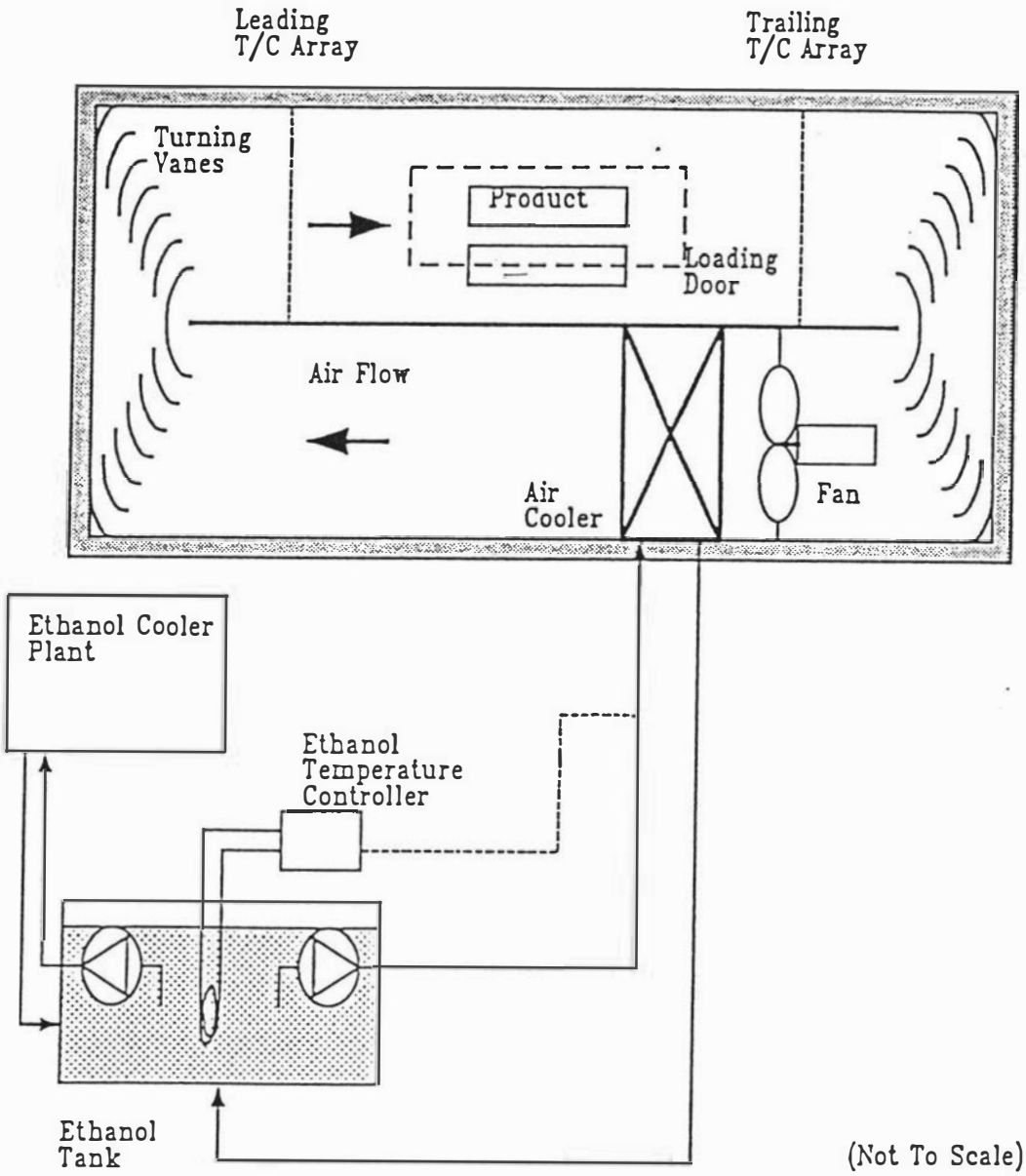


Figure 5-1 Diagram of the MIRINZ environmental tunnel as set up for product heat load measurement.

$$\phi_p = \phi_{ess} - \phi_e \qquad (5\text{-}1)$$

where : ϕ_p is the heat flow from the product at a given time / W

ϕ_e is the electrical heater energy flow at a given time / W

ϕ_{ess} is the electrical heater energy flow at steady state / W

This followed the method of James and Bailey (1981) who applied a similar method to measure the heat load while chilling pork sides. When this technique was tried, there were found to be a number of problems:

1 It was not practical to use alternating current to supply the heat because the regulation of the heater modified the AC waveform, thus making accurate power measurement prohibitively expensive. This mandated the use of a direct current heater, which in turn limited the capacity of the electrical heater to the capacity of the available DC source (about 650W).

2 It was very time-consuming to adjust the plant to the point where the heater was near, but below, maximum output and therefore had enough range to cope with the maximum load without dropping to zero output.

3 At the start of the freezing process, the peak load resulting from the warm product and the air which had entered with the product exceeded the maximum capacity of the electrical heater, and so the air temperature became temporarily uncontrolled. This had a deleterious effect on the estimated heat load.

4 It was very difficult to keep the refrigeration effect constant. In the experimental plant, refrigeration was supplied as chilled ethanol which circulated through the air cooler. The ethanol temperature was controlled to within 0.1°C of its setpoint, but even this small variation had a 2% effect on the supplied refrigeration effect due to changes in the logarithmic mean temperature difference. In turn, this resulted in larger percentage changes in the heater power (5% of maximum

output) and even larger changes in the calculated heat load (10-15% of the mean freezing product heat load). This noise obscured much of the useful data.

5 The baseline electrical heat input, ϕ_{el}, was subject to significant shifts when the door was opened to load and unload product. Heat infiltration around the door was affected by the way in which the door seated and this varied from closing to closing.

6 The large surface area of the environmental tunnel made heat infiltration an important factor. Runs had to be adjusted for changing heat infiltration due to the changing external temperature over the length of the run (12 to 60 hours depending on product and freezing conditions).

7 The heater power measured in this experiment was the output of the controller rather than the controlled variable. Whilst the air temperature was well controlled except immediately after starting a freezing run, the heater power output by the controller was subject to considerable noise. De-tuning the controller helped reduce the noise level, but some noise remained due to this source

In spite of these difficulties, some valid results were achieved through the use of this method although data smoothing was required to make the results intelligible.

It was necessary to estimate the refrigeration effect and include it in the heat load calculation because of the extent of its variation. This was done by measuring the change in temperature of the ethanol flow across the air cooler with a differential thermocouple, and its volumetric flow rate with a turbine flow meter.

The start of each run was inaccurate due to starting effects, while the last half of each run had a heat load which was within the range of the data noise. Results could only be relied upon for the period after start-up and before the heat load dropped below about 40W. This was not acceptable for the purpose of validating the heat load model. It was concluded that the method could only be made satisfactory

if a purpose-designed experimental plant were constructed — an exercise which would have required expenditure beyond the budget of the project.

5.1.3 Differential air temperature measurement

In an attempt to find a method which was less sensitive to external influences, the principle of the flow calorimeter was adapted to product heat load measurement.

The environmental tunnel described in section 5.1.2 was retained, but a thermocouple array was placed before and after the product in the air stream as shown in Figure 5-1. The array comprised 16 differential T-type thermocouples combined together in series and spaced evenly across the cross-section of the tunnel with 16 junctions before the product and 16 after the product. An anemometer traverse was carried out to measure the air flow rate in the working section of the tunnel so that the sensible product heat load could be calculated from equation (5-2).

$$\phi_p = C_{air}\, Q_{air}\, \Delta T_{air} \qquad (5\text{-}2)$$

where:

C_{air} is the volumetric specific heat capacity of air / J kg^{-1} K^{-1}

Q_{air} is the volumetric flow rate of air / m^3 s^{-1}

ΔT_{air} is the change in temperature of the air passing over the product / K

A number of advantages ensued:

1 With the thermocouple arrays just each side of the product, the amount of heat infiltration through the walls of the tunnel which was included in the raw heat load data was reduced by a factor of about 10 compared with the whole tunnel surface infiltration.

2 There was now no need for the tunnel air temperature to be held exactly constant. The previous tight air temperature control (used in the method of section 5.1.2) was therefore discontinued.

3 With the improved air temperature tolerance, it was no longer necessary to include the refrigeration effect in the heat load calculations, and therefore a large source of noise was removed from the data.

There remained two principal disadvantages with this approach:

1 Only sensible heat could be measured. For this reason, a load cell was installed to monitor product weight during processes where latent heat was important. There was some difficulty in making that system function in a repeatable manner, so estimates of the latent heat contributions are not included in the results shown in Chapter 6. Most of the experimental runs were carried out with the product enclosed in a vapour-tight wrapping to minimise any latent heat load as the product heat load model described in Chapter 4 did not explicitly take latent heat load into account.

2 There was some difficulty in accurately measuring the air flow rate. It was not possible to carry out a complete traverse across the tunnel because there were only three points in the tunnel ceiling where the anemometer could be inserted. Fortunately the data collected showed that there was little horizontal variation in air flow rate across the width of the tunnel. There was some variation in air flow rate with vertical position, but this was taken into account by the anemometer traverse.

Given the advantages of this method and the acceptability of its disadvantages, the method of this section was chosen as the best technique for producing repeatable product heat load profiles.

5.2 Materials and preparation

5.2.1 Carton Shape

The first process to be studied was the freezing of boned meat cartons in a moderate air velocity environment, as occurs in many New Zealand meat processing plants. To make the process more convenient to simulate with finite difference methods, the meat analogue Tylose MH1000 was used instead of boneless beef or lamb. The multi-layer cardboard boxes which are used in the industry were also difficult to accurately quantify in a finite difference scheme, so open-topped light-gauge galvanised steel boxes were used to hold the Tylose. The boxes were 335mm wide by 510mm long, and they were filled with Tylose to a depth of 154mm, making each essentially the same shape and volume as a typical export meat carton. Two cartons were used in each run. A diagram of the two cartons is shown in Figure 5-2.

The Tylose was made up using the hot water method (described by Riedel, 1960) in which the Tylose powder was mixed with hot water prior to placing it in the boxes. Tylose made in this way was slower to set than Tylose made with cold water and the slower setting resulted in easier mixing, a smoother surface, and a negligible amount of dry powder in the finished product when compared with Tylose made with cold water. The only additive used was some 5g of copper sulphate per carton to retard the growth of mould and bacteria. No sodium chloride was added.

5.2.2 Lamb and Sheep Carcasses

The frozen lamb or sheep carcass is another significant product of the New Zealand meat industry. It was important to find the best way of representing such an irregularly-shaped product using the ODE heat load prediction method.

A variety of lamb and sheep carcasses were frozen at different air velocities with several of the different wrappings used in the industry.

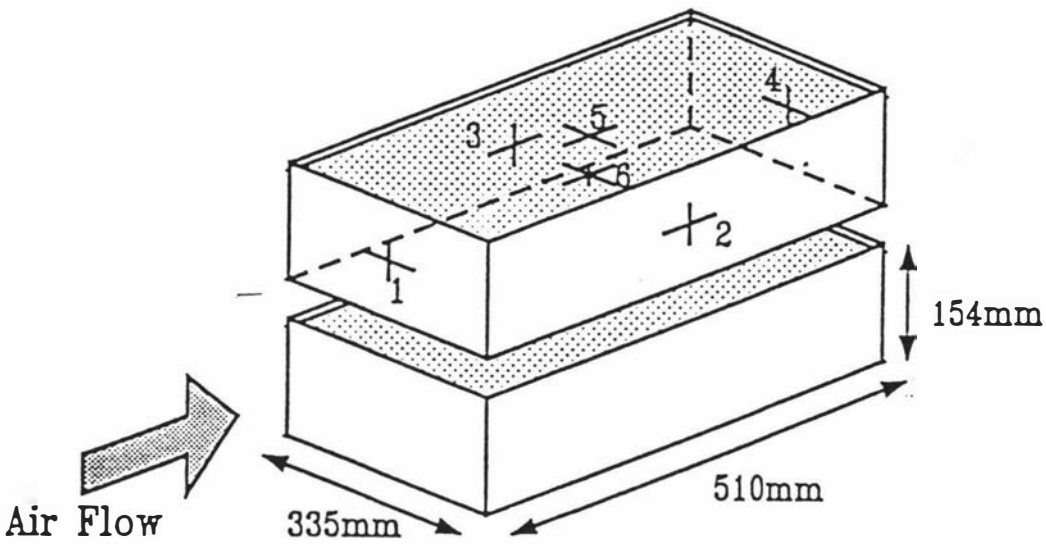


Figure 5-2 Diagram of Tylose-filled meat carton shapes as arranged for product heat load measurement. Thermocouple positions are numbered.

5.3 Experimental method

The tunnel was cooled from room temperature to the working temperature and then run until it equilibrated at the experimental conditions. Equilibration required several days running at the operating temperature as heat load measurements using the method described in section 5.1.2 had indicated that the empty tunnel contained a heat load which declined slowly after starting the tunnel from room temperature. It was speculated that this slowly-declining heat load was due to moisture gradually freezing within the tunnel insulation. The tunnel was run for several days at a temperature of about 35°C in an attempt to evaporate any moisture in the insulation, and this procedure may have succeeded as equilibration times were shorter

afterwards. For many runs, the tunnel was held at its operating temperature after the previous run so the equilibration problem was not encountered.

Meanwhile, the product to be frozen was allowed to equilibrate in a controlled-temperature room at 8°C. The Tylose cartons were left until their temperatures were negligibly different from the controlled room temperature, as monitored by a thermocouple probe in the centre of one of the two cartons. Two cartons were used in order to maintain sufficient heat load for accurate measurement during the experiment. Thermocouples were inserted in the centre and five surfaces of one carton, as shown in Figure 5-2.

The lamb and sheep carcasses were processed to ensure that they were hygienic, and met the New Zealand Accelerated Conditioning and Aging (AC&A) specification for high-quality frozen meat. The lambs and sheep were killed early in the morning, the carcasses were electrically stimulated within 30 minutes of slaughter, then covered with the wrapping specified for that run and placed in the 8°C controlled-temperature room for about 6.5 hours. The highest temperature in the carcasses was then typically about 18°C.

Before loading into the tunnel, each carcass was weighed and six thermocouples were inserted in various locations, always including at least one deep leg and one deep shoulder position. The 0.12 mm type T thermocouples used in the experiment were not very durable and there was typically one broken thermocouple in each heat load measurement run. The six thermocouple pairs were all brought together to a male 25-pin electronic connector which could be quickly mated to a female connector installed in the tunnel.

Each carcass was wired to a stainless steel rod in a configuration similar to the way in which it would hang in the air stream of a typical New Zealand vertical air flow blast freezer. The rod was then hung horizontally from a wire sling to support it in the tunnel. By this means the experiment was kept directly comparable to industrial freezing practice while still fitting a whole carcass into the working section of the tunnel.

Both product types were loaded into the tunnel in the same manner. The tunnel fan was stopped, the side loading door opened, and the product lifted inside. Carcasses were hung on their sling from a double hook in the ceiling of the working section with hind legs towards the airflow. Cartons were placed on metal supports which allowed the air to pass along all surfaces. The carton with the temperature probes inserted was placed on top of the other one, separated from it by similar supports. The six thermocouple probes were plugged into the data logging system, the side door closed, and the fan restarted. The loading operation usually took about 30 seconds, but never more than 60 seconds.

All of the runs were conducted with a nominal ambient air temperature of approximately -21°C but the air temperature was not closely controlled and therefore tended to drop from about -19.5°C to -21.5°C over the course of each run.

Three carcass runs (numbers 6 to 8) were carried out at an average air velocity of 0.35 m/s, but the energy balances for these runs were found to be very poor. It was speculated that the air flow distribution across the tunnel was uneven at such a low air velocity, with a consequent detrimental effect on heat load measurement accuracy. All other runs (carton runs 1 to 3 and carcass runs 1,2,3,4,5 and 9) were conducted at an average air velocity of 1.42 m/s. Most runs were halted when the lowest measured product temperature was within 1°C of the tunnel air temperature to make baseline heat load estimation easier. Some runs were terminated early due to mechanical problems (notably carton run 3).

5.4 Air flow rate measurement

The major area of potential error in the heat load measured by the differential temperature method lay in the measurement of air flow rate in the tunnel. Measurement during an actual run was impractical due to the effect on heat load measurement accuracy of warm air infiltrating through the anemometer access holes, so the air flow rate was measured when there was no run taking place.

Initial air velocity measurements were conducted with the tunnel at room temperature, but an early analysis indicated that the total heat load was being overestimated by a considerable fraction so the tunnel air velocity was re-measured at the operating temperature (-21°C). This was done even though the vane anemometer used for the purpose was not rated as accurate below 0°C (no anemometer available to the author at the time was guaranteed at the temperature in question). Air velocity measurement inaccuracy was minimized by only holding the anemometer in the cold air flow for a few seconds at a time — just long enough to obtain a consistent reading.

Air mass flow rate was calculated from the mean of the measured air velocities, the air density, and the cross-sectional area of the tunnel. While the mass flow rate would have been slightly different due to the increased pressure drop when the product was in the tunnel, the pressure drop caused by the product was expected to be negligible compared with the pressure drop within the air cooler and that caused by the direction changes as the air passed around the tunnel.

5.5 Logged differential thermocouple array output

An example of the raw output from the differential air temperature thermocouple array (for carton run 1) is shown in Figure 5-3. The way in which these data were processed to produce a heat load vs. time profile for each experimental run is described in Chapter 6.

5.6 Conclusions

The differential air temperature method for measuring the heat load vs. time profile of freezing food product was found to be the most successful of the three

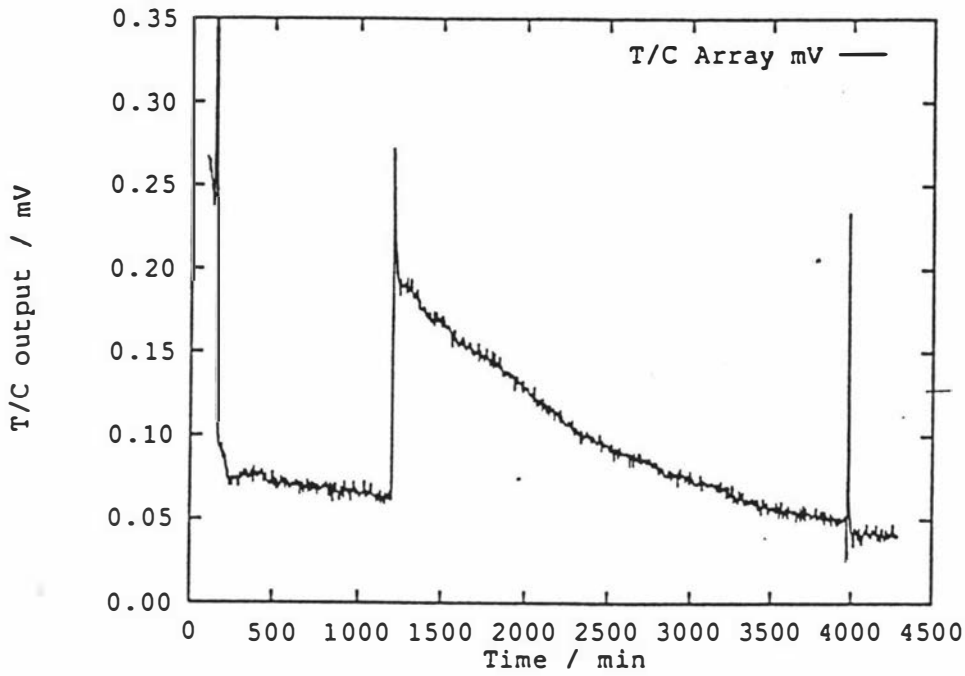


Figure 5-3 Sample output from the differential air temperature arrays. Spikes at 1200 and 4000 minutes indicate loading and unloading respectively

techniques attempted. It was more accurate than measuring the temperature change across an industrial evaporator and the results were easier to process than those of the heat make-up method. The differential air temperature method also required a lower standard of control over the experimental conditions than did the heat make-up method.

Chapter 6: Heat Load Measurement Analysis and Model Testing

A new model for predicting the heat load profile of cooling food product was developed in Chapter 4. Chapter 5 described three methods which were attempted for measuring product heat load. Chapter 5 also outlined the experimental procedure which was followed to obtain heat load vs. time profiles for freezing carton shapes and lamb carcasses using the differential air temperature measurement technique. Chapter 6 reports the results of analyzing the experimental measurements and of validating the ODE heat load model against both the experimental measurements and finite difference heat load predictions.

Much of this analysis has previously been reported in manuscripts by Lovatt *et al* (1992*a*,*b*) which are reproduced in this thesis as Appendices 4.1 and 4.2.

6.1 Heat load baseline estimation

An example of the raw output from the differential air temperature thermocouple array (for carton run 1) is shown in Figure 5-3.

The differential thermocouple heat load measurement technique could not measure the absolute sensible heat load of the product. Heat infiltration around the working section of the environmental tunnel significantly affected the measured air temperature change and the extent of this infiltration load varied from run to run due to differences in the quality of the loading door seal after each closing. Thus it was necessary to estimate the base level of heat infiltration within the working section during each run.

The baseline estimation approach assumed that the heat load at the end of the run was a good estimate of the baseline. One early run showed that this was superior to assuming that the baseline was estimated by the heat load measured immediately prior to loading the product because the heat load at the end of this run was much lower than the heat load before the start. For runs which were cut short

for any reason, the baseline was assessed as being the lowest heat load measured from several hours before the start of the run (when logging was started) until the run finished. To reduce the influence of random error, the measured loads were averaged over 20 minute periods for baseline estimation purposes.

The baseline estimate accuracy could be assessed by checking the energy balance for the experiment — since the starting and finishing temperatures, and the mass of the product were all known, the theoretical change in enthalpy between the two temperatures could be estimated from product thermal properties.

6.2 Carton shapes

6.2.1 Finite difference calculations

The cartons used for experimental heat load measurement were of a sufficiently regular shape that they could be modelled using finite difference methods. Finite difference calculations have been found to predict freezing times within the bounds of experimental freezing time measurement accuracy (Cleland, 1990, Chapter 4) and, by the nature of the finite difference methods, they may be expected to predict heat loads to a comparable degree of accuracy. If finite difference methods could be used to test the ODE method for regular shapes then this would greatly reduce the number of experimental measurement runs required to gain confidence in the ODE method. The first objective of the heat load measurement experiments was therefore to verify the accuracy of finite difference methods when predicting product heat load.

A finite difference calculation was carried out for the conditions encountered in the experiment using the FINDIFF program supplied as part of the RADS package (Cornelius, 1991). This program implemented the Lees scheme described by Cleland (1990, Chapter 4), and the initial FD calculation used a grid of 10×10×8 cells (11×11×9 nodes).

The simulated carton shape was started at the same uniform temperature as the experimental carton (8°C). The heat transfer coefficient used in the FD run was back-calculated to give the same freezing time (to -15°C centre temperature) as was measured in the experimental case.

The Tylose MH1000 used in this experiment had an initial freezing temperature of -1.2°C rather than the expected -0.68°C (Pham, 1987*a*). The Tylose density was measured by fluid displacement to be 1050 kg m^{-3}, thermal conductivity was obtained from Pham (1990), and enthalpy vs. temperature data was obtained from unpublished experiments with an adiabatic calorimeter by the MIRINZ refrigeration group. The thermal conductivity data used in the FD runs are shown in Table 6-1 and the enthalpy vs. temperature data is shown in Table 6-2.

6.2.2 Experimental measurements

The experimentally-measured heat loads are shown together with various predictions in Figure 6-1. The FD calculation is plotted as the "Even HTC" line. Comparison of the predicted and experimentally-measured heat loads demonstrated that there was a problem with the overall energy balance. The change in enthalpy over the process for runs 1 and 2 (as estimated by numerically integrating the measured heat load) was 25% and 12% greater than the theoretical result. The measured enthalpy change for run 3 was 4% less than the theoretical result, but this run was shorter than the others due to equipment failure.

The shapes of the FD and experimental heat load profiles were also quite different.

The failure to obtain an energy balance and the different shapes of the experimental and finite difference heat load profiles must have been due either to error in the experiment, error in the finite difference calculation or some error in both. Both possible error sources were investigated.

Table 6-1 Thermal conductivity of Tylose MH1000

Temperature / °C	Thermal Conductivity / W/mK
-39.4	1.573
-38.4	1.597
-29.6	1.532
-19.7	1.437
-12.8	1.404
-8.9	1.381
-7.0	1.385
-4.4	1.296
-1.2*	1.200
-0.9*	0.481
3.2	0.481
14.9	0.479
27.7	0.511
37.6	0.530

These data were derived from Pham (1990).

*Data marked with an asterisk were added to the measured data to allow accurate linear interpolation by the FINDIFF program.

6.2.3 Sources of experimental error

1 There was some initial heat load due to opening the tunnel door when loading the product. The size of this load was checked by carrying out the normal loading operation, but without actually loading any product into the tunnel. Loading effects were seen to be insignificant within five to ten minutes of commencing the test run.

Table 6-2 Temperature vs. enthalpy data for Tylose MH1000

Temperature / °C	Enthalpy / kJ/kg	Heat Capacity / kJ/kgK
-38.93	1.49	1.89
-35.86	5.88	1.91
-28.09	21.20	2.13
-20.62	36.89	2.38
-13.56	54.18	3.36
-7.83	77.77	6.25
-4.04	110.86	13.56
-2.02	152.78	31.77
-1.09*	200.12	52.88
-1.08	200.12	515.73
-0.90*	293.44	515.73
-0.89*	293.44	3.99
40.71	452.95	4.04

These data were derived from unpublished work by the MIRINZ refrigeration group as reported by Pham (1991). Enthalpies were measured by calorimeter. Heat capacities were required for use by the FINDIFF program and were calculated from the measured enthalpies by numerical differentiation.

*Data marked with an asterisk were added to improve numerical differentiation accuracy.

2 The method used to estimate the heat load baseline was susceptible to some error. Since this error would have been different for each of the runs, it was checked by comparing the three experimental profiles with the smoothed mean of the three. The standard deviation of the experimental profiles about this mean was found to be 6.5W, which was about 5% of the mean product heat load. Error in the heat load

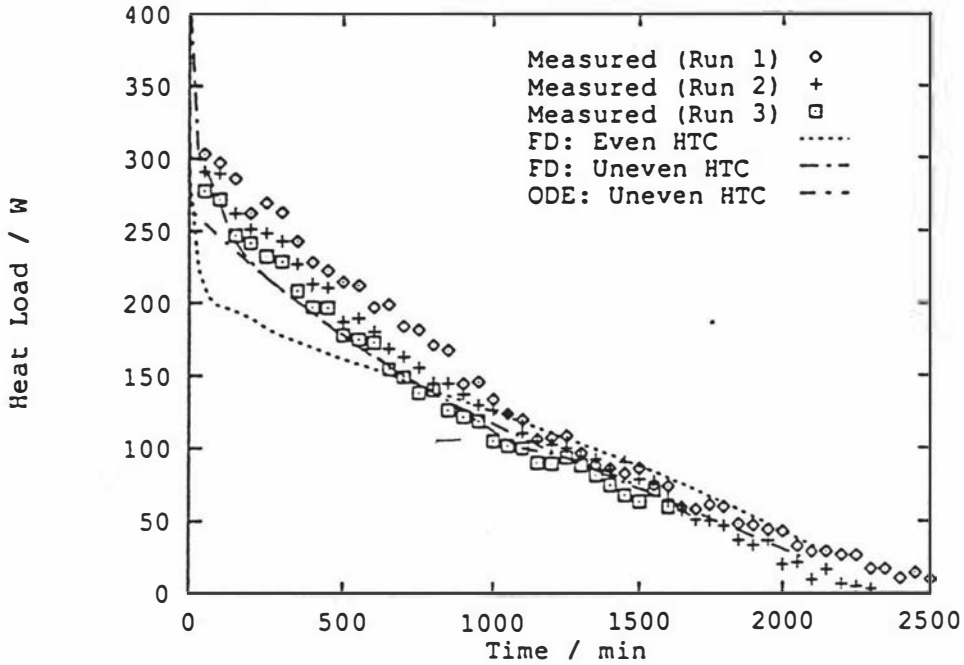


Figure 6-1 Single meat carton shape heat load as measured and as predicted by both FD and ODE methods.

baseline estimate would have contributed to the energy balance error but not to the difference between the heat load profile shapes.

3 The differential thermocouples may have been susceptible to radiation effects which could have caused the thermocouples to report higher temperatures than actually existed in the air. This was indeed a problem while commissioning the experimental plant, but it appeared to be resolved by shielding each of the thermocouples upstream of the product with aluminium foil, while still allowing air to flow around the probes. There may have been some residual trouble with radiation effects, but this was not expected to be a major source of error.

4 As discussed in section 5.4, the process of measuring the air mass flow rate in the tunnel was difficult and error-prone. A 10-15% error in the results of that procedure was quite conceivable, but it would affect the energy balance — not the shape of the heat load profile.

The energy balance error in each run was within the combined uncertainties of the air flow rate measurement method and the heat load baseline estimate. Improved precision may be possible, but air flow rate measurement precision may be expected to set a lower bound upon the overall accuracy of the experiment.

6.2.4 Sources of finite difference error

There were three possible sources of error in the finite difference heat load prediction. The finite difference computer program used for these calculations has been well tested over 15 years and was considered to be reliable, but its input data may not have accurately represented the reality of the experiment.

1 One area of possible error was in the thermal properties used for the calculation. An error in the estimated enthalpy change may explain some of the energy balance error, but this was not expected to contribute more than about 2% error.

2 The radiation effects discussed with reference to the differential thermocouples were not considered in the finite difference calculation. The effective heat transfer coefficient used in the finite difference calculation included a radiation component which treated radiation as a pseudo-convective process. The error introduced by not treating radiation in the correct manner (proportional to the difference between the fourth powers of surface and ambient temperatures) was checked and found to be no more than about 4%.

3 The final assumption made for the finite difference run was that the heat transfer coefficient was the same on all parts of all surfaces. That this was not so was indicated by the plot of measured temperatures in the carton during the process. Figure 6-2 shows a plot of the six measured temperatures during the first carton run. The thermocouple located under the upper (open) surface of the carton (T5 in Figure 6-2)

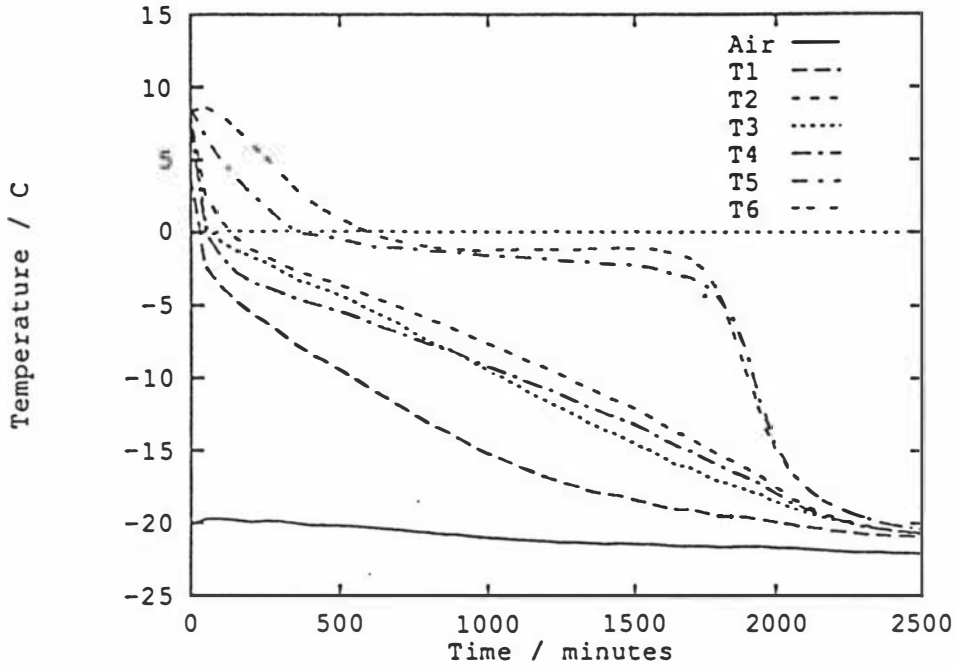


Figure 6-2 Product temperatures around the meat carton shape during freezing — measured.

remained much warmer than would have been expected if the heat transfer coefficient was as high on that face as it was on the other faces. Further, T1 and T4 (the leading and trailing surfaces) should have maintained similar temperatures (as the two surfaces were of the same area), but showed a difference of up to 5°C. T2 and T3 were as close as might be expected given slight variations in depth beneath their surfaces.

To test hypothesis 3, the relative sizes of the heat transfer coefficients in the centre of each surface were evaluated separately using the "Goodman plot" method of Cleland and Earle (1976). The heat transfer coefficient which had previously been found by back-calculating from the freezing time was scaled on each surface to these relative sizes, giving five local heat transfer coefficients — one for each side of the carton where there was a thermocouple. There was no thermocouple on the bottom surface of the carton, so the heat transfer coefficient on that surface was assumed to

be equal to those on the long side surfaces (derived from the measurements recorded from thermocouples T2 and T3).

The FINDIFF program was not capable of simulating a carton shape with a different heat transfer coefficient on each face, so it was modified for the purpose. To maintain the integration accuracy of the earlier predictions, the uneven heat transfer coefficient case was calculated with a finite difference grid of 20×20×16 cells (21×21×17 nodes). The modified finite difference program produced the "Uneven HTC" line in Figure 6-1.

The temperatures predicted by this FD calculation for the centre of each surface are shown in Figure 6-3 for comparison with the experimental temperatures shown on Figure 6-2. T7 is the estimated temperature at the centre of the bottom surface of the carton. The two plots are similar (and Figure 6-3 is much better than the equivalent temperature plot for the "Even HTC" case), indicating that the estimated individual surface heat transfer coefficients were approximately correct.

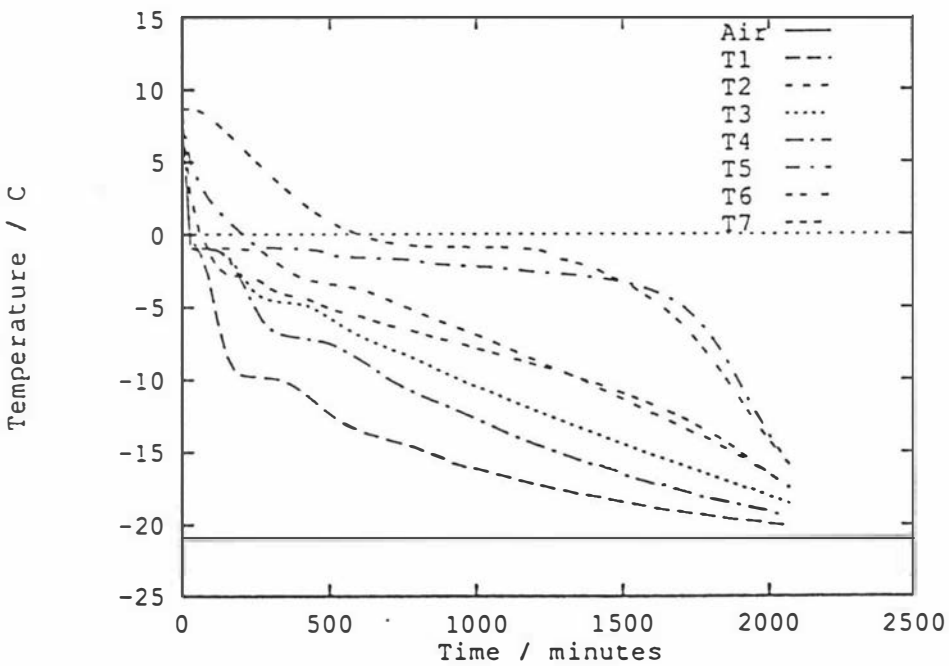


Figure 6-3 Product temperatures around the meat carton shape during freezing — predicted by the FD method with uneven heat transfer coefficients.

The heat load estimated using uneven heat transfer coefficients was a much better fit to that measured experimentally than the uniform surface heat transfer coefficient heat load prediction had been.

Under-estimation of the measured heat load by the FD method early in the run and over-estimation later probably resulted from other uneven heat transfer effects. To properly assess the effects of variable heat transfer coefficient, it would be necessary to measure heat transfer coefficients at more than five points on the carton. It is possible that heat transfer coefficient variation across different parts of the same surface may be at least as great as that found between faces. This was not explored experimentally.

In practice, a typical carton freezer contains several thousand cartons and each carton moves through a variety of positions in the freezer, encountering a variety of air velocities and heat transfer coefficients as it freezes (Pham and Willix, 1987). The total product heat load on an industrial freezer at any time represents the loads of all of these cartons together, but modelling each carton individually (even if only one heat transfer coefficient was assumed for each) would make a complete meat processing plant simulation very complex indeed. A more practical approach is only to model a typical carton with a typical even heat transfer coefficient and to rely upon the averaging effect of the large number of cartons to make the total predicted load similar to the total load on the real plant.

6.2.5 ODE model validation

To apply the ODE model to the conditions found in the experiment, it was necessary to take the uneven heat transfer conditions into account. Pham (1987*b*) showed that (for a slab) where heat transfer coefficients or surface temperatures are asymmetric the freezing time could be estimated by adjusting the critical depth, X, to represent the asymmetry and then calculating as if for symmetrical conditions. An attempt was made to use Pham's technique in conjuction with the method of Chapter 4 to predict the carton shape heat load profile under the experimental

conditions. It would have been very complicated to apply this technique to three-dimensional asymmetry, but most of the carton heat content passes out through its top and bottom surfaces, so it was proposed that the technique only be used in this "primary" dimension and not in the other two.

The critical depth of the carton was adjusted by Pham's (1987*b*) technique to compensate for the uneven heat transfer coefficients on the top and bottom surfaces and E (and hence n) was recalculated for the new critical depth using the method described by Hossain *et al* (1992*a*). The shape of the freezing front in the asymmetric case was not expected to be much different from the symmetric case so N was left unchanged and was still calculated with the original X value using equation (4-31).

The heat load predicted by the asymmetric ODE calculation is shown in Figure 6-1 as the "ODE Method" line. It may be seen that the predicted heat load was very close to that of the "Uneven HTC" FD run except at the start of the process. While this does not constitute a comprehensive test of the ODE method under asymmetric conditions, it does provide some confidence that the method is capable of extension to complicated heat transfer situations.

6.3 Other regular shapes

6.3.1 ODE model validation

Section 6.2 established that finite difference heat load predictions for the meat carton shapes corresponded well with experimental measurements when allowance was made for the difficulties of representing the experimental conditions in a finite difference scheme. This meant that finite difference methods could be used to test the ODE heat load prediction method for a range of regular shapes.

Comparisons of the ODE method against the heat loads predicted by the RADS FINDIFF program for various shapes and freezing conditions are shown in

Figures 6-4 to 6-6. The conditions of these tests were particularly demanding for any heat load or freezing time estimation method:

Initial Temperature:	30°C	- high initial superheat
Final Temperature:	-12°C	- significant sub-cooling
Biot Number:	0.01 to 100	- wide range of *Bi*
Ambient Temperature:	-21°C	

The simulated product was the commonly-used food product analogue Tylose as was used for the meat carton shapes. For each ODE run, N was calculated from equation (4-31) and E was evaluated by the methods described by Hossain *et al* (1992*a*).

Figure 6-4 shows results for a 200 mm thick slab (E = 1.0) with Biot numbers from 0.01 to 100. The time axis has been normalized by dividing by the freezing time, t_f. Heat loads are shown as ratios to the finite difference prediction, so

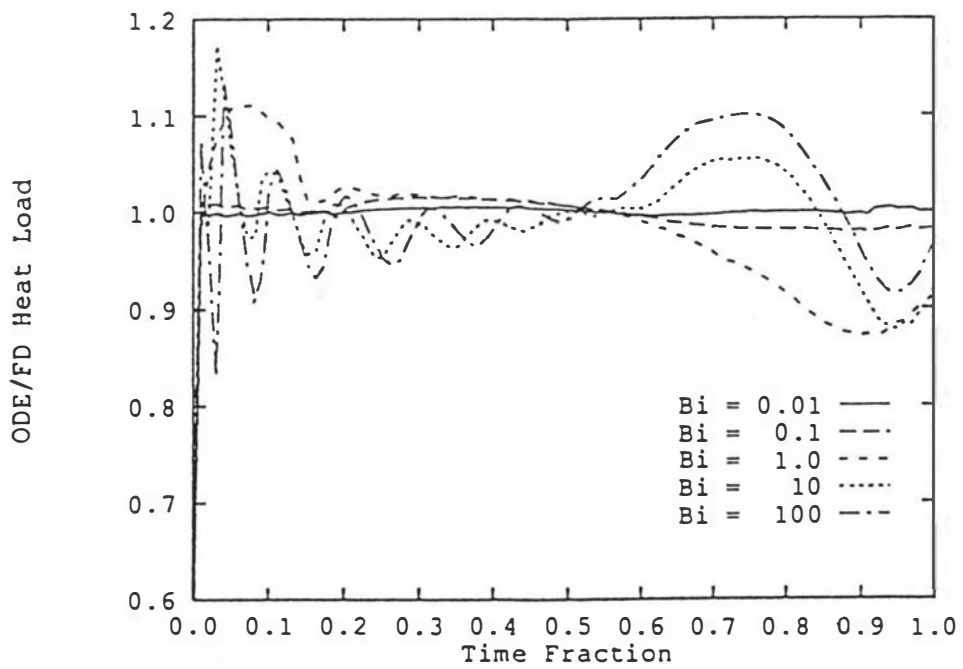


Figure 6-4 Ratio of ODE to FD heat load predictions — Slab, range of *Bi*

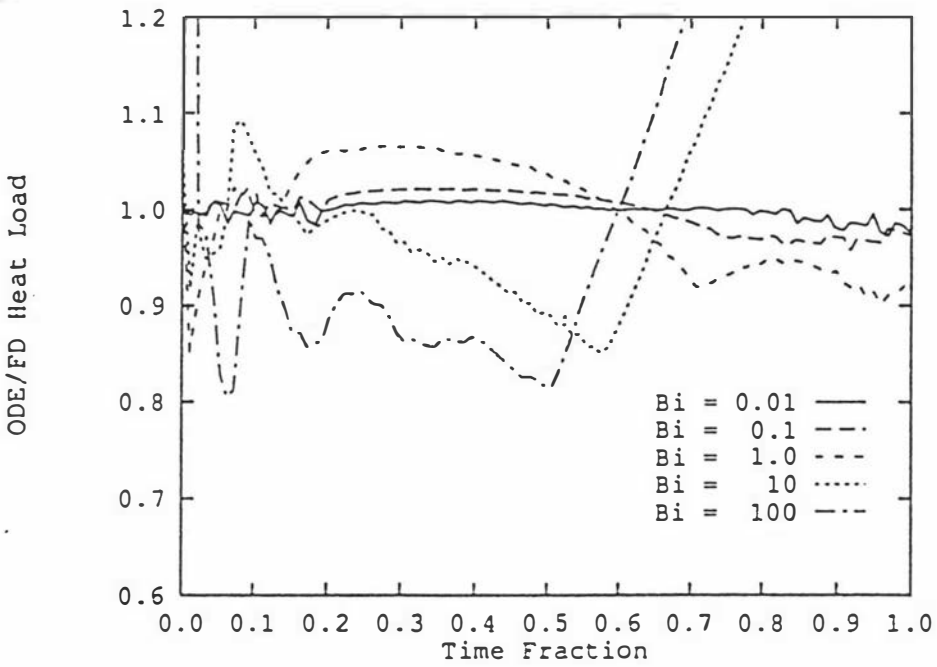


Figure 6-5 Ratio of ODE to FD heat load predictions — Cube, range of *Bi*

that ideally, all of the ratios should equal 1.0 for the whole process. The plotted ratios for Bi = 0.01 and 0.1 are indeed very close to 1.0, but the ODE method predictions for even the highest values of Bi are well within 10% of the FD predictions for almost all of the cooling process.

Figure 6-5 shows results for a 200 mm thick cube cooled under the same conditions as the slab. This is a more difficult test than the slab as E for the cube varies from 3.0 at Bi = 0.01 to 2.23 at Bi = 100. Even so, the Bi = 0.01, 0.1, and 1.0 cases are still within 10% of the FD prediction throughout the process and the Bi = 10 and 100 cases are within 20% of the FD prediction until the process is 70% finished. Deviations late in the process are more acceptable due to the reduced total heat load at this time. Oscillations in the heat load ratio early in the process for the higher Bi cases were not due to the ODE method, but to discretization error in the finite difference calculation which could have been solved by increasing the number of integration nodes used (at the cost of more computer resources).

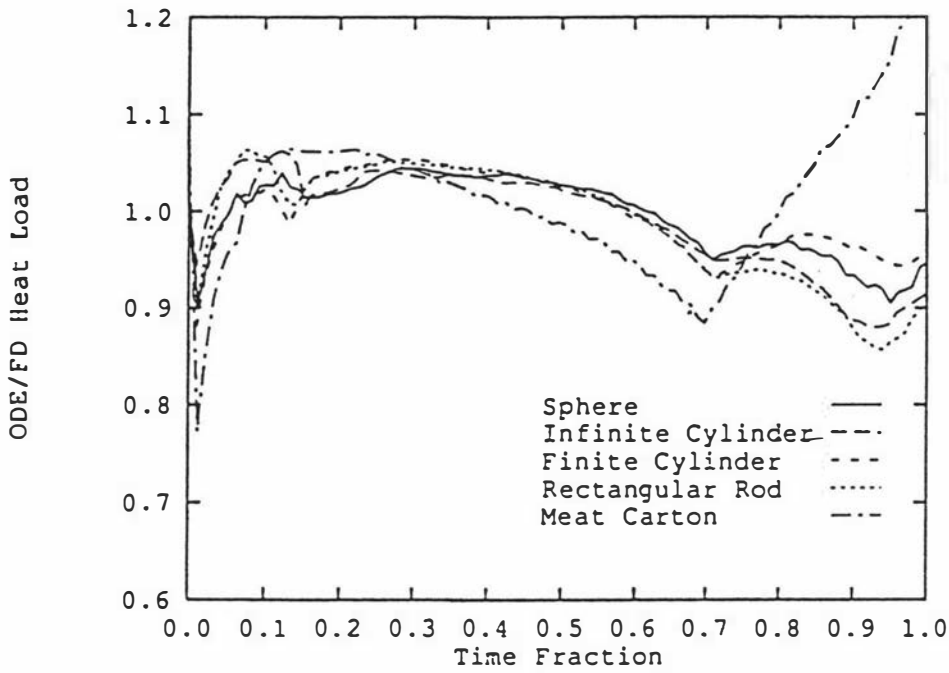


Figure 6-6 Ratio of ODE to FD heat load predictions — Bi = 1.0, range of shapes.

Figure 6-6 shows the ODE to FD heat load ratios for a selection of five shapes presented for Bi = 1.0. The sphere (E = 3.0) and infinite cylinder (E = 2.0, but with E = 2.01 used to avoid the singularity discussed in section 4.9) had diameters of 200mm. The rectangular rod (E = 1.899) was 200 mm square and the finite cylinder (E = 2.447) was 200 mm in diameter and 300 mm long. The meat carton (E = 1.326) was 154 mm by 335 mm by 510 mm. Again, all shapes were well within 10% of the FD prediction except very early and very late in the process.

In section 4.8, it was pointed out that the actual value of N for any shape other than a slab, cylinder or sphere may be expected to change during the freezing process, but that the way in which it would change is difficult to predict. To test the benefits of assuming that N changed in some simple way with the extent of freezing, some ODE method calculations were carried out which assumed that N changed linearly with frozen volume from the value estimated by equation (4-31) to E. This produced no net improvement over using the N of equation (4-31) throughout the run.

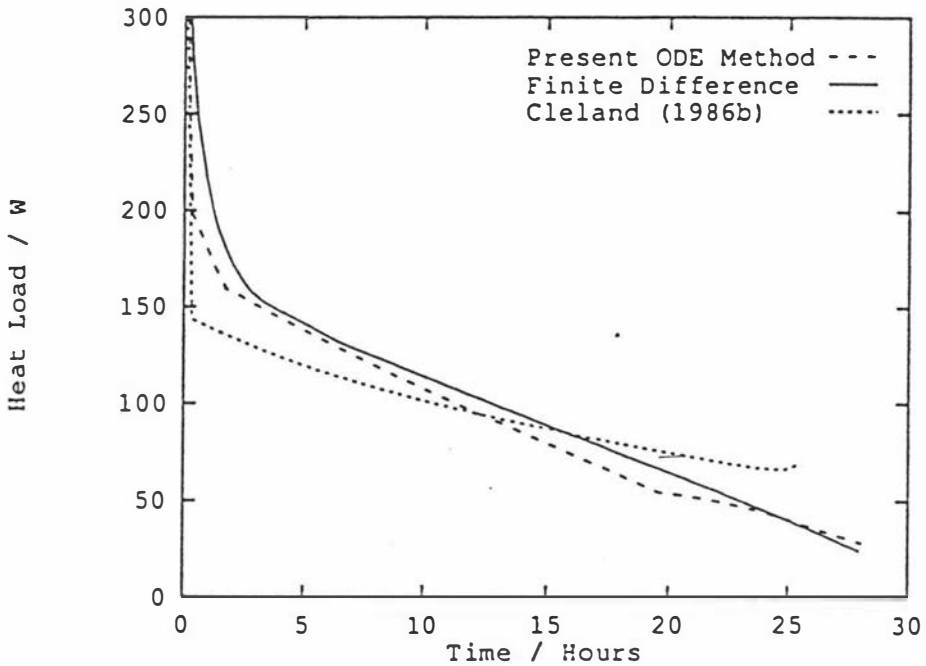


Figure 6-7 Heat load of a carton shape as predicted by the FD method, the ODE method of Chapter 4 and the ODE method of Cleland (1986*b*).

Figure 6-7 puts the comparison in a practical perspective by demonstrating the scale of the heat loads involved for a typical New Zealand meat carton freezing situation. The meat carton run was the same as that shown in Figure 6-6. The prediction of the Cleland (1986*b*) method is also shown for this case. It can be seen that the apparent error in the ODE heat load prediction at the end of the process is, in fact, small. Further, the method of Chapter 4 is a significant improvement over the previous best ODE heat load prediction method of Cleland (1986*b*).

6.4 Lamb carcasses

6.4.1 Experimental results

Summary results for the lamb carcass runs are shown in Table 6-3. Of the nine experimental runs carried out to measure the heat load profile of lamb and sheep carcasses, the three which were conducted at an air velocity of 0.35 ms^{-1} had very

poor energy balances as is shown by the "Enth Chg Ratio" column of Table 6-3. It was considered likely that this was due to air flow stratification in the tunnel and the fact that lower product heat loads at low air velocities made measurement noise a larger fraction of the total load.

Given the air mass flow rate uncertainties discussed in section 6.2.3 and the difficulty in estimating thermal property data for carcasses of varying composition, runs 1, 2, 3, 4, 5 and 9 had a tolerable level of heat balance error.

Variability in heat transfer coefficient over the product surface was expected to be less important with the carcasses than it was with the cartons because for most of the runs, the carcass surfaces were covered with wrappings which were expected to dominate the value of the heat transfer coefficient. The rate of heat transfer from the cooling meat into the cavity of the carcass was expected to be essentially zero and temperature measurements made by the probes inserted into various parts of the carcass confirmed this.

No finite difference or finite element calculations were conducted for the carcasses as it was difficult to accurately describe their shape to the FD or FE method. The following analysis therefore focuses directly upon testing the ODE method.

To apply the ODE model to a lamb carcass, it was necessary to obtain the values of the two shape factors, E and N.

6.4.2 Evaluation of E for a lamb carcass

E has previously been evaluated for a lamb carcass by Cleland and Earle (1982), who calculated that an E value of 2.4 allowed freezing times calculated by freezing time prediction methods to best fit experimentally-measured freezing times. In the light of the work of McNabb *et al* (1990), the opportunity existed to evaluate E by a new technique.

The E-estimation method (Hossain *et al*, 1991*b*) required that an irregular shape be approximated by an ellipsoid with the same volume, cross-section

Table 6-3 Lamb and sheep carcass experimental data summary

Run	Wrapping	Mass (kg)	Critical Dimension (m)	Exp Enth Chg (J)	Exp Enth Chg (J/kg)	Starting Temp (°C)	Ending Temp (°C)	Frz Time (hrs)		HTC (W/m^2K)	Th Enth Chg (J/kg)	Enth Chg Ratio
								Leg	Shoulder			
1	Shrink	11.67	0.131	14727039	252391	8.3	-18.0	10.3	15.1	14.4	247333	1.02
2	Shrink	10.48	0.124	16447569	313885	12.3	-19.7	12.8	12.7	17.0	264798	1.19
3	Poly+Stk	23.36	0.179	36361465	311314	17.4	-19.7	18.0	24.7	13.7	281934	1.10
4	Poly+Stk	26.65	0.190	35898854	269410	14.4	-19.8	14.7	21.4	18.5	272086	0.99
5	Bare	22.09	0.175	25803835	233625	17.7	-19.2	9.4	13.6	35.0	281773	0.83
6	Poly+Stk	21.20	0.172	52519199	495464	14.7	-17.4	33.3	40.8	6.3	267378	1.85
7	Stockinet	23.79	0.181	23266702	195601	16.6	-17.3	21.5	27.0	11.6	273516	0.72
8	Bare	20.10	0.168	19694101	195961	15.7	-16.3	14.4	18.0	18.0	267999	0.73
9	Polybag	27.70	0.193	32265715	232965	14.8	-13.7	23.0	23.0	16.2	258074	0.90

Notes:

Shrink - Shrink wrapping
Poly+Stk - Polythene bag with stockinet covering
Polybag - Polythene bag

- Starting and Ending temperatures (in the columns "Starting Temp" and "Ending Temp") are means of all temperatures measured, and therefore only approximate the actual mass average temperatures at those times.
- All runs had an air temperature of -21°C. Air velocities were $1.42ms^{-1}$ (runs 1, 2, 3, 4, 5 and 9) and $0.35ms^{-1}$ (runs 6,7 and 8).
- The columns labelled "Exp Enth Chg" show the total enthalpy change and enthalpy change per unit mass evaluated by numerically integrating the measured heat load data.
- The columns labelled "Frz Time (hrs)" show the experimentally measured times required for the product to freeze to -10°C in the deep leg and deep shoulder positions respectively.
- The column labelled "HTC" shows the mean heat transfer coefficient estimated for each carcass by changing the HTC parameter of the Pham (1986) freezing time prediction equation until the freezing time prediction agreed with the experimental freezing time for the carcass shoulder.
- The column labelled "Th Enth Chg" shows the theoretical enthalpy change per unit mass between the starting and ending temperatures using the enthalpy data of Fleming (1969).
- The column labelled "Enth Chg Ratio" shows the ratio of the "Exp Enth Chg" value to the "Th Enth Chg" value for each carcass and is therefore an indication of the energy balance for each run. For runs 5, 7 and 8, evaporative cooling was important, but was not measured by the differential air temperature method.

perpendicular to the major axis, and characteristic dimension. Three lamb carcasses of widely varying weights and grades were frozen and cut through in various areas to obtain this data. It was not possible to do this with the carcasses actually used the heat load experiment.

Firstly, each carcass was divided into three sections — leg (from the front of the pelvis backwards), loin (from the front of the pelvis to two ribs up from the base of the sternum, including most of the rack and all of the flap), and shoulder (the rest of the carcass, including the neck and forelegs). Each of these sections was expected to include a thermal centre (a unique location which would be the last part of the section to freeze). Each section was weighed and its volume calculated from its mass and density. The leg and shoulder sections were then cut in half vertically along the spine, and all sections were cut into slices perpendicular to the apparent major axis of the section (usually the central bone). The slice with the greatest characteristic dimension from each section was selected as the likely location of the thermal centre for that section. For the shoulder and leg sections, both left and right halves were measured.

It has previously been assumed that the thermal centre of a whole lamb carcass lay at the "deep leg" location. Consideration of the freezing times shown in Table 6-3 indicates that low heat transfer within the carcass cavity for most of the experimental runs resulted in the last frozen point being within the shoulder, close to the inner cavity. For the purposes of the ODE heat load prediction method, the critical depth, X, was taken as being the full thickness of the shoulder, and E was calculated from the geometric data using the method of Hossain *et al* (1992*b*) to be 1.48.

6.4.3 Evaluation of N for a lamb carcass

It was suggested in section 4.9 that N could be estimated by equation (4-31). Sections 6.2 and 6.3 have confirmed that this was a useful approximation for regular shapes.

For the lamb and sheep carcasses, V, the carcass volume, was estimated from the known mass and estimated density. X was measured whilst evaluating E.

The surface area of the carcass was more difficult to estimate. It was assumed that the carcass area was similar to the pelt area and the carcass mass vs. area data of Fleming and Earle (1967) were correlated using a second order polynomial to produce the following result:

$$A = 0.0653M - 0.000927M^2 \qquad (6\text{-}1)$$

where A was the pelt area of the carcass in square metres and M was the carcass mass in kilograms. The critical dimension (i.e. $2X$) is shown in Table 6-3 for each carcass. The N values calculated from this equation were about 3.7 for runs 1 and 2, and about 4.0 for the other runs.

6.4.4 Evaluation of thermal properties

The ODE heat load estimation method required a number of thermal properties to characterise the material to be cooled. It was impractical to render down entire carcasses in order to estimate a mean composition, so the enthalpies measured by Fleming (1969) were used together with thermal conductivities estimated from Fleming's measured compositions using the methods recommended by Miles *et al* (1983). The following properties were used as input data for the heat load prediction method:

- Frozen thermal conductivity (1.486 W m^{-1} K^{-1})
- Unfrozen thermal conductivity (0.467 W m^{-1} K^{-1})
- Volumetric enthalpy at the onset of freezing (based on 0 enthalpy at -40°C) (264.7 MJ m^{-3})
- Frozen volumetric specific heat capacity (1.945 MJ m^{-3} K^{-1})
- Unfrozen volumetric specific heat capacity (3.476 MJ m^{-3} K^{-1})

6.4.5 Evaluation of the mean heat transfer coefficient

The heat transfer coefficient in each run was evaluated indirectly. Given the measured time required for each carcass to freeze to -10°C, a freezing time prediction method of established accuracy (Pham, 1986) was used to back-calculate the mean heat transfer coefficient. The heat transfer coefficients thus derived are shown in Table 6-3.

6.4.6 Comparison of experimental carcass results with the ODE method

When the N values found from equation (4-31) were used to predict the heat load profiles of the lamb carcasses studied in the experiment, the fit of the prediction to the experimental measurement was found to be poor. Figure 6-8 shows a typical example for carcass run 4.

It was pointed out in section 4.9 that equation (4-31) was only strictly correct at the start of the freezing process, and that N could be expected to vary after that time as the shape of the unfrozen region changed. In a lamb carcass, the shape of the unfrozen region would quickly become quite different from the shape of the carcass surface, and therefore the true value of N would also change quickly. Further, the multiplicity of thermal centres (both shoulders, both legs, the loin, and possibly others) mean that later in the process there would be more than one unfrozen region, thus conflicting with one of the assumptions made in developing the model (section 4.5).

The range of values which may be assumed by N has not been fully established. Hossain *et al* (1992*b*) note that $E \leq A\,X\,/\,V$ under all conditions so it was postulated that $N = E$ and $N = A\,X\,/\,V$ might represent bounds on N. The heat load profiles predicted using these bounds are shown for carcass run 4 in Figure 6-8 (for which $A\,X\,/\,V = 4.0$). As a further refinement to the upper bound, it was noted

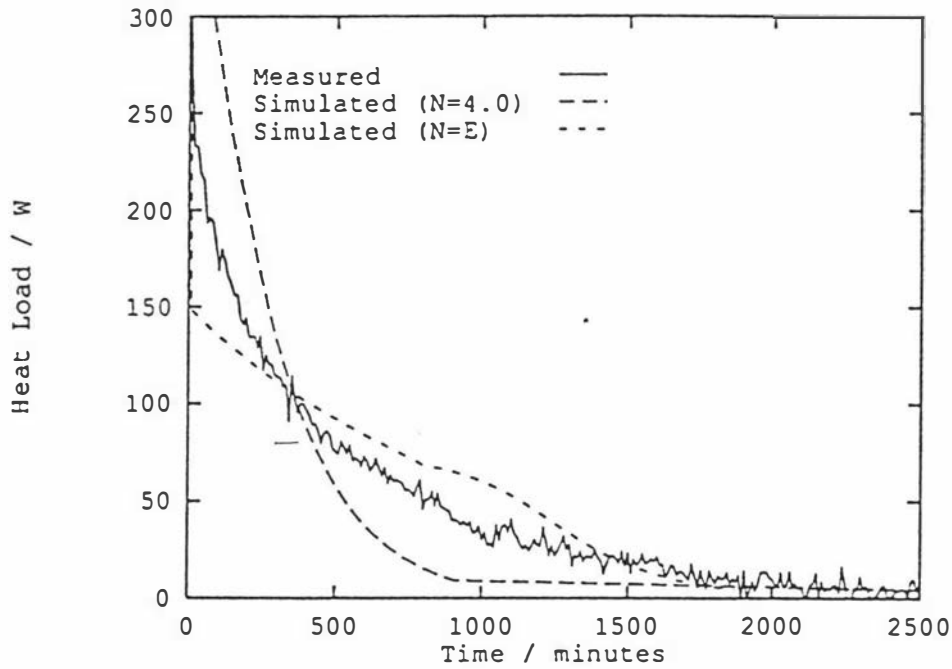


Figure 6-8 Measured heat load for lamb carcass run 4, and loads predicted by the ODE method which show the effects of using extreme *N* values.

that the regular shape with the highest surface area to volume ratio is the sphere, for which $A X / V = 3.0$. *N* values higher than 3.0 make the initial part of the heat load profile very steep and such a steep initial heat load profile did not seem likely in practical circumstances.

In the absence of better information, a trial and error approach was used and an *N* value of 2.5 was found to provide the best fit to the experimental data. Figures 6-9 to 6-17 show the experimental heat load profiles and the heat load profiles predicted by the ODE method for the nine carcass runs. For each of runs 1, 2, 3, 4, 5 and 9, the ODE heat load estimate was within about 10% of the experimental result during the first half of each run, and within 20% of the experimental result during the second half of each run, when the absolute heat load was smaller. The low air velocities of runs 6 to 8 made them significantly worse than the higher air velocity runs.

The sudden change in the direction of the predicted heat load curve about half way through each plot was due to a transition from the freezing model to the sub-

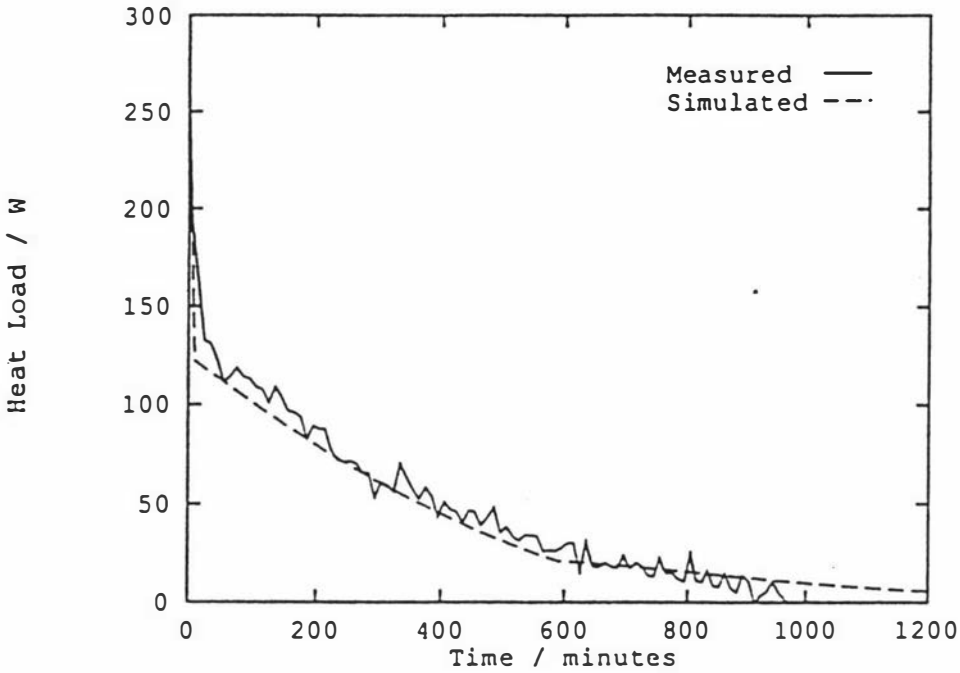


Figure 6-9 Measurement and ODE prediction of lamb carcass heat load — Run 1.

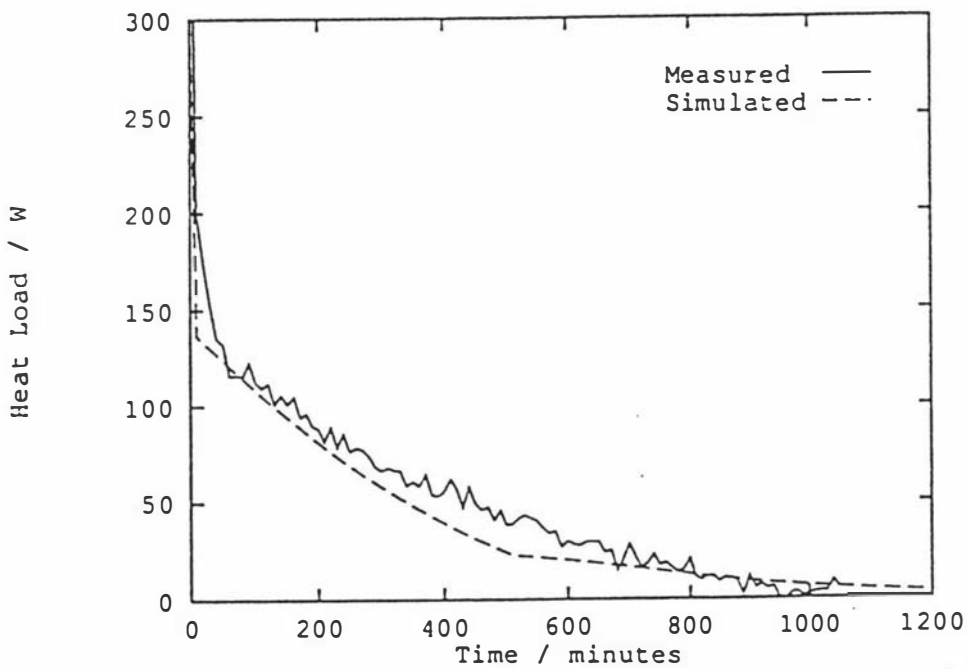


Figure 6-10 Measurement and ODE prediction of lamb carcass heat load — Run 2.

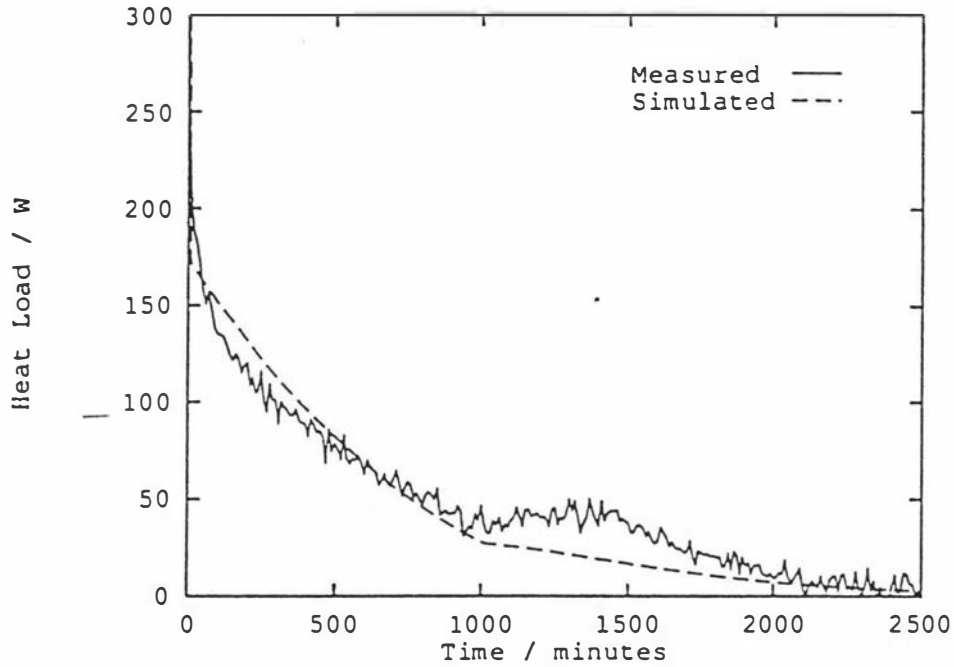


Figure 6-11 Measurement and ODE prediction of lamb carcass heat load — Run 3.

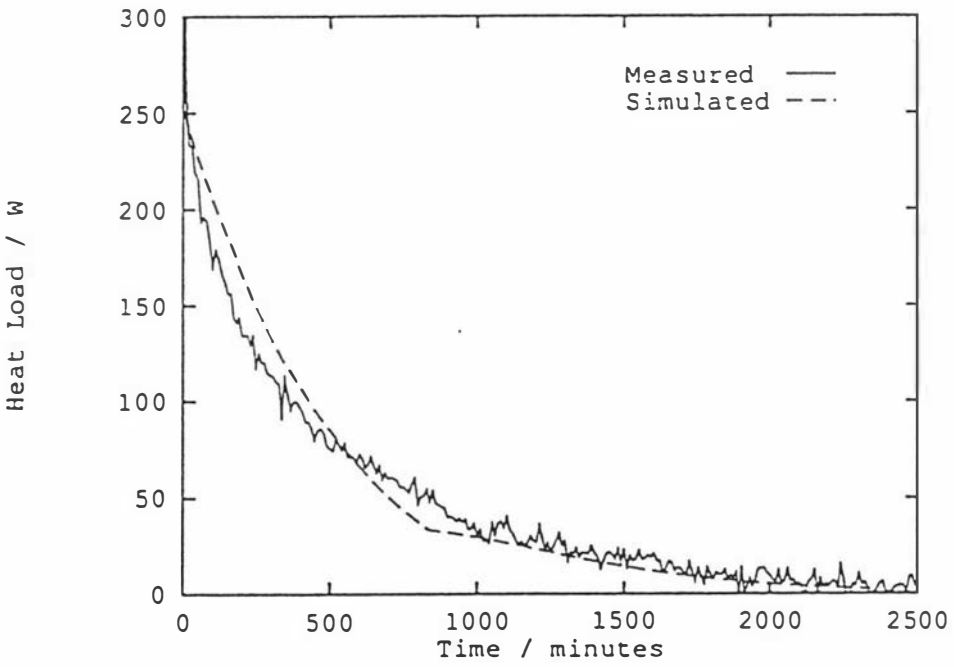


Figure 6-12 Measurement and ODE prediction of lamb carcass heat load — Run 4.

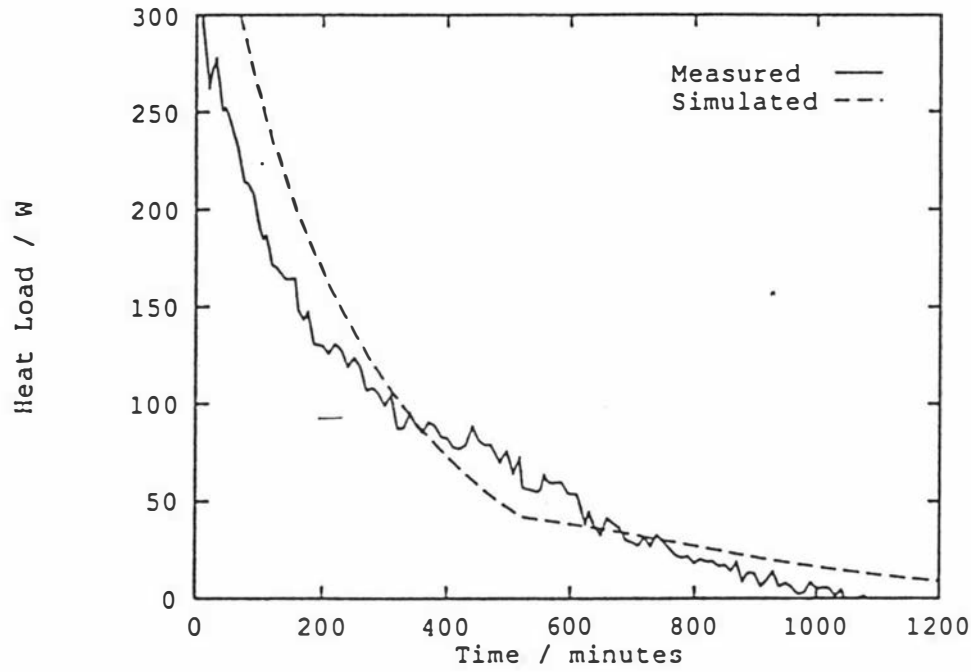


Figure 6-13 Measurement and ODE prediction of lamb carcass heat load — Run 5.

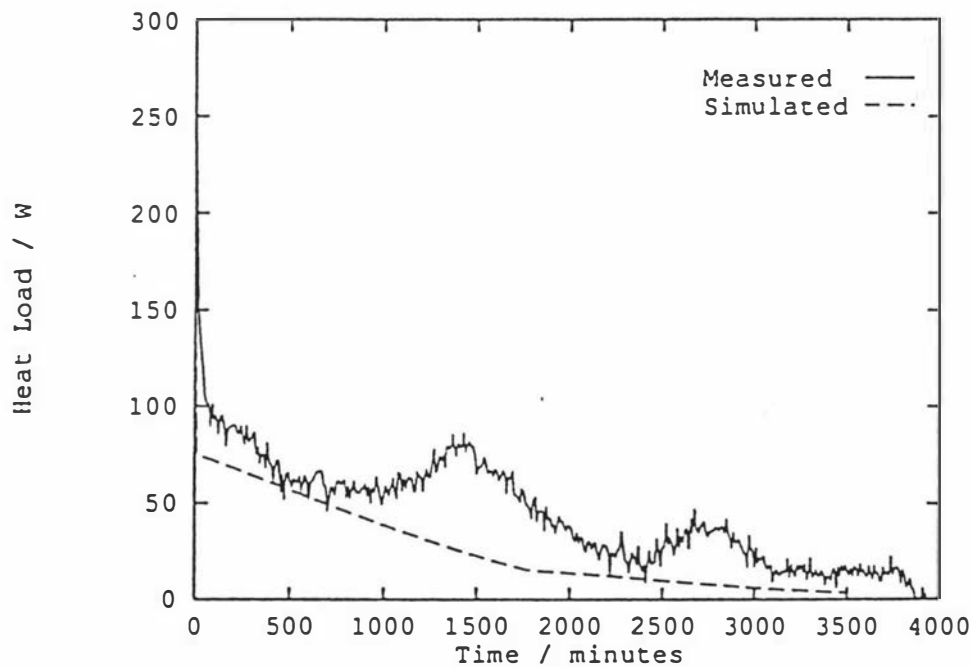


Figure 6-14 Measurement and ODE prediction of lamb carcass heat load — Run 6.

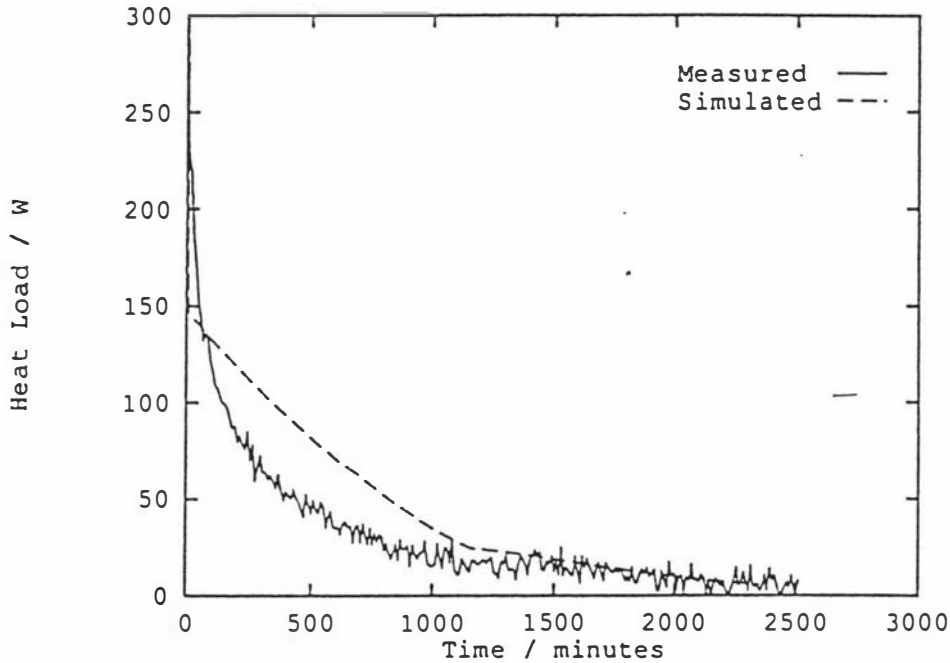


Figure 6-15 Measurement and ODE prediction of lamb carcass heat load — Run 7.

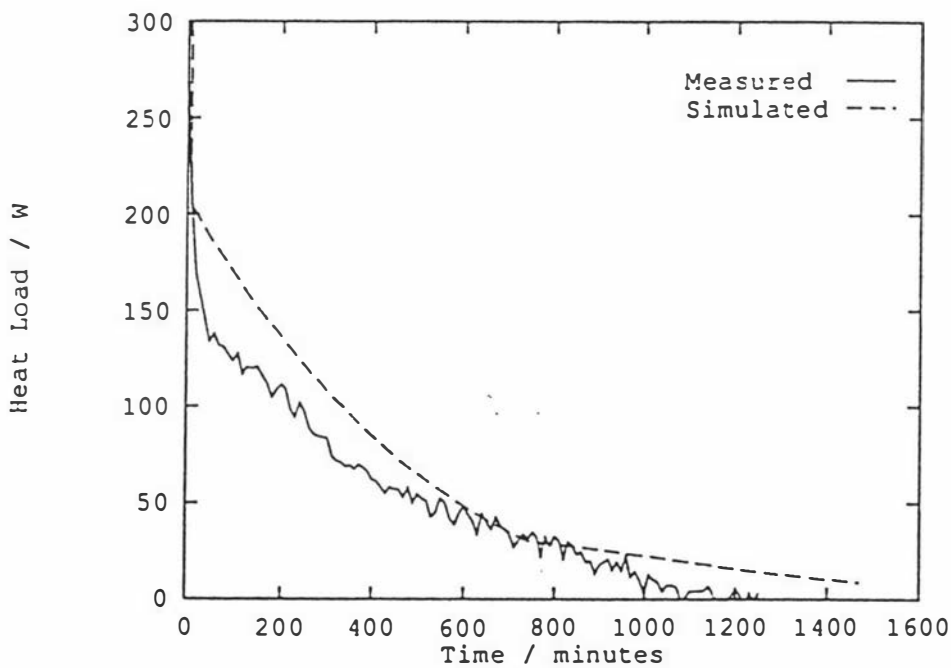


Figure 6-16 Measurement and ODE prediction of lamb carcass heat load — Run 8.

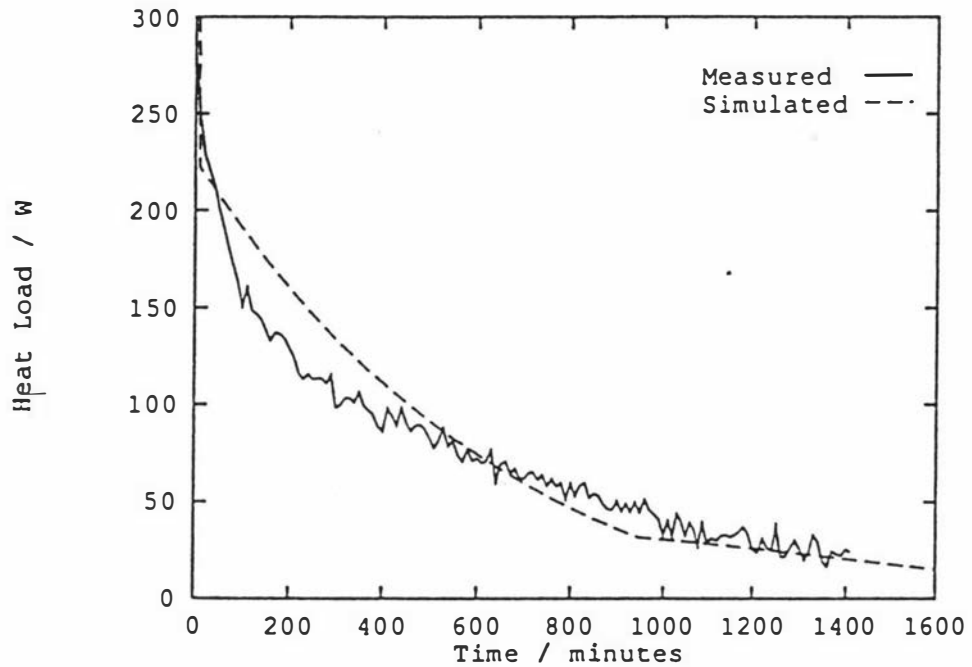


Figure 6-17 Measurement and ODE prediction of lamb carcass heat load — Run 9.

cooling model. The reasons for this lack of smoothness have been discussed in section 4.8.

As was done with regular shapes in section 6.3, an attempt was made to model the changing value of N for carcass heat load prediction by assuming that N changed linearly with frozen volume, but this was not pursued as there was no net improvement compared with the results predicted by using a single N value.

6.5 Discussion

6.5.1 Experimental technique

While the results of the product freezing heat load measurements were not as consistent as had been hoped, they were good enough to show that the differential air temperature measurement method was practical and useful.

Some of the problems encountered in the experimental technique were not fully resolved. Inaccuracies due to poor air distribution at low air velocities, and variability in heat transfer coefficient over the surface of the product were problems which reflected the environment in which the product was frozen. These problems would be lessened in practice by the averaging effect of the large numbers of carcasses or cartons found in an industrial freezer.

It is hard to obtain accurate estimates of the heat transfer coefficients over a product surface. Even if good heat transfer coefficient estimates were available for each part of the product, it would be difficult to include such detail in a relatively simple heat load prediction method. It would also be impractical to require that the user of such a method provide such detailed data.

6.5.2 ODE heat load prediction method

The ODE heat load estimation method described in Chapter 4 performed at least up to the standard of the measured data for both the carton and carcass runs. The expression $N = A\ X\ /\ V$ appears to be satisfactory for regular shapes.

Where a shape is irregular and has more than one thermal centre, the following approximate guidelines could be used:

- Calculate $N = A\ X\ /\ V$.
- Calculate E by the methods of Hossain *et al* (1992*a*,*b*).
- If N is greater than 3.0, set $N = 3.0$.

These guidelines are based only upon the author's experience, and should not be used once guidelines which have some theoretical basis are developed. It is speculated that there may be some link between N and E, but while the nature of this relationship is uncertain, an additional guideline that seems to have some merit is to restrict N so that it may not be greater than about $E+1$.

Chapter 7: Simulation Environment Design

A structure within which a group of models may be implemented and connected together is most commonly called a *simulation system.* In the present context, where the subject of simulation is also a system, such terminology may become confusing so such a structure is described here as a *simulation environment.*

For brevity of reference, the environment which was designed in this project for the purpose of applying dynamic models, controls and operating strategies to simulations of real or hypothetical refrigeration systems has been given the name RefSim. This chapter describes the principles which were used to design the RefSim environment and the design itself. The name is also shared by the computer program in which the design was implemented and that computer program is described in Chapter 8.

7.1 Design principles

Several general principles were considered in designing the RefSim environment.

7.1.1 Simulation as a representation of reality

The concept of *model* may be defined in a variety of ways, depending upon point of view and level of abstraction. When considering the design of an environment in which to implement models, it is appropriate to use a definition which suggests the implementation of a model as a sequence of instructions while remaining at a high level of abstraction:

> A model is not only a "compact representation of a real phenomenon which can generate a behavior comparable with some behavior of interest in the real system" but also a vehicle to make more evident key characteristics of an object under study.
> — Elzas (1984)

This definition of *model* has a number of important consequences.

A model is a representation, therefore it has a form by which it represents reality. The quantitative models with which this project is concerned are mostly represented by mathematical expressions, but few are completely represented by mathematics alone. Many models require the evaluation of several expressions in a particular order. Others require decisions (which depend upon their current states) to be made in order to determine their future states. Some may interact with other models. It is necessary to use an algorithmic form rather than a mathematical form to completely describe such models.

A general algorithmic form may be useful in some contexts but if a model is to be evaluated without error then the representation must be such that it can be evaluated automatically. This suggests that the best representation for a model is as (possibly part of) a computer program in an implemented programming language. While correct evaluation is essential, a model must also be understandable to modellers and users. A computer program representation aids this goal at the syntactic level because the rules for interpreting the program are unambiguously expressed in the language definition. On the other hand, a poorly-written computer program will defeat this goal at the semantic level.

A model is a compact representation. If one assumes that the phenomenon to be represented contains no redundancy, then it cannot be completely described by any model which is less complex than itself. This implies that a compact representation must describe the real phenomenon incompletely. If there is a model

which completely describes the input/output behaviour of the real system, then Elzas (1984) terms it the *base model.*

A model generates behaviour. Given a set of inputs, it will produce a set of outputs such that the outputs are determined by the combination of the current model state with the inputs. Where a model generates behaviour over a period of time, the set of inputs may include all inputs up to the time of the output, and so it may be necessary for a model to have a "memory" of past events.

The behaviour generated by a model is comparable to some behaviour of interest in the real system. When designing a model, it is necessary to define the behaviour which is of interest and the extent to which the behaviour generated by the model should be comparable with the behaviour of the real system. This comparability is confirmed during the validation phase of model development.

By restricting the behaviour of interest and the required level of comparability, the scope and therefore the necessary complexity of the model may be greatly reduced. A very common restriction limits interest to the boundaries of the model. If this is so then the model is free to have any interior structure which generates behaviour comparable to the real system. The lowest level of detail which is of interest must be defined early in the model development process as it may strongly affect model generality and complexity.

Finally, a model is a vehicle to make more evident key characteristics of an object under study. A model should have a defined purpose otherwise there is no point in its existence. A clear purpose can make the model development process much easier than it would otherwise be.

Given a set of models which generate behaviour, simulation is the process of behaviour generation and a simulation environment is a structure which facilitates this process.

7.1.2 Model validation

One of the purposes of generating model behaviour is to compare it with the behaviour of the real system. This critical comparison is called model validation.

Following Elzas (1984), an *experimental base* is the set of all experiments which may be performed upon the real system. From this set, Elzas (1984) further defines an *experimental frame* which is that subset of all feasible experiments which may be performed reproducibly on both the real system and the model. A fully validated model should completely reproduce the behaviour of the real system which has been defined as being of interest to the modeller.

The form of model description should be designed to assist validation. It should allow easy access to all features of the model which are included in the experimental frame. The boundaries of the model should be defined so that they match the boundaries of the real system. Thus the modelled phenomena exist exactly within the boundaries of the real system.

The simulation environment should be designed to assist validation. When a group of models are connected together to simulate a larger system, the behaviour of interest for each model should still be accessible to the modeller. When it is desired to validate a single model independently of others, it should be possible to set up a "test-rig" within the simulation environment to provide the model with its input data without the need to simulate a larger system or to include other models whose behaviour may complicate the validation process.

7.2 Scope of a refrigeration system simulation

The area of interest to this project was the simulation of industrial refrigeration systems — especially those found in the New Zealand meat industry. Consideration of the broad range of compression refrigeration plants and heat pumps required little additional effort. Extension of the simulation scope to absorption refrigeration systems would have required significantly more work with little

practical gain in the context of the project, so this was not done. The elements of the simulation may be divided into three main groups:

7.2.1 Refrigeration system components

The simulation environment contains models for all of the components found in a compression refrigeration system. The refrigeration system components to be modelled may include compressors, filters, pipelines, condensers, refrigerant vessels, expansion and control valves, heat exchangers and evaporators. At lower levels of abstraction, models may include (for example) individual cylinders, inlet and outlet valves within a compressor. At higher levels of abstraction several real components may be combined into a single model, for instance by including filters as part of a pipeline model, or pipelines as part of a refrigerant vessel model. Identification of the most appropriate level of abstraction will be discussed in section 7.5.

7.2.2 Application components

The behaviour of a refrigeration system is usually dependent upon the load which it sustains. This load is generated by the refrigeration application. The range of possible refrigeration applications is large, but a number of common components may be identified. Freezing and chilling food product is a common application in the food process industries. Frequently, there is a fluid (e.g. air, brine) by which heat is transferred from the product to the evaporator. The boundaries of the room or vessel in which the food processing takes place may also be components. In the case of an air-filled room, these boundaries may comprise walls, ceiling, floor, doors and ventilation ducts. In addition to product, there may be structures with significant heat capacity, machines, personnel and other sources of sensible and latent heat which may affect the behaviour of the refrigeration system. Each of these items may be regarded as application components.

7.2.3 Control systems

All but the smallest and simplest refrigeration systems operate away from their normal point of balance, so control systems are important components of refrigeration systems. Controllers range from single-input single-output differential, P, PI, PID and similar controllers used for evaporator and individual compressor control to sophisticated multiple-input multiple-output strategy controllers. In a refrigeration system where the available capacity exceeds the application load, control action may dominate system dynamics.

The controllers to be considered by a simulation may not be restricted to the programmable-logic, analogue electronic and computerised automatic devices which are normally considered as controllers. Where the behaviour of the refrigeration system is significantly affected by the actions of the plant operator, that person should be considered as a controller for the purposes of the simulation, even though the decision-making process of an operator may be difficult to model when compared with automatic controllers.

In the context of this project, control systems were particularly important as the development of a simulation environment for the testing of sophisticated control strategies was one of the reasons for the project.

7.3 Combined continuous/discrete event simulation

A dynamic system generates behaviour over a period of time. As time passes, the state of the system may change in some manner which a dynamic simulation is intended to predict.

Kreutzer (1986) has identified three principle paradigms which have been used in system simulation. The first, *continuous system simulation*, views the system behaviour as a smooth progression through a set of states. The passage of time in a continuous system may be sliced into arbitrarily small increments and as the increments become smaller the behaviour of the system appears progressively

smoother. This concept has proved useful to engineers in all fields, and almost all of the refrigeration system simulations discussed in the literature review have applied the continuous system paradigm. Models in this paradigm are described using ordinary or partial differential equations.

The second paradigm, *discrete event simulation*, views the system behaviour as a set of state changes which occur at discrete time intervals. The state of the system does not change between events, but at the time of each event there is a sudden step change from one state to another. The time between discrete events is of no interest, and is therefore not simulated. The discrete event paradigm has been used in the information and management sciences, and queuing models form a typical example.

The third paradigm, *combined continuous/discrete event simulation*, has been developed more recently. A combined simulation views system behaviour as a predominantly continuous phenomenon which incorporates the possibility of irregularities and discontinuities. Between discrete events, the simulation proceeds in a continuous manner, but step changes of state may occur at each event. This allows the combined simulation to deal with real-world discontinuities which may defeat continuous simulations. One example is the possibility that a simulated fluid vessel may become empty or overflow. Combined simulation is a superset of the two preceding paradigms, but efficiency and complexity considerations mean that this paradigm is used only where a combined approach is necessary for accurate simulation.

7.3.1 Applicability to refrigeration system simulation

The majority of refrigeration system simulations have been conducted using the continuous simulation paradigm. This being so, it was necessary to justify the use of the more complicated combined simulation paradigm in the present work.

The example of a fluid vessel which may empty or overflow often exists within a refrigeration system. James (1988) identified this as a possibility in the

FME refrigerant vessel model, but dealt with it in the conventional manner for a continuous system simulation: such an event does not occur if the plant is operated correctly so it need not be considered in simulations of normal operation. Other discrete events may be caused by the scheduled switching of compressors from on to off, off to on, or between compression pathways. Evaporators may be turned on, turned off, or defrosted at scheduled times.

For application components, discrete events are often more important due to the short reaction time of air-filled rooms. Product may be loaded and unloaded and doors may be opened and closed, both with dramatic and sudden effects upon the heat load in the room.

Control systems may also change state suddenly, though they may be more difficult to deal with if the state changes cannot be predicted in advance.

In a complex industrial refrigeration system simulation, all of these discrete events may occur and to ignore them would risk significant error. Cleland (1985*b*) had found it necessary to deal with the opening and closing of doors as discrete events within the RADS simulation environment but the discrete event mechanism was not made available to any other model. After considering the other discrete event types discussed above, it was decided that RefSim should be designed as a full combined continuous/discrete event simulation environment with both continuous simulation and discrete event facilities available to all models.

7.3.2 Special difficulties

Two different kinds of event may exist in a combined simulation. *Time events* are scheduled in advance of their occurrence to change the system state at a predefined point in time. *State events* are triggered by the system itself when a predefined set of conditions is met.

Time events may be dealt with by scheduling the event well in advance and carrying out a continuous simulation exactly up to the event time. State events are more difficult to handle because they interrupt the normal course of continuous

simulation and require modifications to the continuous simulation algorithm so that their occurrence may be recognised.

It is important for efficiency and accuracy reasons to recognise each discrete event so that it may be handled properly. The alternative approach of ignoring discrete events may cause either a dramatic reduction in the rate of continuous simulation (as the continuous simulation algorithm reduces its time step in an attempt to step over the discrete event) or a dramatic increase in integration error after the event (if the continuous simulation algorithm is incapable of error control) (Crosbie, 1984).

7.4 Appropriateness of object-oriented design to the simulation problem

In a survey of general system simulation styles and languages, Kreutzer (1986) evaluated a variety of simulation design and implementation methodologies and assessed the object-oriented methodology as being "a very elegant and productive programming style for dealing with complex models and software systems" (p.103). Kreutzer concluded for combined simulation that "object-oriented modelling seems to offer natural representations of systems of this kind" (p.205).

The object-oriented point of view seeks to correlate the structure of the simulation with the structure of the real system. Each component of the real system is represented by a distinct component (an *object*) in the simulation. Every interaction between real system components is identified with a corresponding interaction between objects. Superimposing the real system structure upon the simulation produces a model which is both a quantitative and a structural representation of the system.

Four basic features characterise an object-oriented methodology: encapsulation, inheritance, polymorphism and message passing. Object-orientation has been called "anthropomorphic programming or programming by personification" (Lalonde and Pugh, 1990, p.1) so it is natural to discuss objects as if they were

active entities rather than symbolic representations of real system components. This practice is followed in the following discussion.

7.4.1 Encapsulation

In most computer program design methodologies, data and function are separate concepts which only come together when function transforms data or data determines a choice of function. This does not accurately represent a real system, where data and function are often very closely associated.

Where there is a loose connection between data and function, the data for one system component may be as closely bound to the function of another component as it is to its own. This can make it very difficult to identify where one component meets another, where the boundary between them lies, and how they relate to each other.

Object-orientation encapsulates an object's data along with its function. The state of an object is only directly available to the object itself, and not to any other object. The function of the object may only be performed upon its own data, and not upon data belonging to some other part of the system.

Having encapsulated an object's code and data together, there remain a small set of relationships which the object may have with other objects. These are identified as the interface between the object and the rest of the system, or perhaps between the object and other specified objects in the system. Where an object represents a model in a simulation system, this interface clearly defines the model boundary and what links it has to other models.

Besides making the model expression much clearer than in other designs, the object-oriented approach localizes any model errors, making model development much easier. Model validation is enhanced by encapsulation because an encapsulated object may be extracted from a larger system and fitted into a "test-rig" with little difficulty. The test-rig design is straightforward because the inputs required by the model to be tested are explicitly defined by its interface.

7.4.2 Inheritance

In a real system, items may be grouped into classes and sub-classes. Members of a class share a number of properties and members of sub-classes share the properties of their superclass and have additional properties of their own. It makes sense to develop models in such a way that the common properties of the objects are grouped together and inherited by each of the derived objects. Only the differences between objects need be considered separately.

An object-oriented design allows this commonality to be exploited by having an explicit hierarchy of objects with common properties implemented only by the class whose descendants share those properties. As with encapsulation, this localizes model design and construction errors. Inheritance also recognizes similarities between models which mirror the similarities between real system components, thereby enhancing the correlation between the real system and its simulated representation.

7.4.3 Polymorphism

As well as having common properties, a group of models may often have common functionality, even though the details of that functionality may differ. For example, both valves and doors may open and close, but the ways in which they do so can be quite different. The idea of polymorphism lets a group of objects share a function while allowing each object to specify a different technique for carrying out that function. When an object receives the instruction to open (for instance), it will do so in its own way and the object itself is responsible for choosing that way.

Polymorphism further enhances encapsulation because it allows an object's function to be known to other objects without making available the knowledge of how the object carries out that function.

7.4.4 Message passing

Once an object's function is encapsulated with its data, it becomes essential to provide a mechanism for transferring data between objects. The technique used in object-oriented systems is that of the message.

When object A wishes to obtain some information about the state of object B, A sends a message to B asking for that information. If B can provide that information, then the information is returned to A. Only certain of the properties held by B are available to be sent to other objects. This concept can be enhanced by the convention that A must identify itself to B when requesting the information. Therefore, B can decline to supply information to A should B consider A to be an inappropriate object to receive that information.

The same principle applies should object A wish to change the state of B. Object A must send B a message requesting that B change its state, which B may accept or decline. Thus the only object that may actually change the state of an object is the object itself.

The message passing convention prevents objects from directly accessing or manipulating the contents of other objects. A distinction is drawn between an object's own data which remains unchanged unless the object explicitly changes it, and data belonging to other objects which is assumed to be subject to constant change and therefore should be requested anew every time it is required.

7.5 Mapping models to objects in RefSim

The nature of object-oriented design requires that each single component in the system be identified as a single object. For the general case of object-oriented software design, the object identification phase can be difficult, but for simulation software there is an intervening modelling phase which eases this problem. Once a set of models has been identified, their translation into objects is straightforward.

7.5.1 Component model design

A component model should be a self-contained description of that component, containing only those properties and functions which are intrinsic to that component. Derivations of the most important RefSim models are given in Chapter 9 but as an example of the model identification process consider the RefSim *Room* model:

A room contains a volume of air with some absolute humidity (the ratio of water vapour mass to dry air mass) and temperature. If it is assumed that the air is well mixed, then one humidity and one temperature may serve to adequately describe the state of the room. If it is assumed that the volume of the room and the air pressure in the room are invariant, then the state of the room may be changed only by flows of heat and water (generally in the form of vapour). An initial description of the room therefore has one fixed property (Volume) and two dynamic properties (Humidity and Temperature) whose current values depend upon their past values and upon the net flows of water and heat into the room. Using the acquisitive convention, these flows are assumed to pass from other models into the room — flows in the other direction have a negative sign.

Consider now the possibility of extreme conditions in the room. Heat flows may affect the room temperature and water flows may affect the absolute humidity. Assuming the availability of exact thermodynamic and psychrometric properties for air, extreme temperature conditions could be reached at absolute zero (safely outside the range of interest of this project), and extreme humidity conditions could be reached at both zero and saturated humidity. Zero humidity conditions should be dealt with by models which calculate the flows into the room (the interface models discussed in section 7.5.2) as zero humidity is a natural extension of low humidity, but saturated humidity represents a possible discontinuity in room behaviour and so must be dealt with by the room model.

One way to deal with saturated humidity in a room would be to allow the air to become supersaturated. One room model in RefSim (implememented as the model *RADSRoom* and detailed in Appendix 5) does just this. *Room* is derived from

an alternative approach. Where warm humid air passes into a real room, saturated humidity may be exceeded for a time. When this happens, water droplets will precipitate and a fog will form in the air, or "drip" (or frost, depending upon room temperature) will form on the structures in the room. *Room* contains an additional property (WaterMass) which represents the amount of water present in the room in either liquid or frozen form. If the air is supersaturated, water vapour condenses (releasing its latent heat into the air) and WaterMass increases. If the air is subsaturated and WaterMass is positive, then water evaporates (taking its heat of evaporation from the air) and WaterMass decreases. The rates at which condensation and evaporation take place depend upon the extent of super- or sub-saturation respectively and upon the room TimeConstant (the ratio of room volume to the total air flow rate in the room). The temperature of the WaterMass is assumed to equal that of the room and the heat capacity of the WaterMass is considered when evaluating the rate of change in room temperature. This treatment of the saturation humidity problem is reasonable if the WaterMass is well distributed throughout the room, with no "puddles".

With this analysis complete, the *Room* model may be described by the diagram shown in Figure 7-1. Dynamic properties are:

Humidity, Temperature, WaterMass

The one fixed property is:

Volume

One property which may be variable but not dynamic is:

TimeConstant

(due to the approximate nature of the WaterMass part of this model, TimeConstant is actually a fixed variable in the *Room* implementation so that it is not necessary for *Room* to add up all of the air flows every time it is evaluated)

A number of other properties are derived from these which may be useful either to other models, or as output to the user:

RelHum, Heat_Load, Water_Load

(RelHum is the relative humidity in the room, Heat_Load and Water_Load are the sums of all positive Heat and Water flows into the room respectively)

The most important things to note about this model are not the features which it has but the features which it does not have. The *Room* model does not explicitly consider the possible existence of walls, ceilings, floors, doors, ventilation, structures,

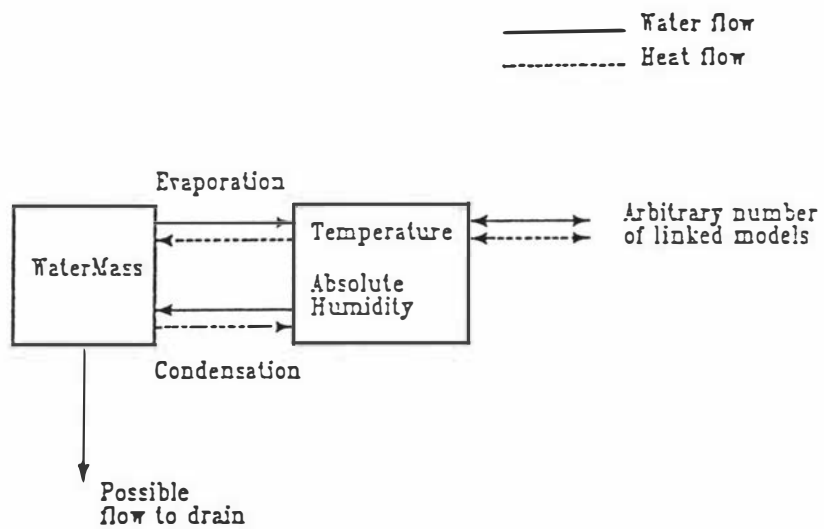


Figure 7-1 Conceptual diagram of the RefSim *Room* model.

machines, people, product, drains, or any other sort of component model. All such models are only considered in the abstract sense as providers of heat and water flows to the room. If any other models are connected to the room in some way, then their names will be provided to the *Room* model so that it knows the models from which to request heat and water flows.

Properties like:

contains_food_product, is_refrigerated, has_concrete_floor

are not intrinsic properties of a room but relationships which the room may have with other components of the system (food product, refrigerant evaporators, concrete floors). As such, they are handled by the message-passing mechanism which links models together rather than by the internal description of the model itself.

By separating the concerns of the model itself from the relationships that it may have with other models, the model can be described at a higher level of abstraction and can therefore be more general. Instead of having several different types of room depending upon contents or usage (after the manner of RADS, described by Cornelius (1991), for instance), there can be one room model which is capable of a variety of relationships with other models.

In some cases, it may be difficult to identify the model to which a particular property should belong. This may indicate that an additional model is required. Consider the heat transfer coefficient on the surface of a body. This is not a property of either the body or the surrounding fluid, but of the interface between the two because heat transfer coefficient is a function of both fluid and body properties. This analysis would normally indicate that the body-fluid interface should be the subject of a separate model, and that approach would be compatible with the RefSim system should such a detailed study be necessary. Efficiency considerations preclude the proliferation of models when large and complex systems are to be simulated, however, so a compromise has been made and the RefSim *ThermalObject* model includes heat transfer coefficient as one of its intrinsic properties. This has the

disadvantage that although the ambient temperature around a *ThermalObject* may change during a simulation, the heat transfer coefficient may not, thereby reducing the generality of the *ThermalObject* model.

7.5.2 Model types

One way in which models may be differentiated is by distinguishing item, interface, instrument and environment models.

Item models have variable and/or dynamic properties. Some or all of these property values are usually of interest to the user. Item models may be linked to zero (in practical cases, one) or more other models. The *Room* and *ThermalObject* models are examples of item models.

Interface models have no dynamic properties of their own. They are linked to exactly two other models, each of which represents one side of the two-sided interface. Flows may occur through an Interface model and the extent of those flows is determined by the states of the models on either side and by any variable properties belonging to the interface model. *Door* is an example of a model which may form an interface between two room models.

Instrument models are used in RefSim to group process controllers. Instruments differ from other models in that they transfer information rather than some conserved quantity like mass or energy. This means that they may be linked to other models without those models necessarily being linked to them, whereas a linking mismatch would be a logical error for Item or Interface model types.

Environment models are simple models which are completely passive. Their properties may be invariant, or they may change according to some schedule, but they do not change in response to other models. Environment models are used to represent ambient conditions, fixed heat flows (such as those from machines within a room) and to act as dummy models to represent any parts of a system which are not being fully simulated. Environment models are also ideal for setting up test-rigs in which to validate individual models.

7.6 Communication between models

7.6.1 The data request

A model can obtain data from a target model by requesting that the target provide that data. Should the target model be able and willing to supply the specified data to the requesting model, the data is returned as requested. If the data cannot be returned as requested due to non-availability of the data, or because the requesting model type is inappropriate (for example, it may be inappropriate for a compressor to request a room temperature), such a request causes an error report. If the requested data is temporarily unavailable, or the requesting model is temporarily inappropriate, then a safe default value is returned. An example of the last case would occur if a compressor could operate on one of several suction lines and one of the lines to which it was not connected at the time requested a mass flow rate. In this situation, the compressor would return 0.0 kg/s as the mass flow rate taken from that suction line.

When a model receives a data request, it may sometimes have to use the identity of the requesting model to indicate the correct value to be returned. For example, if a pipeline receives a request for pressure it must decide the end of the pipe to which the requesting model corresponds and return the pressure at that end of the pipe. If an evaporator receives a request for mass flow rate, then it must decide whether the requesting model is the refrigerant source or sink (in which case it should return the refrigerant flow rate with the appropriate sign) or the application (in which case it should return the rate of water vapour flow onto the evaporator surface).

7.6.2 Allowable data transfer

The following types of data can be requested from a model:

Time	-	seconds, only from mTime and mUserTime models.
Temperature	-	degrees Celsius
Pressure	-	Pascals
Heatflow	-	Watts
Massflow	-	kg/s
ControlVar	-	unspecified dimensionality, only from controllers (the range depends upon the MinOutput and MaxOutput parameters of the controller)
RelHumidity	-	fraction 0 → 1
AbsHumidity	-	wt/wt
State	-	fraction liquid 0 → 1
None	-	used for error cases.

In general, the data which models may request from each other correspond to values which may be measured in a real refrigeration plant. Heatflow is an exception to this guideline which is necessary if thermal models (as developed by Cleland, 1983) are implemented within RefSim.

The State data type serves a dual role. For refrigeration system component models, it represents the liquid fraction of the refrigerant passing between the requesting and the target models. For application component models, State represents the liquid fraction of any water flow (State = 1.0 for liquid water, State = 0.0 for water vapour in air).

The ControlVar data type represents the output of a controller. Where the controller may have multiple outputs, it is responsible for deciding which value to return depending upon the identity of the requesting model.

This list may be extended without deviating from the RefSim design, but the data types available already are sufficient for the level of detail required to simulate complete refrigeration plants and their applications.

7.7 Required level of component model detail

Refrigeration system components can be (and have been) modelled using many different levels of detail, and so the RefSim environment does not restrict models to particular levels of detail or complexity. To maximise compatibility between models of different complexity, it is desirable that each model attempt to provide a full interface as described in section 7.6.2 in so far as that is appropriate to the model type (e.g. it is inappropriate to request Pressure from a food product model), even if some of the values returned by the model are approximate. For example, a thermal model of a refrigerant evaporator should make some estimate of the refrigerant mass flow rate passing through it in case the evaporator model should be linked to a hydrodynamic header or pipeline model which may require that data. If a model is completely unable to estimate a value for a requested data item, then it should produce an error report to indicate that it is incompatible with the model requesting that data.

No restrictions are placed upon the level of detail used in model evaluation. Any model may be described by any number of algebraic or ordinary differential equations. It would be more difficult to describe models by the use of partial differential equations as no partial differential equation solver is included in the current set of simulation utilities (section 7.11). PDE models would not, however, be incompatible with the RefSim environment.

7.8 Model hierarchy

Object oriented design encourages the development of hierarchies of objects where descendant objects inherit the features of their parents and add more features. Consideration of the models to be included in an industrial refrigeration system simulation produced the model hierarchy shown in Figure 7-2.

In the model hierarchy, descent is shown by indentation. The root model (Model) is found at the top and is the leftmost model. The other model sub-hierarchies are shown to the right and below. Models at the same level of indentation share the same level in the hierarchy.

The full model hierarchy was not identified during the early design stage, but instead it developed as models were added to the system. In fact, the hierarchy should not be regarded as complete, and additional models may easily be added by descending from whichever existing model they most closely resemble.

7.9 Model interface specification

In RefSim, all models are derived from the base class *Model*. *Model* provides three variables and six methods of which four may be re-implemented by its derived objects:

Variables:

name	the name of the model instance.
LinkedObjects	a list of other models which are linked to this one.
Evaluated	a flag which is TRUE if the model has been evaluated in the current time step and FALSE otherwise.

Virtual methods (may be re-implemented by derived objects):

GetValue	which takes a model and variable type as parameters. GetValue first calls the Evaluate method and then the

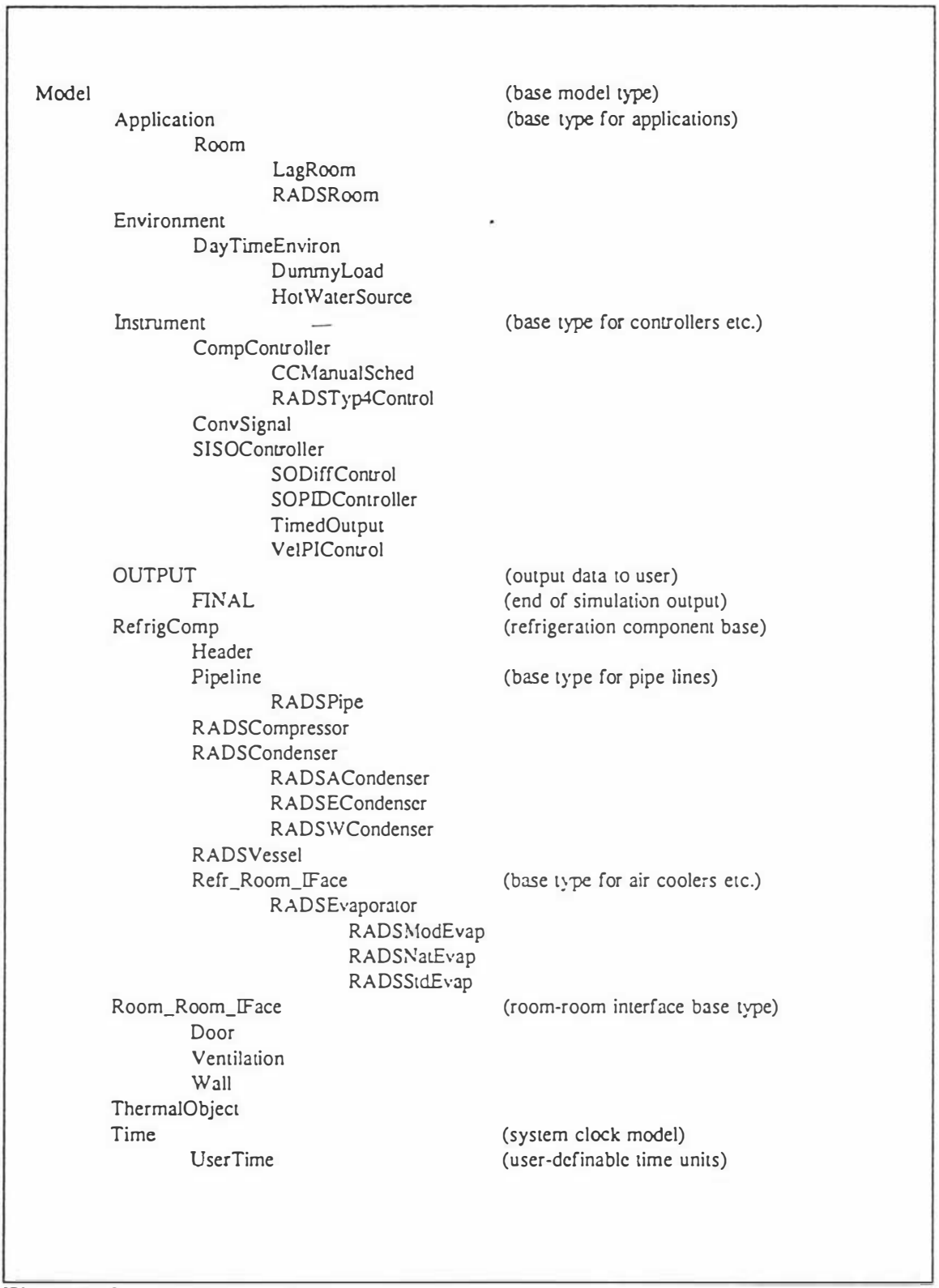


Figure 7-2 RefSim model hierarchy.

current model is prepared to provide the specified variable type to the querying model, it returns the most up to date value of the specified variable.

Event — which executes a discrete event for that model. Normally this occurs because it was scheduled earlier by the model itself.

Evaluate — if the Evaluated flag is not already set, Evaluate sets the Evaluated flag and then calculates the current values of the model state variables.

Initialise — which sets up initial default values, reads in data from an input file via a centralised parser (section 7.11) and schedules any initial events. This is only called once, at the start of the simulation, for any given model instance.

Methods which may not be re-implemented by derived objects:

NewEvaluation — which prepares for a new evaluation by reseting the Evaluated flag.

DummyEvaluate — which sets the Evaluated flag for this model and all of the models which are linked to this one.

Some derived models provide additional methods, but these are only for use by the model itself, except for the case of OUTPUT and its derived models.

7.10 Model linking

As each model is created and initialised, it is added to a model list which identifies the models to be evaluated during the simulation. At each ODE solver evaluation step, each model in the list has its Evaluate method called.

When a model is initialised, it registers each of its dynamic variables with the simulation system for solution. Registration adds the variable to a list of dynamic variables which are to be integrated.

Individual variables within a model may be registered for output with the OUTPUT model during initialisation. On the occurrence of any OUTPUT event, the values of all registered variables are written to an output file.

Lastly, models may be linked with other neighbouring models. Each model maintains a list of those models to which it is linked as provided by the user in the input data. Once all of the models have been read into the environment, the model names are cross-referenced with the model object instances, and those instances are recorded in the linked model list for each model. Upon evaluation, each model runs through the list of models linked to it and obtains whatever data it may require by calling the GetValue method of each of those models before doing its own calculations. If a variable value is requested from a model which has not yet been evaluated during the current time step, the model carries out its own evaluation before returning the variable value to the requester. The Evaluated flag is set before the GetValue methods of linked models are evaluated so that if any of those models requests a variable value from the current model it will receive the value as it was after the last evaluation. Groups of models are therefore always evaluated sequentially.

In principle, it may be necessary for groups of models to be evaluated in parallel rather than in sequence because their properties are most accurately expressed by a set of simultaneous equations. Simultaneous model evaluation would be difficult to carry out without violating the principle of encapsulation outlined in section 7.4.1. In practice, the error incurred by evaluating models sequentially rather than simultaneously is expected to be small as long as the changes in model properties between evaluations are small.

The linking system is sufficiently flexible so that once a model is declared to be linked to a number of other models, the models from which it actually requests data and to which it will provide data may vary with time or with state. For

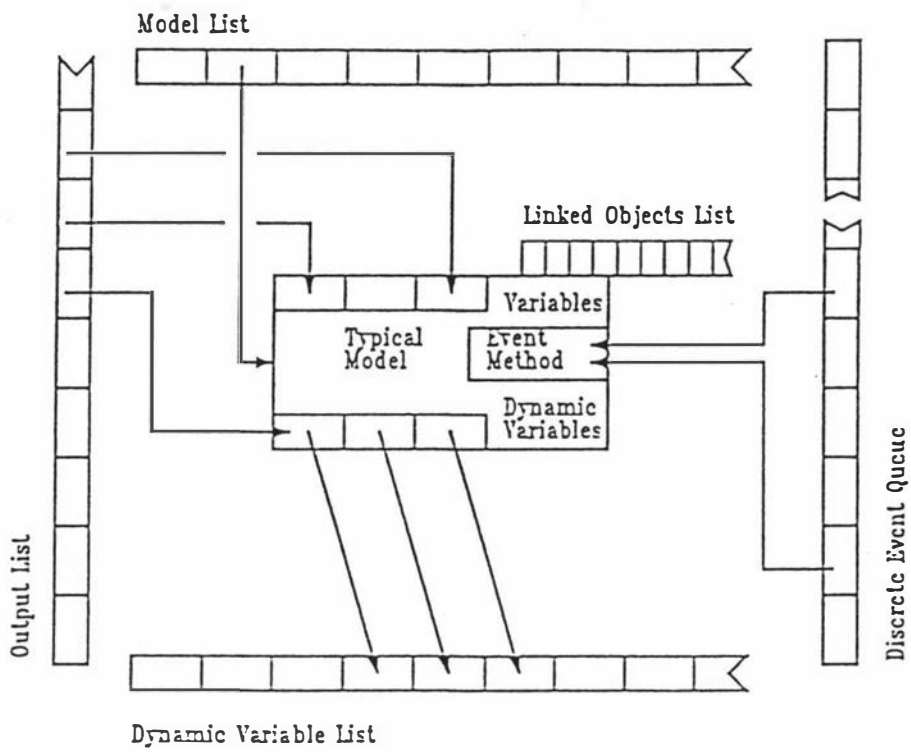


Figure 7-3 Links between a model and the simulation environment data structures.

example, having linked a compressor to two different suctions, the choice of which suction to use may be made by the compressor controller.

Figure 7-3 shows the possible model linkages in diagrammatic form. The number of linkages in which a model may participate is limited only by available storage.

7.11 Access to simulation utilities

Unlike component models, the simulation utilities do not map easily to objects, and so they are designed in a conventional procedural manner. The implementation of these simulation utilities is described in section 8.2.

The RefSim input language is defined in Appendix 2 using the Extended Backus-Naur Form described by Wirth (1982). Two parsers are provided for reading the simulation input data described in that language. *Parser* reads the overall input file, creating the model instances specified by the input data and executing the Initialise method each new model instance. The *Initialise* method is responsible for reading the data particular to that model instance from the input file. To do this, the *Initialise* method may call the model data parser *MParser*, which is sufficiently general to read the data required by almost any type of model. *MParser* checks that the input parameters for each model are correct but it leaves data value verification to the *Initialise* method.

Data types which may be input by *MParser* include scalars (such as initial temperatures, pressures etc), strings (such as controller names), tables (such as time-output value profiles for TimedOutput or polynomial coefficients for the RADSEvaporator derived classes), linked model lists and output variable lists. Other data types may be added while remaining compatible with the existing design.

An example containing part of a RefSim input file is shown in Figure 7-4. The input file from which the example was taken was automatically generated from a RADS .APP file (described by Cornelius, 1991) which accounts for the somewhat cryptic model names. The example application is the batch freezer Room 11 at the

Alliance Ocean Beach Plant, Bluff, New Zealand (code-named *zfrm11*) which freezes lamb carcasses (code-named *zfrm11Prod1*). The room has a door (*zfrm11Door1*), and that door has an external environment (*zfrm11Door1E0*) which describes the conditions in the corridor outside the room. The data values used in the example are discussed in section 10.3. A complete input file for a water chiller simulation is shown in Figure 10-3.

An ordinary differential equation solver is provided for the solution of dynamic variables. The ODE solver accesses the Evaluate method in each active—model instance, so models need not have access to the ODE solver itself. Control over the behaviour of the ODE solver and its associated step size controllers may be exercised by the user upon starting a RefSim simulation.

A discrete event scheduler is provided for the execution of discrete events during simulation. Models may schedule time events by calling the scheduler AddEvent procedure with the event time and handler model as parameters. Time events may be scheduled only during initialisation or when a previously-scheduled event has occurred. Models may schedule state events during evaluation by calling the AddStateEvent procedure. AddStateEvent sets the scheduler StateChanged flag to restart the current ODE solution step after the state event has been scheduled. The behaviour of the discrete event scheduler may be controlled by the user upon starting a RefSim simulation (Appendix 7).

A library of refrigerant, air and water thermodynamic and transport property correlations, and air-water psychrometric correlations is provided for the use of all models.

The full range of object-oriented data structures used by the RefSim environment is available to any component model which may require them.

```
{       This example contains some of the models which comprise the RefSim description of
        Alliance Ocean Beach Plant batch lamb carcass freezer room No.11.  Text enclosed in
        braces is ignored by RefSim and is used for comments. }

MODEL zfrm11Prod1 ( ThermalObject )              { Product model description, see Chapter 4}
        Cs      =       2.055744E+6              { Frozen heat capacity J/m3K }
        Cl      =       3.347898E+6              { Unfrozen heat capacity J/m3K }
        Hf      =       2.723043E+8              { Enthalpy at Tf J/m3 }
        ks      =       1.450941853              { Frozen thermal conductivity W/mK }
        kl      =       0.462051696              { Unfrozen thermal conductivity W/mK }
        h       =       8.396989194              { Surface heat transfer coefficient W/m2K }
        Tf      =       -1.15000000              { Initial freezing temperature °C }
        Tff     =       0.000000000              { Section 4.10 }
        V       =       0.013430339              { Volume of the body m3 }
        A       =       0.335000000              { Surface area of the body m2 }
        X       =       0.071000000              { Critical depth m }
        N       =       2.500000000              { Heat load shape factor }
        E       =       1.480000000              { Freezing time shape factor }
        Tstart          =       25.00000000      { Initial temperature of the body °C }
        Multiple        =       2700.000000      { Number of bodies in the application }
        VapDiffuRes     =       41.02866531      { Vapour diffusion resistance, section 9.7}
        MassTrCoef      =       0.00924097253    { Mass transfer coefficient, section 9.7}
        TABLE Load [    12.50000000     30.50000000     { Load/unload times, hours }
                        36.50000000     54.50000000
                        60.50000000     78.50000000
                        84.50000000     102.5000000     ]
        RepeatInterval  =       24.00000000      { Interval between re-uses of this model,
                                                   hours }
        [ zfrm11 ]                               { Model to which this one is linked }
        < Temp x >                               { Variables to be written to the output file }
END zfrm11Prod1

MODEL zfrm11Door1E0 ( Environment )              { Exterior environment of zfrm11Door1 }
        Temperature     =       10.000000000     { Temperature °C }
        RelHumidity     =       0.750000000      { Relative humidity, fractional }
        [ zfrm11Door1 ]                          { Model to which this one is linked }
END zfrm11Door1E0

MODEL zfrm11Door1 ( Door )                       { Loading door, see section 9.4 }
        OHTC            =       0.000000000      { No heat transfer when the door is closed}
        Height          =       2.000000000      { Doorway height m }
        Width           =       1.500000000      { Doorway width m }
        FirstOpen       =       7.500000000      { Time first opened in the day, hours }
        LastClose       =       16.00000000      { Time last closed in the day, hours }
        FractionOpen    =       0.500000000      { Fraction of time open}
        ProtFactor      =       1.000000000      { Door protection factor }
        [ zfrm11 zfrm11Door1E0 ]                 { Models to which this is linked }
END zfrm11Door1
```

Figure 7-4 Extract from a RefSim simulation input file.

Chapter 8: RefSim Implementation

This chapter describes the refrigeration system simulation environment implementation which follows the design described in Chapter 7. Model implementation details are described in Chapter 9. The complete source code for RefSim is provided in Appendix 8 (on diskette).

8.1 Implementation goals

It was desirable that the RefSim implementation achieve several goals:

1 Clarity. In accordance with the principles outlined in section 7.1.1, one of the best ways of describing a model is in terms of a computer programming language. A model implemented in this way is defined unambiguously (because the syntax of the language is unambiguous) and it may be evaluated automatically without error. While a computer program cannot be ambiguous, it may nevertheless be unclear and difficult for a human reader to understand. It was important to ensure that a high level of code clarity was maintained in the RefSim implementation.

2 Accuracy. Any model is an approximate representation of a real phenomenon. The process of model evaluation should not degrade the accuracy of the models any further.

3 Easy maintenance and extension. The addition or modification of models and system utilities should require little effort. To some extent this goal relies upon goal 1, but easy maintenance and extension are further enhanced if the implementation utilizes the same concepts of encapsulation, inheritance, polymorphism and message passing as did the design. In particular, use of encapsulation in the implementation

allows a modification to be made to one part of RefSim without that modification affecting the rest of the program.

4 Minimal use of computer resources. It was desirable that the RefSim implementation execute quickly on the available computer equipment and that it do so within the available memory. This would minimize development and testing costs: In the context of this project, goal 4 had a lower priority than goals 1, 2, and 3 because the time spent executing the program was likely to be a small fraction of the development time even if the execution was very slow.

8.2 Choice of implementation language

The object-oriented design described in the preceding chapter could have been implemented in almost any computer programming language. The majority of simulation environments discussed in Chapter 2 were implemented in FORTRAN (ANSI, 1978). Eldredge *et al* (1990) showed that this was true over the whole simulation field, but noted that most authors "attribute this to the fact that the choice of language is primarily based upon the availability of the language to the user and the user's knowledge of the language". For these reasons of availability and user's familiarity, a preliminary implementation of the RefSim environment was developed in FORTRAN-77.

8.2.1 FORTRAN-77 implementation

The FORTRAN-77 version was implemented to the stage where it could simulate simple refrigeration applications, but not an entire refrigeration plant. The FORTRAN-77 language lacked both a method for dynamic memory allocation and a *pointer* data type so a memory pool was simulated using two large one-dimensional arrays, one of type INTEGER*4 and one of type REAL*4, which were superimposed using the EQUIVALENCE statement. This data structure was then used like

dynamic memory, with each object instance being allocated a part of the structure for its exclusive use. INTEGERs were used as pointers to reference other object instances in the system and data values were stored as REALs at specified offsets from the start of the object. Each model had a key number to identify its type and an index (pointer) to the next model in the evaluation chain. Model evaluation proceeded by moving through the memory pool from model to model, calling the model evaluation subroutines indicated by the key number.

Even with the use of constant PARAMETER values to indicate offsets and after taking advantage of extensions to the FORTRAN-77 standard which allowed variable names with mixed case and more than six characters, the model descriptions became unclear. The essence of each model description became lost in the circumlocutions required to reference the values of its properties. For example, a reference to the REAL value (stored in the array *mr*) which indicated the pressure at end 1 of a pipeline from within the pipeline model itself (*Self*) had the form:

```
mr(Self + Pressure + 1)
```

Lack of clarity was not the only problem. Any error in the values of *Self* or *Pressure* could result in the wrong data value being referenced or modified. The size of the data stored for each model type also had to be calculated carefully to avoid object instances overlapping in the memory pool. The PARAMETER values and object sizes had to be checked every time a model was modified by adding an additional variable. This problem was made even more difficult by the fact that most models could have arbitrarily long lists of indices to indicate the other models to which they were linked, and so the storage required for each instance of the same model type could be different. Development of the FORTRAN-77 implementation was halted when these difficulties became too taxing.

Some consideration was given to implementing a pre-processor for the FORTRAN-77 compiler to automate the mechanical processes of dynamic memory allocation, checking offsets, object sizes and references. Several such approaches

have been discussed by Kreutzer (1986, Chapter 4). It was decided, however, that these problems had already been solved by a number of other existing programming languages and that the best approach was to use a more suitable language.

8.2.2 TopSpeed Modula-2 implementation

The development of object-oriented design techniques has led to the development of programming languages in which object-oriented designs could be implemented directly. Object-oriented programming languages may be divided into two major groups: pure and hybrid. The advantages of each approach have been discussed by Cox (1986, Chapter 1).

In a pure object-oriented language, each and every component of the language is an object. The Smalltalk language (Lalonde and Pugh, 1990) is the best developed example of this approach. The principles of software development which were initially established for Smalltalk were found to be applicable to any object-oriented programming language so most modern object-oriented languages and the software implemented in those languages derive their conceptual bases from this root. The use of Smalltalk itself or of another pure object-oriented language for developing a refrigeration system simulation environment was precluded for efficiency reasons because pure object-oriented systems generally require substantial computing resources (Cox, 1986, p.7). It was preferable to reserve these resources for executing the simulation itself.

Hybrid object-oriented languages have attempted to combine the efficiency of conventional languages (Pascal, Modula-2, and C, for example) with the clarity and expressive power of object-oriented techniques. Hybrid languages achieve greater efficiency than pure object-oriented languages by using objects for only those parts of the program which require them, while retaining procedural representation and conventional data types where they are suitable. In Smalltalk, for instance, an object is always implemented by describing it in terms of other objects, but a hybrid language provides the additional option of implementing an object in terms of a

procedural description in the base conventional language. This is advantageous in the context of a simulation environment, because as well as providing greater execution efficiency, it allows each model to be described in a procedural language which is familiar to many modellers.

The Modula-2 programming language (Wirth, 1982) was found by the present author to provide significant advantages in the development of large software systems due to its utilisation of the *module* concept. TopSpeed Modula-2 by JPI (1991) provided extensions to the base language which made it a suitable hybrid object-oriented language for implementing the RefSim environment. The other language considered for the purpose was C++ (Ellis and Stroustrup, 1990), but TopSpeed Modula-2 was preferred because the underlying Modula-2 language was very similar to the popular Pascal language.

RefSim was developed using the MS-DOS version of TopSpeed Modula-2 Version 3.01. It was run on IBM PC and compatible computer systems under MS-DOS or compatible operating systems.

8.3 Program structure

The RefSim implementation may be divided into the same two parts as its design: simulation utilities and models. The way in which these two parts fit into the physical program structure is shown in Figure 8-1. In this diagram, each arrow indicates that the procedure at the tail of the arrow uses the procedure at the head of the arrow. The "uses" relationship is implemented when a high level procedure calls a lower level procedure.

Figure 8-2 shows the simulation utilities which are available for use by all model methods. These utilities provide commonly-used thermodynamic and transport property correlations, commonly-used data structures, discrete event scheduling and random number generation.

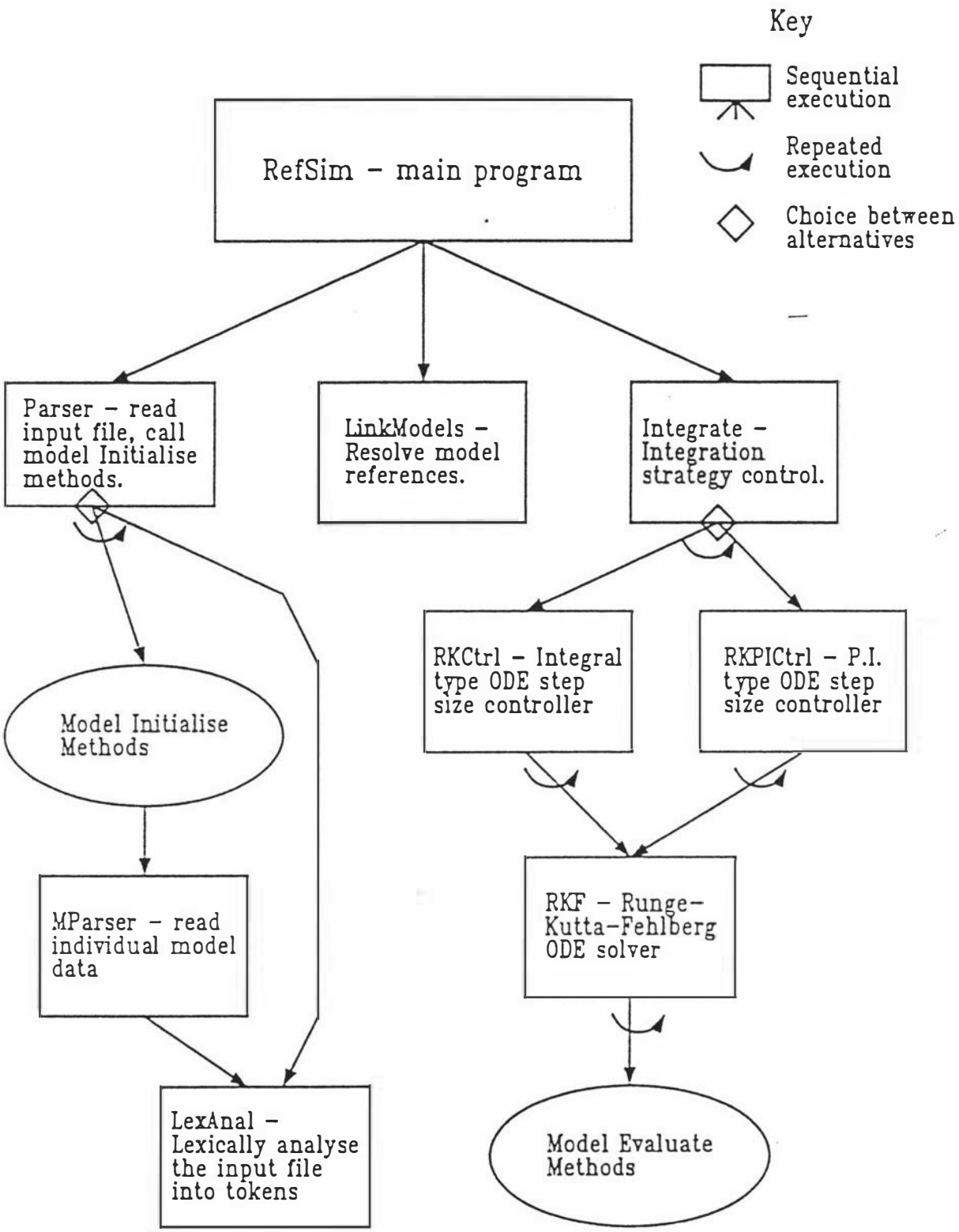


Figure 8-1 RefSim program structure diagram.

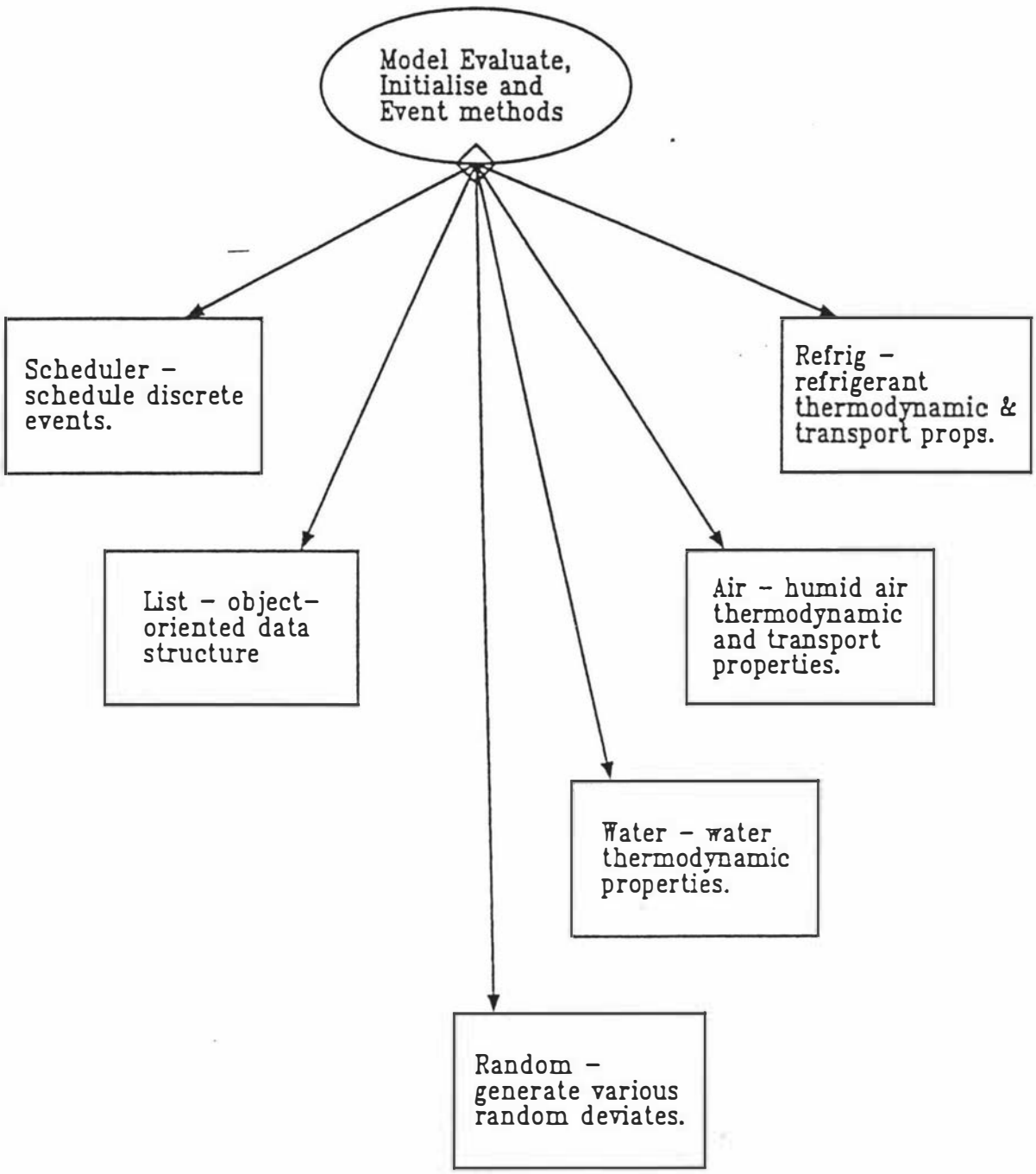


Figure 8-2 Simulation utilities which may be called by model methods.

The simulation utilities shown on these diagrams are described in the remainder of this chapter. The individual model implementations are described in Chapter 9.

8.4 Simulation environment

The object-oriented simulation environment design was followed closely in the implementation. Object-orientation provided no benefit in the implementation of the simulation utilities, so these were written as procedural code.

A range of simulation utilities is available to any model running within the RefSim system. The availability of these features made each model description simpler, smaller and more reliable. A further advantage of centralising the simulation utilities was that they were only implemented once and so more time could be spent ensuring that each utility was of a high standard. The simulation utilities are described first so that the subsequent model descriptions may be more easily understood. The basic data structures used in RefSim which are available to models are also described here.

The most common data type used by the models in RefSim is the LONGREAL number. The LONGREAL is a floating point number which is capable of expressing values in the range $\pm 2.3\ (10^{-308})$ to $\pm 1.7\ (10^{+308})$ with a precision of approximately 15 decimal digits. The other data types which are provided directly by TopSpeed Modula-2 are described by JPI (1991, pp.15-23).

Diagrams of the more sophisticated data structures are provided in section 9.1 together with an example of how a simple simulation is represented within RefSim.

8.4.1 The simulation generator App2Ref

This was the only RefSim simulation utility to be written as a separate program.

A number of refrigeration plants have been modelled using the RADS dynamic simulation system (Cleland, 1985*b*, Cornelius, 1991). To save the need to re-enter the data for these plants from scratch, the program *App2Ref* was written to translate RADS application files (as generated by the RADS program APPLICS) directly into the RefSim input language. This process was facilitated by the availability of all the major RADS models within RefSim.

App2Ref accepts a single parameter which is the name of a RADS application file and it writes the resulting RefSim input language file to the standard output from where it may be redirected to a file for inclusion in a RefSim simulation. *App2Ref* is started with a command of the following form:

```
C:\> App2Ref freezer.app >freezer.ref
```

which produces a RefSim input file containing the many model descriptions which together represent the application characterised by the file "freezer.app". The text "C:\>" is an operating system prompt. The text ">freezer.ref" indicates that output is directed by the operating system into the file "freezer.ref".

App2Ref does not generate a complete RefSim input file. Header and trailer records must be appended to the start and end respectively of the *App2Ref* output file to provide the simulation control parameters. Additional application and refrigeration component model descriptions may also have to be added to create a complete RefSim input file.

No program was provided for converting the RADS engine-room file into a RefSim simulation input file because the work required to do that task by hand was comparatively small.

8.4.2 Fluid thermodynamic and transport property correlations

All fluid thermodynamic and transport property calculations in RefSim are carried out in the modules Refrig (for refrigerant properties), Air (for air properties and humidity calculations) and Water (for water properties). The interfaces to these modules are clearly defined so that alternative procedures could easily be written in the event that the correlations described below were not satisfactory. These interfaces are shown in Figures 8-3, 8-4, and 8-5. A change to any of the correlations would only require re-compilation of the applicable property module and re-linking of the RefSim program.

The refrigerant thermodynamic property correlations used in RefSim were those described by Cleland (1986*a*), which were in turn based upon those of Chan and Haselden (1979*a,b,c*). The correlations of Chan and Haselden were based upon thermodynamic theory and experimental data compiled by a working party of the International Institute of Refrigeration and they were found to accurately represent the available experimental data. Cleland's correlations produced results which were within ±0.25% of the Chan and Haselden results, but they were much faster to evaluate, making them a great deal more suitable for use in RefSim. The polynomial equations are evaluated in factorized form to maximise both speed and accuracy. Cleland's correlations covered R12, R22, R114, R502 and R717, so the current implementation of RefSim is limited to systems using these refrigerants. Should they be required, polynomial curve fits for additional refrigerants may be derived in the manner described by Cleland (1986*a*).

The interface shown in Figure 8-3 uses an enumerated type (Refrig.Type) variable containing the refrigerant type (R12, R22, R114, R502, R717 or SecondaryRefrig) to indicate which refrigerant properties are required. SecondaryRefrig is not an appropriate refrigerant type to be used when accessing these property correlations.

Air property calculation procedures were derived by assuming that dry air behaved as an ideal gas and then adding modifications for humidity to the density,

PROCEDURE EvapTemp (ref : Refrig.Type; p : LONGREAL) : LONGREAL;
Returns evaporation temperature at pressure, -p-

PROCEDURE EvapPress (ref : Refrig.Type; t : LONGREAL) : LONGREAL;
Returns saturation pressure at temperature, -t-

PROCEDURE LiquidEnthalpy (ref : Refrig.Type; t : LONGREAL) : LONGREAL;
Returns saturated liquid enthalpy at temperature -t-

PROCEDURE VapourEnthalpy (ref : Refrig.Type; t : LONGREAL) : LONGREAL;
Returns saturated vapour enthalpy at temperature -t-

PROCEDURE LiquidSpecHt (ref : Refrig.Type; t : LONGREAL) : LONGREAL;
Returns saturated liquid specific heat at temperature -t-

PROCEDURE VapourSpecHt (ref : Refrig.Type; t : LONGREAL) : LONGREAL;
Returns saturated vapour specific heat at temperature -t-

PROCEDURE SupEnthalpy (ref : Refrig.Type; t1,t2 : LONGREAL) : LONGREAL;
Returns superheated vapour enthalpy at saturation temperature -t1- and actual temperature -t2-

PROCEDURE SpecificVolume (ref : Refrig.Type; t : LONGREAL) : LONGREAL;
Returns the specific volume of saturated vapour at temperature -t-

PROCEDURE SupSpecificVolume (ref : Refrig.Type; t1,t2 : LONGREAL) : LONGREAL;
Returns the specific volume of superheated vapour at saturation temperature -t1- and actual temperature -t2-

PROCEDURE Gamma (ref : Refrig.Type; t1,t2 : LONGREAL) : LONGREAL;
Returns value of γ for saturated cycle enthalpy change in compression from saturated temperature -t1- to -t2-

PROCEDURE SupGamma (ref : Refrig.Type; t1,t2,t3 : LONGREAL) : LONGREAL;
Returns γ for superheated cycle enthalpy change in compression from superheated vapour with saturated temperature -t1- and actual temperature -t2- to vapour with saturated temperature -t3-

Figure 8-3 Refrigerant property library interface. *ref* is the refrigerant type, e.g. R717. Temperatures in °C. Pressures in Pa. Enthalpies in J/kg. Specific volumes in m^3/kg.

heat capacity and enthalpy correlations. Antoine's equation (Reid *et al*, 1987, p.208) was used to calculate the saturated water vapour pressure in air and hence the saturated humidity.

PROCEDURE Density (Temp,AbsHum : LONGREAL) : LONGREAL;
Returns the density of humid air (kg/m3) at temperature -Temp- (deg C) and absolute humidity -AbsHum- (kg/kg).

PROCEDURE Enthalpy (Temp,AbsHum : LONGREAL) : LONGREAL;
Returns the enthalpy of humid air (J/kg) at temperature -Temp- (deg C) and absolute humidity -AbsHum- (kg/kg).

PROCEDURE HeatCapacity (Temp,AbsHum : LONGREAL) : LONGREAL;
Returns the heat capacity of humid air (J/kgK) at temperature -Temp- (deg C) and absolute humidity -AbsHum- (kg/kg).

PROCEDURE SatHumidity (Temp : LONGREAL) : LONGREAL;
Returns the saturated absolute humidity of air (kg/kg) at temperature -Temp- (deg C)

Figure 8-4 Air property library interface

Water property routines were derived by assuming constant heat capacities for water and ice. The latent heat of freezing caused a discontinuity at 0°C in the enthalpy correlation, but in the heat capacity correlation, the latent heat was spread over the range from 0.0 to -0.1°C to avoid a singularity at 0°C.

The air and water property correlations usually predicted results which were within ±1% of tabulated data (e.g. that of Perry and Green, 1984, Chapter 3). The

PROCEDURE Enthalpy (T : LONGREAL) : LONGREAL;
Returns the enthalpy of water (in J/kg) at temperature T / deg C

PROCEDURE VapourEnthalpy (T : LONGREAL) : LONGREAL;
Returns the enthalpy of water vapour (in J/kg) at temperature T / deg C

PROCEDURE VaporisationHeat (T : LONGREAL) : LONGREAL;
Returns the heat of vaporisation of water (in J/kg) at temperature T / deg C

PROCEDURE HeatCapacity (T : LONGREAL) : LONGREAL;
Returns the heat capacity of water (in J/kgK) at temperature T / deg C. If T is close to 0.0 deg C, the latent heat gets included as if it were spread over a small temperature range close to 0 deg C.

Figure 8-5 Water property library interface

water heat capacity correlation in the range 0.0 to -0.1°C was an exception to this guideline.

No general correlations were supplied to estimate the thermal properties of solids (metals, concrete and food product, for instance) due to the wide variety of materials which could be considered. Property correlations for these materials must therefore be included separately in any model which uses them. A detailed description of the enthalpy correlation used for the materials comprising ThermalObject models may be found in section 4.4.1. No other model which is currently implemented within RefSim uses solid material thermal properties.

The property correlation modules are very loosely bound to the rest of the system and adding another module for a specific purpose would not disrupt the existing structure. For example, if a group of multi-zone heat exchanger models were to be implemented then those models may require a common set of correlations for the heat capacities of metals. If this were so, then it would be convenient to implement a metal thermal property correlation module at the same time as the new models.

8.4.3 Ordinary differential equation solver

Many of the component models in the literature were described by ordinary differential equations (ODEs), so it was important that an ODE solver be available to all models. The ODE solver had to:

- solve an arbitrary number of ordinary differential equations.
- operate efficiently, with a minimum number of time steps required.
- control integration error to a user-specified tolerance.
- be robust enough to handle discontinuities in the integrated variables without failure.

The method chosen was a Runge-Kutta-Fehlberg (RKF) numerical ODE solver with fourth order error estimation. Local extrapolation was used to make it a fifth order method overall. The RKF method was originally described by Fehlberg (1969) and the implementation used in RefSim was derived from the FORTRAN code given by Cheney and Kincaid (1985, p.326). The RKF method requires six model evaluations per time step and produces both fourth and fifth order solutions. The difference between the two solutions is an estimate of the error in the fourth order solution. This difference can be used by a step size controller to control the integration error as described in section 8.4.4. In practice, the fifth order solution is often used for further calculations with the assumption that it will provide a better estimate of the true solution than the fourth order estimate, and this is done in the RefSim implementation. Press *et al* (1986, p.556) note that this rarely does any harm compared with using the fourth order solution and is often beneficial for equation systems of practical interest.

A stepsize-controlled Runge-Kutta method may take steps of three different sorts. The largest steps taken are *major* steps as dictated by an integration strategy controller (section 8.4.5). In a purely continuous simulation, one major step is taken between each time at which the user requires output. Within each major step, many *substeps* may be taken, with the size of each step decided by the integration step size controller (section 8.4.4). Unlike a major step, a substep may fail if the estimated integration error at the end of the step is too large. In this case, the substep is made again from its starting point with a smaller step size. Finally, within each substep the Runge-Kutta method makes a number of probes into the ODE solution space. Each probe requires the evaluation of all the functions in the equation system. The RKF method makes six such probes.

Each of the variables integrated by the ODE solver is of type *DynVarRec*. This variable type is defined in Figure 8-6.

Press *et al* (1986, Chapter 15) describe other ODE solution methods which were considered for use in RefSim. These included the Bulirsch-Stoer method and a range of multi-step predictor-corrector methods. The Bulirsch-Stoer method was

The DynVarRec variable type defined within RefSim is a record with eleven fields, each of which is a LONGREAL:

val	(value) The current best estimate of the dynamic variable value. May be read by the model which calculates the derivative of the dynamic variable.
der	(derivative) The current best estimate of the derivative of the dynamic variable. May be written by the model which calculates it.
valstart	The value of *val* at the start of the current substep. Used by the ODE solver when it is necessary to restart a substep, and by controller models which use this field of the system time variable to calculate a rough derivative of the controller input value.
err	The most recent difference between the fourth and fifth order integration error estimates for this dynamic variable. Used only by the ODE solver step size controller.
scal	The error value used by the ODE solver step size controller to decide whether or not the last step succeeded, and the next step size is the *err* field divided by the *scal* field. In RefSim, $scal = \|valstart\| + 1.0$, so the precision requirement is a modified relative error.
f1,f2,f3,f4,f5,f6	The values of *der* for each of the six probes in a substep. Used only by the ODE solver, which has the responsibility to transfer the value of the *der* field to the appropriate *f* field after each probe.

The only three fields which must be retained if the ODE solver is changed are *val*, *der*, and *valstart* as they are the only fields which are accessed by the models. The *err* field must also be retained (and a value supplied for it at the end of each substep) if the existing step size controllers are retained.

Figure 8-6 DynVarRec — the dynamic variable type

rejected due to its poor behaviour in the presence of discontinuities, while predictor-corrector methods were rejected due to their level of programming complexity when variable step size and integration error control features were included. The nature of the RefSim system is such that the addition of another ODE solver as an alternative to the existing RKF method would require very little effort in addition to designing and programming the alternative ODE solver itself.

8.4.4 Ordinary differential equation solver stepsize controllers

A choice of two ODE solver stepsize controllers is available in RefSim. The first is a conventional integral-action step size controller as described by *Press et al* (1986, p.558), with the addition of automatic safety factor retuning. The second is a proportional-integral-action step size controller as described by Gustafsson *et al* (1988).

8.4.4.1 Conventional controller with safety factor retuning

The integral-action step size controller is the default and has been used most extensively during test runs. The controller is described by equation (8-1).

$$h_0 = S h_1 \left|\frac{\Delta_0}{\Delta_1}\right|^{0.20} \quad \text{if } \Delta_0 \geq \Delta_1$$
$$= S h_1 \left|\frac{\Delta_0}{\Delta_1}\right|^{0.25} \quad \text{if } \Delta_0 < \Delta_1 \qquad (8\text{-}1)$$

where:

h_0	is the desired step size
h_1	was the step size during the last substep
Δ_0	is the desired substep tolerance
Δ_1	was the substep tolerance during the last substep
S	a safety factor, set by Press *et al* (1986) to be 0.9

If the desired substep tolerance is exceeded during a step, that step is repeated with the reduced step size, h_0. This is a commonly-used type of step size controller in adaptive step size ODE solution schemes.

The version implemented in RefSim has one important modification from that of Press *et al*: a step size change safety factor (S) retuner. Some difficulties were encountered during testing when the step size controller was found to perform poorly in certain parts of some simulations. The derivation of the controller assumes that the integration error varies smoothly with stepsize. While this may be true of single ODEs, it may not be true of large systems of ODEs such as those found in complex simulations, especially where there may be discontinuities caused by discrete events. This caused the controller to be too "optimistic" and sometimes increase the step size greatly, only to have the substep fail (perhaps several times) and have to reduce the step size back to a more appropriate level. RefSim was wasting many function evaluations during simulation runs due to this phenomenon, so the step size change safety factor was made a variable which was altered regularly using equation (8-2).

$$S_0 = \frac{S_1}{FracOK}\left(\frac{Nsteps}{Nsteps + TotFail}\right) \qquad (8\text{-}2)$$

where:

	$Nsteps$	is the number of successful substeps in this major step.
	$TotFail$	is the number of failed substeps in this major step.
	$FracOK$	is the desired fraction of successful substeps.

This maintained the number of successful substeps at a fixed fraction ($FracOK$) of the total number of substeps. The ideal fraction of failed steps is a function of model evaluation cost and the ODE solver order. Experience with the systems simulated with RefSim indicated that a failure rate of approximately 10% was optimum for these simulations. Accordingly, $FracOK$ was set equal to 0.9.

One possible improvement upon the safety factor retuner would be to have it maximise the simulation rate. The simulation rate is already measured by the output system and it could be made accessible to the ODE step size controller. Simulation

rate is sensitive to many factors besides the ODE solver step failure rate, however, (such as the rate of discrete event occurrence and the lengths of time between discrete events), so it would be difficult to obtain a consistent figure of merit upon which the retuner could base its decisions.

8.4.4.2 Proportional-integral step size controller

The proportional-integral (PI) step size controller was derived from Gustafsson *et al* (1988) and it is provided for cases where the integral step size controller performs poorly. The controller is described by the equations:

$$
\begin{aligned}
e &= \log(\Delta_0/h) - \log(\Delta_1/h) \\
P &= K_p\, e \\
I_{temp} &= I + K_I\, e \\
h_{temp} &= \exp(P + I_{temp}) \\
h &= \theta_{max}\ h, \text{ if } h_{temp} > \theta_{max}\ h \\
&= h_{temp}\ , \text{ if } h_{temp} \le \theta_{max}\ h \\
I &= I_{temp} + (\log h - \log h_{temp})
\end{aligned}
\qquad (8\text{-}3)
$$

where:

- K_p is the proportional gain (= 0.13 normally, = 0.0 if the last step was rejected).
- K_I is the integration gain (= 0.067 normally, = 0.2 if the last step was rejected).
- I the accumulated error integral.
- θ_{max} the maximum allowable step size increase factor (= 2.0).
- h the actual substep size.

A step is rejected and repeated with the newly-calculated step size if inequality (8-4) is satisfied.

$$\frac{\|\Delta_1\|}{h} > \varrho\ \Delta_0 \qquad (8\text{-}4)$$

where: ϱ is the rejection margin (= 1.2).

Gustafsson *et al* note that for stiff and semi-stiff sets of ODEs the PI controller has much better stability characteristics than the traditional integral controller. Where fluid flow dynamics are modelled within a refrigeration system simulation, or where the volumes of stirred-tank models are small relative to the flows through them, the system of ODEs which describe the simulation may well be marginally stiff. In this case, it may be advantageous to use this alternative step size controller.

8.4.4.3 Step size controller usage

In practice, the re-tuning feature of the integral action controller was found to be more useful than the stiff stability of the PI controller so the former controller was used for most simulations. One consequence of this was that the alternative (PI) controller received a smaller amount of testing.

The user may choose between step size controllers by setting a command line parameter when starting RefSim. The two controllers treat the tolerance specification slightly differently, so it may be necessary to change the user-specified tolerance if controllers are changed in order to obtain the same precision in the end. RefSim runtime options are discussed in Appendix 7.

State events are discrete events which may occur due to the behaviour of a model as opposed to time events which are scheduled in advance. Time events are handled directly by the Integrate procedure (section 8.4.5), but state event handling requires action by the step size controllers. Both step size controllers include code to check the Scheduler.StateChanged flag which allows detection of state events. A detailed description of the discrete event mechanism may be found in section 8.4.6.

8.4.5 Integration strategy controller

Overall integration strategy is determined by the Integrate procedure, which performs the following functions:

1 it calls the Scheduler.DoAllBefore procedure (section 8.4.6) with the current time as its parameter to execute all pending discrete events.

2 it calls the Scheduler.NextEventTime procedure (section 8.4.6) to obtain the time of the next scheduled event.

3 it sets the error tolerance for each individual variable based upon the general error tolerance specified by the user (Appendix 7).

4 it sets the size of each major integration step by subtracting the current simulation time from the next scheduled event time.

5 it calls the ODE step size controller (section 8.4.4) chosen by the user (Appendix 7).

8.4.6 Discrete event scheduler

Discrete events within RefSim are handled by the Scheduler module. This module contains procedures for scheduling and handling both time and state events. The distinction between these event types is described in section 7.3.3.

When an event is scheduled, a pointer to the model assigned to deal with that event is added to a discrete event queue. Priorities in the event queue are set by the time for which the event is scheduled, with earlier times having higher priorities. The discrete event queue is based upon the PriorityQueue object.

Scheduler contains five procedures and one flag which are accessible to models and other parts of RefSim:

1 The AddEvent procedure takes as parameters (a) a model to handle the event and (b) the time of the event. When the event time occurs, the Event method of the specified model is executed. AddEvent may be called by any model. The event-handling model is SELF (i.e. the model scheduling the event) for all of the models currently implemented within RefSim, so each model schedules events for itself only. Models are free to schedule events for other models, however, should that be necessary.

2 The AddStateEvent procedure takes one parameter: a model to handle the event. A state event is scheduled for that model at the current time and the StateChanged flag is set to TRUE.

3 The StateChanged flag is a BOOLEAN variable. The ODE solver checks the StateChanged flag at the end of each probe. If StateChanged is TRUE, the current substep is aborted and the step size controller returns to the integration strategy control procedure, Integrate. Integrate then selects the next step size so that the step is made exactly up to the now-pending state event and StateChanged is set to FALSE.

4 The DoAllBefore procedure takes a time as parameter and executes all events in the queue which are scheduled to occur before the specified time. This is called by the Integrate procedure with the current time as its parameter.

5 The NextEventTime procedure returns the time for which the next event is scheduled. This is called by the Integrate procedure.

6 The Kill procedure disposes of all scheduled events and frees all dynamically-allocated memory used by the scheduler priority queue. The queue is unusable thereafter, so this procedure should only be used immediately prior to halting. This is called by the FINAL output model.

8.4.7 List

The basic collection data structure used in RefSim is the sorted list object "List" implemented in the module OrdList. List is implemented using the skip list of Pugh (1990). This is a recently-developed data structure which is considerably easier to implement than balanced trees while having access times comparable to balanced trees. Insert and delete times are much shorter than for balanced trees, and the skip list has smaller per node pointer storage requirements. RefSim implements a 4th-order skip list (p = ¼, in the terminology of Pugh, 1990).

List is a container object (Cox, 1986, p.9) which holds objects of type "ListEl". The inheritance mechanism described in section 7.4.2 is used to derive other objects from ListEl which may also be held in a List. These other objects may have different properties and different method implementations. An overall diagram of the skip list structure is shown in Figure 8-7.

The skip list structure is probabilistic (i.e. the time required for random access to a particular element in the list is governed by a probability distribution), so it is possible that one or more of the data structures within a simulation may become significantly unbalanced. In the unlikely event that a skip list becomes unbalanced, it will require much more computation to randomly access some elements than others. Random access times in an unbalanced skip list will degenerate towards those of simple linear lists. The most straightforward solution to this problem is for the user to re-run the simulation with the "-r" (randomize) flag set (Appendix 7) to generate a different sequence of pseudo-random deviates each time RefSim is run. Random access to skip list elements only occurs during simulation initialisation and when inserting a scheduled event in the scheduler priority queue. All other skip list accesses during simulation execution are sequential so unbalanced skip lists will affect the time required for the simulation to start but have little effect on the simulation rate.

While the major uses of these objects occur within the simulation framework, they are available for use by any model.

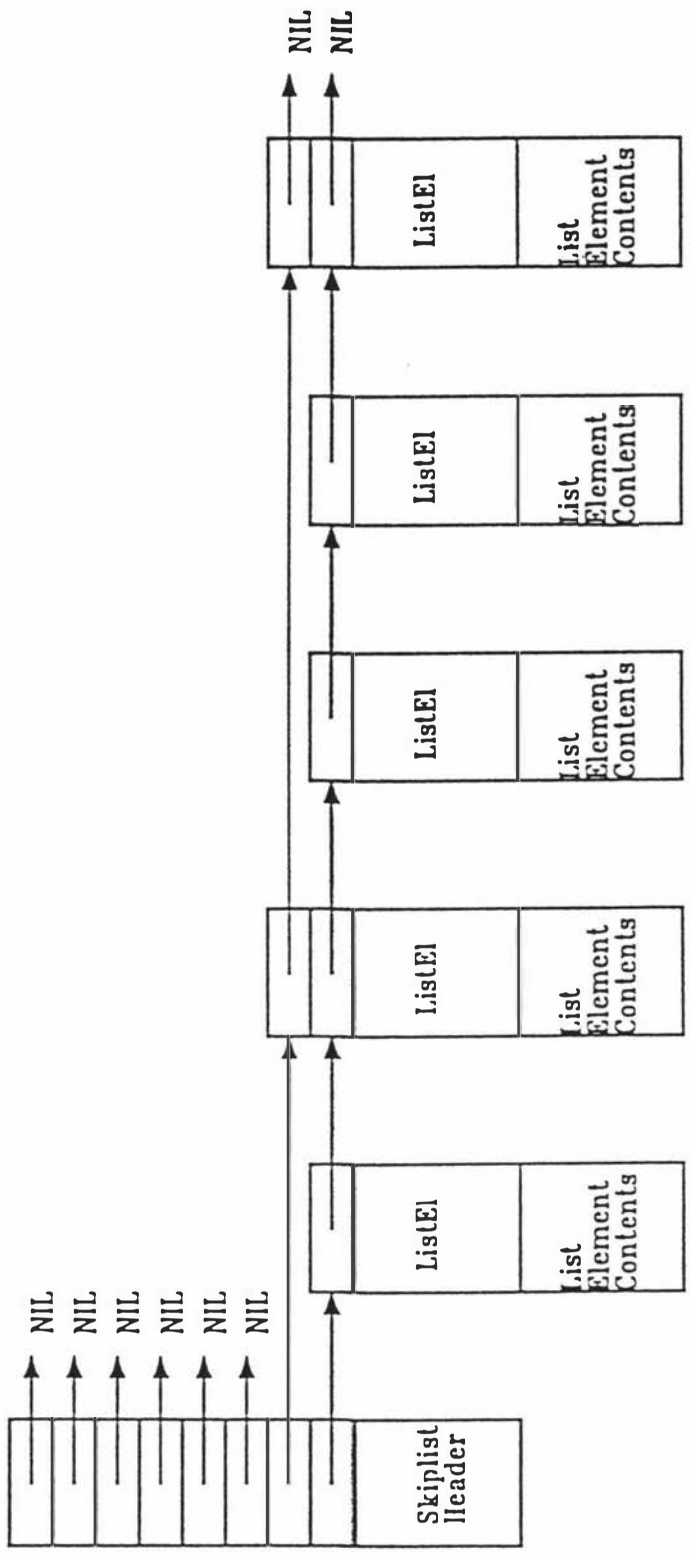


Figure 8-7 Structure of the RefSim skip list implementation

8.4.8 Pseudo-random number generators

The module Random provides three procedures which provide random deviates as LONGREAL numbers.

1 *Uniform* returns a uniformly distributed deviate between 0.0 and 1.0.

2 *Normal* returns a normally-distributed deviate with a mean of 0.0 and a standard deviation of 1.0. Normal random deviates are generated from uniform deviates using the Box-Muller transformation described by Press *et al* (1986, p.202-203).

3 *Exponential* returns an exponentially-distributed deviate with a mean of 1.0. Exponential deviates are generated from uniform deviates using a logarithmic transformation described by Press *et al* (1986, p.201).

These procedures are currently used only by the Door model, but are available for use by any model.

8.4.9 Input Parser

This utility is in three parts: Parser, MParser, and LexAnal.

8.4.9.1 Parser

Parser reads the whole RefSim input file. For each model instance, Parser calls the CreateModel procedure which inserts an instance of the specified model into the ModelList and then calls the Initialise method of that model instance. It is not useful to call Parser from any model.

A detailed description of the RefSim input file format is provided in Appendix 2.

8.4.9.2 MParser

The Initialise method is responsible for setting up the model parameters. Normally, this will necessitate some data being read from the simulation input file, and MParser is provided for this purpose. MParser accepts a parameter list which contains several associative array object instances.

The associative array object is defined in the *Assoc* module and it is used because it provides a convenient way for a model to obtain arbitrary numbers of parameters from MParser. An associative array differs from a conventional array in that it is indexed by character strings rather than by integers. This allows the Initialise method of each model to use the names of its parameters as indices in the associative arrays. The associative array concept was derived from Aho *et al* (1988, p.50) who implemented it as part of the AWK programming language. The associative arrays used in RefSim differ from those in AWK in that they are type-checked: different associative array types exist for storing strings, real numbers and tables. The implementation method also differs from that of AWK as the RefSim associative array object is a sub-class of the List object while the AWK implementation uses a hashed table (Aho *et al*, 1988, p.204).

The Initialise method must set default values in the associative arrays for all of the variables which it will accept as input data to the model and then passes the associative arrays to MParser. MParser then reads the model data up to the "END" keyword, changing the values in the associative arrays wherever necessary so that they correspond with the values provided in the input file. If the input data file contains a value which has not been given a default value set already, MParser generates an error message identifying the value read, the valid input variable names, and the line of the input file which MParser was reading at the time.

Once the associative array values have been set, MParser returns to the Initialise method, which can then obtain its parameter values directly from the associative arrays without the need to do any parsing itself. For some variables there are no sensible default values and the RefSim user must provide values for them.

The Initialise method should give this sort of variable recognizable values which would not normally appear (e.g. a temperature of -1000°C) so that it may output an error message if the user fails to provide values for those variables.

8.4.9.3 LexAnal

LexAnal carries out lexical analysis for MParser and Parser. It accepts the input file as a stream of characters and splits those characters into tokens (identifiers, punctuation and real numbers) which are acceptable to MParser and Parser. LexAnal also maintains a record of the current location in the input file for use in error messages. LexAnal maintains a one token plus one character look ahead on the input file.

8.4.10 Output

RefSim produces simulation output as a single file in tabular format with an overall title and titles for each column followed by columns of real numbers. Each model instance may request output in its Initialise method (usually based upon data that it has read from the RefSim input file) by calling the Register method of the OUTPUT model instance Output.Do. The requesting model instance supplies its own name, the name of the variable to be output and the units in which that variable's value is expressed (all as ARRAY OF CHAR) and the variable itself (which must be a single LONGREAL). The output object, Output.Do, then stores this data in a linear list along with a pointer to the variable which is to be output.

Output.Do writes the column headings when requested to do so by Parser (once all model data has been read in) and then schedules its first regular output event. When each output event occurs, Output.Do runs through the linear list of output variable pointers, dereferences each and writes the resulting value to the output file.

In a strict sense, this technique of accessing variables internal to models violates the principle that an object's data should be private and only accessible through calling one of the object's methods. The more direct approach taken here is considered justified because it is very convenient and is used for output purposes only (i.e. the variable values are read but they are never altered in this way). The alternative would have been to make a much greater range of variables available for access through the GetValue method — most of which would only ever have been accessed by Output.Do.

The other output object is Output.Final, which is the only instance of the FINAL model type. The FINAL model type is a child of OUTPUT with its event method modified to handle output at the finishing time, followed by a clean-up which deallocates all memory which has been dynamically allocated. This clean-up guards against the phenomenon of "memory leak", where some dynamic memory may be allocated by the program and subsequently forgotten.

8.4.11 Simulation error handling

The SimErr module contains a range of error-reporting procedures to simplify the handling of input or calculation errors prior to and during simulation runs. SimErr also contains math function and floating point exception-handling procedures. The available procedures are shown in Figure 8-8. Finishup, MathErr and FloatErr should not be called directly by any model. Fatal causes the program to halt, and it is called by all of the other error procedures except Warning. Warning prints a message and then returns so that the program may continue to execute after the warning.

8.5 Model implementation interface specifications

The model interface specification of section 7.9 was implemented in the definition of the base model type *Model*, from which all other models were

```
PROCEDURE Finishup ();
        Called on program termination

PROCEDURE Fatal (message : ARRAY OF CHAR);
        Fatal error output and HALT

PROCEDURE Warning (message : ARRAY OF CHAR);
        Warning text output

PROCEDURE Math (VAR e : MATHLIB.Exception) : INTEGER;
        Math exception handler

PROCEDURE FloatExcHandler (signal,type : INTEGER);
        Floating point exception handler

PROCEDURE Parse (mesg,token : ARRAY OF CHAR);
        Fatal error while parsing the input file

PROCEDURE ParseList (tok,mesg : ARRAY OF CHAR; VAR a : Assoc.AssocList);
        Fatal error while reading list of variables

PROCEDURE ParseOutput (a : Assoc.String);
        Fatal error while reading list of output variables

PROCEDURE Link (mesg1,tok1,mesg2,tok2 : ARRAY OF CHAR);
        Fatal error while linking models together

PROCEDURE WarnLink (tok1,tok2 : ARRAY OF CHAR);
        -tok1- is referenced in -tok2- but not vice versa -- optional warning

PROCEDURE LinkCount  (modeltype,modelname,mesg : ARRAY OF CHAR; count :
                      CARDINAL);
        Wrong number of models linked to this one. -mesg- should be something like 'exactly' or
        'no more than' or 'no less than' the number -count-

PROCEDURE NoVariable (modeltype, modelname : ARRAY OF CHAR; v : Variable.Type);
        Model requested to provide the value of a variable which is not available

PROCEDURE InappropriateModel  (modeltype, modelname : ARRAY OF CHAR; VAR m :
                               RefSys.Model);
        Model requested to provide information to an inappropriate model

PROCEDURE OutOfBounds (name,value : ARRAY OF CHAR; val : LONGREAL);
        Property calculation called with parameter out of valid bounds

PROCEDURE UnrecRefrigerant (name : ARRAY OF CHAR; ref : Refrig.Type);
        Refrigerant property calculation called with an invalid refrigerant number
```

Figure 8-8 SimErr simulation error reporting procedures

descended. This definition is shown in Figure 8-9.

The GetValue method is responsible for returning the current value of the requested variable to the requesting model. The Evaluate method is responsible for evaluating the equations which make up the model. The functions of the Event and Initialise methods are described in detail by sections 8.4.6, 8.4.9 and 8.4.10 above.

```
CLASS Model;
  name          : Name;
  LinkedObjects : NameList;

  VIRTUAL PROCEDURE GetValue (VAR m : Model; v : Variable.Type) : LONGREAL;

  VIRTUAL PROCEDURE Event ();

  VIRTUAL PROCEDURE Evaluate ();

  VIRTUAL PROCEDURE Initialise (Title : ARRAY OF CHAR);

END Model;
```

Figure 8-9 Model interface specification as implemented.

8.6 Model implementation caveats

A model will function within the RefSim system if it corresponds to the interface specifications outlined in section 8.5, but it may not function in the expected manner unless the following points are considered.

8.6.1 Changes in state level

Sudden changes of state variable levels can be troublesome due to the possibility that the ODE solver may backtrack if a timestep fails. Such changes are accommodated by RefSim if they take place between major ODE solver steps. Once a major step is successfully completed, there will be no backtracking and so a level change may take place. Models which take advantage of this feature include doors

(which may be open or closed), thermal objects (which may move from one environment to another) and evaporators (which may switch on or off). The common characteristic of these models is that the change in state may be scheduled in advance, and so the ODE solver ensures that the change takes place between major time steps.

Where a change may not be scheduled in advance, the problem is more difficult. Cases such as this occur frequently in multiple output process controllers. The RADSTyp4Control model, for instance, controls a group of compressors of which some are on full load, some on no load, and one on part load. If the capacity requirement drops then the requested loading on the partly loaded compressor drops until it reaches zero, at which point that compressor is switched off and the next compressor in turn is set to part load. The reverse is true if the required capacity increases. The transition from one compressor on part load to another cannot be predicted in advance and so it cannot be scheduled to take place between major time steps. Further, such a level change is quite likely to cause an ODE solver step failure if the current critical variable for ODE solver error control is in a model closely associated with the controller. This means that the same segment of time is simulated twice (or more) and the process controller model is not in the same initial state the second time as it was the first time.

Discrete events which occur due to the results of model evaluation are called state events. Crosbie (1984) described a strategy for dealing correctly with state events and this strategy has been implemented in RefSim as follows:

When a level change is required, the model schedules a state event by calling the Scheduler.AddStateEvent procedure. AddStateEvent signals the ODE solver that the current substep should be aborted by setting the Scheduler.StateChanged flag. The ODE solver then takes a step exactly up to the time scheduled for the level change and executes the scheduled event to change the state level. The simulation proceeds from that point, with the state level change successfully made between timesteps.

8.6.2 Models with memory

ODE solver backtracking may also cause a problem if a model has some memory of its state or states at earlier points in time which is apart from that contained in the dynamic variables solved by the ODE solver. Most models of physical components have no such memory, but it may be needed by process controller models.

A PID controller requires the current value, the integral, and the derivative of its input error. The integral may be found by solving an ODE, but the derivative is more difficult. The position form PID controller in RefSim tests the current value of the simulation system time against its value at the start of the substep to find if they are equal. If so, the current probe is occurring at the start of the substep and the controlled variable value at the start of the step is stored and the control action uses the derivative value that was calculated during the last probe. If the current probe is not the first in the substep, the derivative action is calculated from the current controlled variable value and the stored value at the start of the substep using a two-point differential formula and the derivative value is stored for use the next time the probe occurs at the start of a substep.

Other models which require derivatives may calculate them by similar methods.

A velocity form PID controller would have required a three-point formula to calculate the derivative action, so RefSim provides only a PI controller in velocity form.

For proper handling of controller derivative action, RefSim would have required a numerical integral equation solver running in parallel with the numerical differential equation solver and accepting data at the same probe times. This constituted an unwarranted degree of complexity given the limited number of models which required an integral equation solver and so it was not implemented.

8.7 Execution efficiency

A number of features of the RefSim implementation provide execution time and memory use efficiencies substantially superior to a straightforward implementation of the design.

8.7.1 Discrete event scheduler rounding

The arbitrary times at which discrete events may take place can cause unnecessary delays in solving the ODEs which comprise the continuous simulation part of the RefSim models. If several events are scheduled close together in time, the maximum ODE solver step length is reduced so that there is no less than one step between events. This may cause the step length to be much smaller than that required by ODE integration error control considerations so the simulation may proceed very slowly when many events are occurring in succession.

RefSim allows the user to avoid this cause of performance degradation by specifying a scheduler rounding factor. When an event is scheduled, the event execution time is set to the multiple of the rounding factor which is nearest to the next event time. This results in events being separated by no less time than the rounding factor.

One feature of the rounding factor implementation is that if an event is scheduled for a time which is less than half the rounding factor in the future, it will be executed immediately.

The effect of the rounding factor on the accuracy of the simulation is small unless the rounding factor is of a comparable magnitude to the mean time between events. If the mean time between events for any model is short, the combined discrete event/continuous simulation takes on more of a discrete event character and the ODEs have less importance. In this case, it may be possible to convert the dynamic variables in the model into algebraic variables, thereby converting entirely to a discrete event simulation.

8.7.2 The "Multiple" parameter

When simulating the application side of a refrigeration plant, it is often necessary to simulate large numbers of very similar items. For example, one coldstore in a meat processing plant which was used in testing RefSim contained about 109,000 lamb carcasses. Each ThermalObject model requires about 300 bytes of storage for itself as well as two dynamic variables and a link to the Room model in which it is contained, so it is impractical to simulate each carcass individually. Even if the computer memory were available, gathering the input data for each of these carcasses would be impractical.

As a solution to this problem, several models (including ThermalObject) have a parameter called "Multiple", which is set to the number of duplicate items to be simulated. Calculations are carried out for one instance of the model and flow variables (heat flow and mass flow) into or out of the model are multiplied by the "Multiple" factor. Where there are some differences between items, a combination of approaches is necessary. For example, if a store contains 10,000 meat cartons and 30,000 carcasses, then it may be appropriate to have two ThermalObject models: one of a carton with "Multiple = 10000" and one of a carcass with "Multiple = 30000". If the product items in the application have a range of characteristics (e.g. a range of masses), then it may be appropriate to use several ThermalObject models, each representing a part of the range.

Some consideration was given to defining ranges of product characteristics using a mean and standard deviation for each variable model property. An enhanced version of the ThermalObject model could be used to describe a range of real objects in this way but it was inappropriate to implement this idea at the environment level as it would violate the encapsulation principles of section 7.4.1.

The "Multiple" parameter is also available for RADSEvaporator models, though care should be taken that the evaporators are not individually controlled or defrosted otherwise merging the models in this way could implicitly change the control or defrosting strategies.

App2Ref automatically merges ThermalObject and RADSEvaporator models when it generates RefSim input files from RADS application files if it finds that all of the product or all of the evaporators in a room are identical.

Similar Environment models may also be merged, but in this case a "Multiple" parameter is not necessary due to the passive nature of the Environment model.

8.8 Limitations, hardware and software requirements

The decision to use TopSpeed Modula-2 to implement the RefSim system had a number of disadvantages. The most serious of these was the limited portability of the resulting software. TopSpeed Modula-2 was only available for IBM PC and compatible personal computers running either the MS-DOS (Microsoft, 1991) or OS/2 operating systems, and so RefSim is similarly restricted.

As compiled for this project, RefSim requires that the computer have an Intel 80386 or compatible processor and an Intel 80387 or compatible math co-processor (described by Pappas and Murray, 1988) supporting the IEEE-754 floating point math standard (ANSI/IEEE, 1985). It could be compiled for computers with a lower hardware specification, but run times for all but the most trivial simulations would become prohibitively long.

Memory requirements for the program itself are modest (approximately 170kB for the version described here) but complex simulations require large amounts of memory. This makes it impractical to run large RefSim simulations if memory is restricted, although it would be possible if RefSim were compiled to use code and data overlays or to store data in expanded memory (Microsoft, 1991, p.276). Overlays are segments of code and data which may be swapped to disk or to expanded memory when they are not required by the program. The use of overlays or expanded memory may incur significant execution time penalties. A better option for very large simulations would be to compile RefSim under the OS/2 operating system, for which the TopSpeed Modula-2 compiler is also available (JPI, 1991), or

to accept reduced calculation precision and to change all LONGREAL variables to REAL (which would almost halve the memory requirements of most models).

The first thing to do if memory requirements are excessive or run times are too long would be to check for duplicate models. Refrigerant evaporators are prime candidates for merging together several models and including a "Multiple" factor in the merged model.

As an indication of hardware required to carry out practical simulations, RefSim was developed on a personal computer equipped with an Intel 80386SX-16, Intel 80387SX-16 math co-processor, 4MB of RAM and running Version 5.0 of the MS-DOS operating system (Microsoft, 1991). The MS-DOS operating system does not make all of the computer's memory available to executable programs so a maximum of 450kB of memory was available for model instances. For large simulations such as the meat processing plant test cases described in sections 10.3 and 10.4, about 400kB of memory was required for model instances. RefSim could simulate at a rate of about four times faster than real time on this development system. The same simulations ran at a rate of about nine times faster than real time on another computer which was equipped with an Intel 80386-25 processor and 80387-25 math co-processor. Integration tolerance was set to 0.01 in both cases.

Smaller simulations executed much more quickly. The water chiller simulation described in section 10.2 executed at a rate of about 120 times faster than real time on the development system. The water chiller simulation would have run even more quickly but the execution speed was limited by the time required to write the output data to disk every 30 simulated seconds.

8.9 Conclusions

The RefSim environment as implemented was capable of simulating the large industrial refrigeration plants with which the project was principally concerned while using readily-available computer hardware and software. While it had been hoped that the simulation rates achieved by RefSim would be higher than those which were

actually attained, in practice the rates were high enough to produce results without undue delay. As a comparison, the RADS V3.1 program SIMUL simulated about three times faster than the RefSim rates quoted in section 8.8 on the same computers. It was hypothesised that the difference in simulation rates was mainly due to the use of integration error control in RefSim. Although SIMUL employed a time step reducer, it did not attempt to estimate the integration error and only reduced the time step length when temperature values changed rapidly.

Chapter 10 describes the process of testing the whole RefSim implementation against measured data and other simulation environments for three different plants.

Chapter 9: Model Implementation

This chapter commences by describing a simple simulation which demonstrates the features accessed by the component models. It then continues by describing the way in which the component models have been implemented in RefSim. Only the physical and mathematical bases of those models developed or enhanced specifically for this project are presented in this chapter.

The remainder of the models implemented within RefSim follow the implementation of RADS (Cleland, 1985*b*) as described by Cornelius (1991). These models have been re-implemented to correspond with the model design principles of Chapter 7, but have not otherwise been modified. The RADS models are not described in detail, but they do appear in the summary of all models implemented in RefSim (in Appendix 5) and also in the TopSpeed Modula-2 source code for the whole system (in Appendix 8, on diskette).

Experimental validation of the RefSim environment and its associated models is detailed in Chapter 10.

9.1 Example simulation

Figure 9-1 shows a simple unrefrigerated room. A simulation of this room would contain a total of ten model instances derived from five model types:

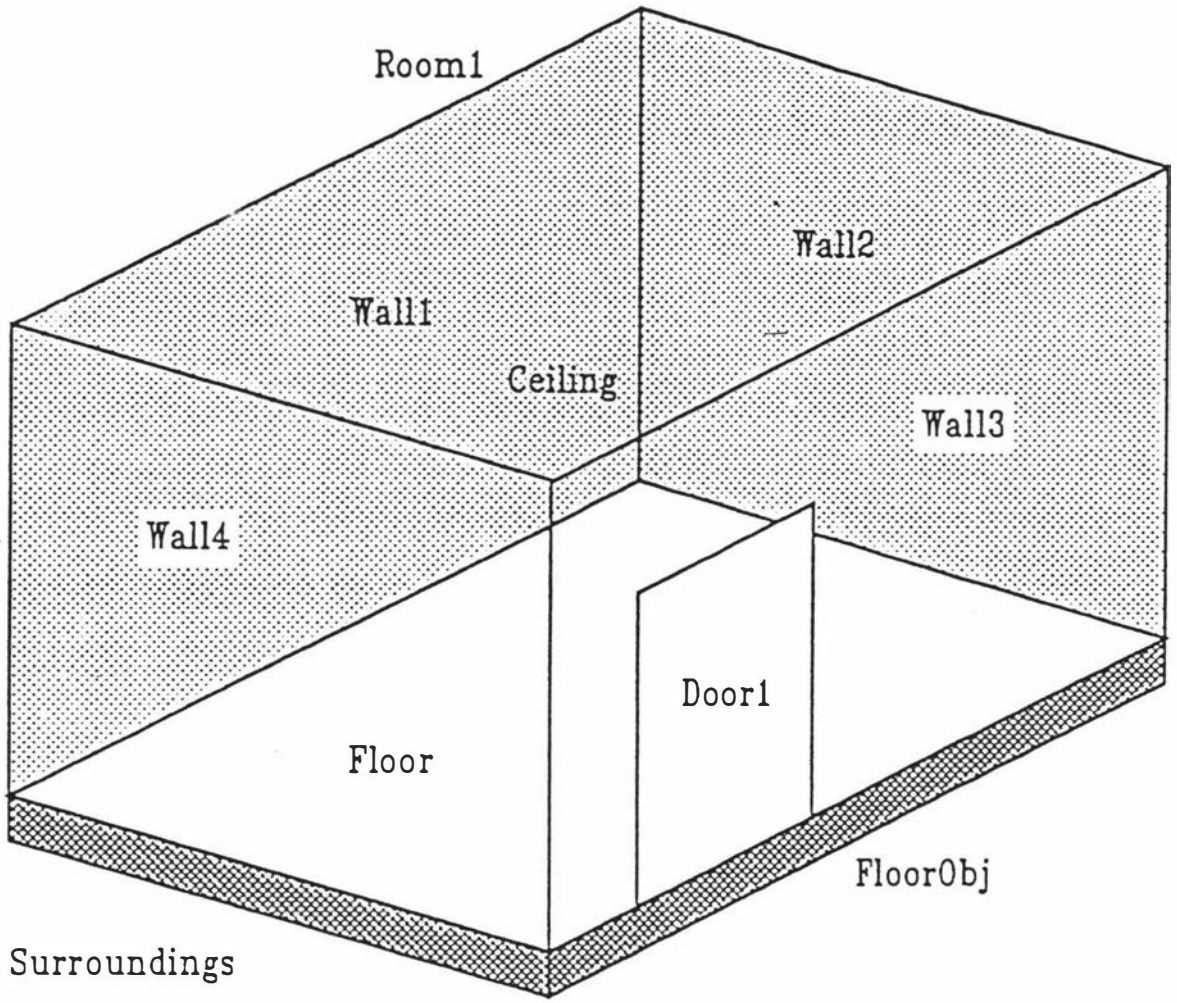
1 Six *Wall* models (Wall1, Wall2, Wall3, Wall4, Ceiling, Floor).

2 One *Door* model (Door1).

3 One *ThermalObject* model (FloorObj) representing the thermal mass of the concrete floor.

4 One *Environment* model (Surroundings) representing the ambient conditions outside the room.

5 One *Room* model (Room1) representing the humid air mass in the room.



Model instances within the room:

Name	Type
Door1	Door
Wall1	Wall
Wall2	Wall
Wall3	Wall
Wall4	Wall
Ceiling	Wall
Floor	Wall
Room1	Room
FloorObj	ThermalObject
Surroundings	Environment

Figure 9-1 Simple unrefrigerated room simulation.

The data structure which lists all of the available models within RefSim is TypeList. Figure 9-2 shows the minimum content of TypeList which is required to support this simple simulation. This simulation requires that there be models available for *Door*, *Environment*, *Room*, *ThermalObject* and *Wall*. The available models are listed according to the ASCII (American Standard Code for Information Interchange) collating sequence. A pointer from each node in TypeList points to the base instance of its model. The base instance is copied by the CreateModel procedure when a new model instance is created.

As RefSim reads the input file containing the descriptions of the model instances, it creates a ModelList as shown in Figure 9-3 which contains a pointer to each model instance. Again, the models are stored in the ASCII collating sequence. ModelList does not actually store the type of each model instance, but rather (as discussed in Chapter 7) each model "knows its own type" and responds appropriately when the RefSim EvaluateModels procedure sends it the "Evaluate" message.

In this simple simulation, there are few discrete events scheduled. Figure 9-4 shows the state of the discrete event queue immediately after initialisation but before any events have been executed. Events are scheduled for the initial and final outputs, for notionally loading the FloorObj *ThermalObject* into the room at the start of the simulation, and for the initial door opening event. Subsequent events to be scheduled later during the simulation would include other output events, door openings and door closings.

9.2 Room

As discussed in section 7.5.2, *Room* models a humid air volume and includes consideration of the condensed water mass in the room. The condensed water is assumed to be well distributed around the room, with no "puddles", i.e. it forms a thin layer of frost, drip or fog depending upon the room conditions.

The mass of humid air in the room is the product of the room volume and the air density:

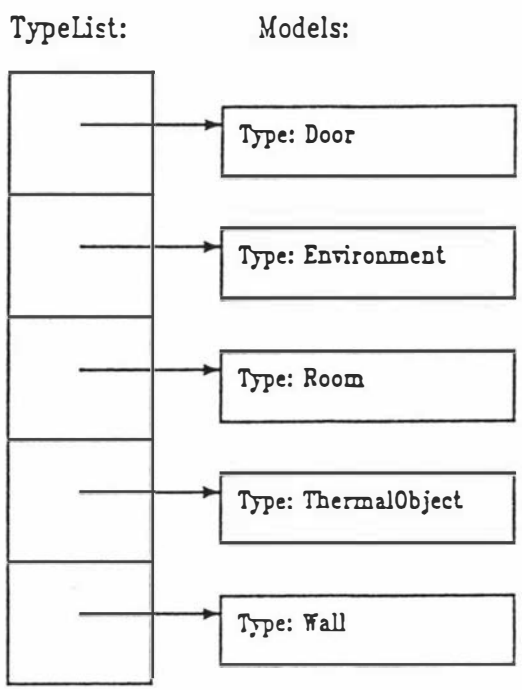


Figure 9-2 Minimum model type list for the unrefrigerated room simulation.

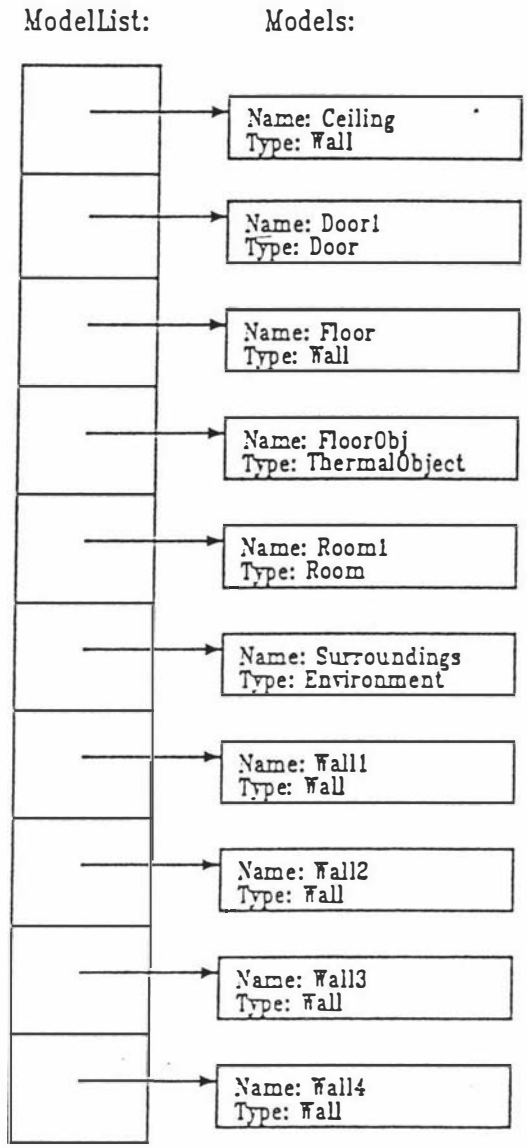
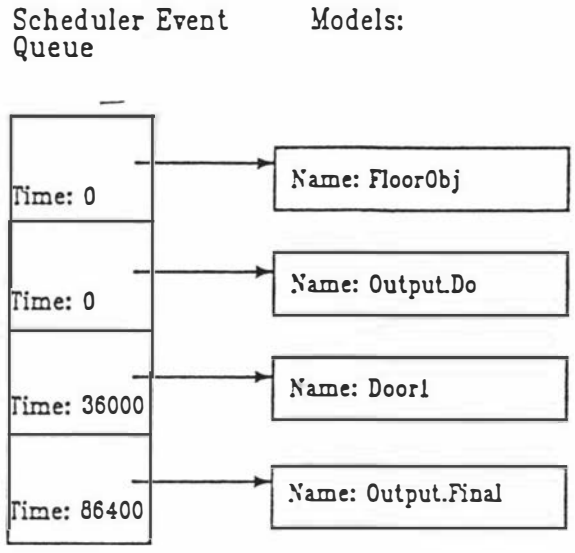


Figure 9-3 Model list for the unrefrigerated room simulation.



FloorObj is loaded into Room1 at a time of 0 seconds (it is in the room at the start of the simulation).

The first of the regular outputs is scheduled for 0 seconds so that it will occur prior to any evaluation.

The first time that Door1 is opened is 10am (36000 seconds after the start of the simulation).

This diagram assumes that the simulation is to proceed for 24 hours (so Output.Final is scheduled for 86400 seconds).

Figure 9-4 State of the Scheduler event queue at the start of the unrefrigerated room simulation.

$$M_{air} = V \rho_{air} \tag{9-1}$$

where: M is a mass / kg

V is the room volume / m^3

ρ is the humid air density / kg m^{-3}

$_{air}$ refers to the humid air in the room

The air mass may change as a consequence of density changes, and if the pressure in the room is to remain constant, this change in mass must equal the net rate of air flow into the room from other models:

$$\frac{dM_{air}}{dt} = V \frac{d\rho_{air}}{dt} = \sum_{\text{all models}} m_{model} \tag{9-2}$$

where: $_{model}$ refers to a flow into the humid air from each of the models to which the room is linked.

m is a humid air mass flow / kg s^{-1}

The change in total enthalpy content of the humid air depends upon the net energy flow into the air:

$$\frac{dH}{dt} = \phi_{Room} \tag{9-3}$$

where: H is the total enthalpy content of the humid air / J.

ϕ is an energy flow / W.

$_{Room}$ refers to a net flow into the humid air

The total enthalpy H depends upon the mass of air and its specific enthalpy $\hat{H}$, so:

$$\frac{d(M_{air}\,\hat{H})}{dt} = \phi_{Room} \tag{9-4}$$

Air density is a function of temperature and pressure, so if the air pressure is to remain constant while the temperature changes there must be some leakage between the room and its surroundings. The change in total room enthalpy is easy to evaluate if the room temperature is increasing (and therefore air mass is being lost from the room), but the enthalpy change when the room temperature is decreasing will depend upon the conditions in the area from which the air leaks into the room.

From an energy balance over the room, the net energy flow into the humid air is found to be:

$$\phi_{Room} = \phi_c + \sum_{\text{all models}} \phi_{model} \qquad (9\text{-}5)$$

where: $_c$ refers to a flow from changes to the water mass in the room (due to water condensing or evaporating).

Heat flows from other models may be due to either actual heat flows or to differences in specific enthalpies of the mass flows described by equation (9-2).

A model of the humid air mass in a refrigerated room has the potential to become very complicated if all of the interactions between the variables are considered, even if the model uses only a single zone. On the other hand, some previous workers (e.g. Cleland, 1983, 1985*a*,*b*) have simplified the room model by assuming that the mass of air in the room is invariant.

If it is assumed that the mass of air in the room and its humid heat capacity do not change with temperature (and hence with time), and that the condensed water mass is similarly invariant, this implies that:

$$\frac{dT}{dt} = \frac{\dot{\phi}_{Room}}{M_{air} C_{air} + W C_{water}} \qquad (9\text{-}6)$$

where: $\dot{\phi}$ is a sensible heat flow / W.

C is a heat capacity / J kg^{-1} K^{-1}

T is the temperature in the room / °C

W is the mass of condensed water in the room / kg

The rate of water vapour mass change is:

$$\frac{dM_{vapour}}{dt} = w_{Room} = w_c + \sum_{\text{all models}} w_{model} \qquad (9\text{-}7)$$

where: w is a water flow / kg s^{-1}

$_{vapour}$ refers to the water vapour in the room

Again assuming that the mass of air is constant, the rate of humidity change is:

$$\frac{dh}{dt} = \frac{1}{M_{air}} \frac{dM_{vapour}}{dt} \qquad (9\text{-}8)$$

where: h is the absolute humidity (M_{vapour} / M_{air})

Assuming that the condensed water is at the same temperature as the air in the room, the sensible heat released by condensing water vapour is:

$$\phi_c = \hat{H}_{fg} w_c \qquad (9\text{-}9)$$

where: $\hat{H}_{fg}$ is the difference in enthalpy between water liquid (or solid if $T < 0°C$) and vapour at the same temperature / J kg^{-1}

The rate of water vapour condensation can be modelled simply by assuming that it is a function of the difference between the current relative humidity and the saturated relative humidity (i.e. 1.0) in the room, and of the general rate of air movement in the room:

$$w_c \quad \frac{(RH - 1)\, h_{sat}\, M_{dry\ air}}{\tau} \qquad (9\text{-}10)$$

where: RH is the relative humidity

h_{sat} is the saturated mass basis humidity at the current temperature.

τ is the room time constant / s

The room time constant, is defined by $\tau = V/F$ where F is the total volumetric flow rate of air through the room. In most rooms the most important component of F would be the air flow through forced-convection evaporators, but door infiltration and ventilation may also contribute to air movement in the room. *App2Ref* calculates F by adding evaporator flow rates, and uses a volume-based heuristic to estimate τ for natural convection rooms.

The rate of water vapour condensation may become negative. If this should occur, then water will evaporate as long as $W > 0$.

The mass of condensed water in the room also depends upon the rate of any liquid water flow (w_{liq}) into the room, so:

$$\frac{dW}{dt} = w_c + w_{liq} \qquad (9\text{-}11)$$

Liquid flows would most commonly result from drains, and would therefore be negative. Again, $dW/dt = 0$ if $W \leq 0$.

When the *Room* model was implemented in RefSim, it was found that the full model described by equations (9-1) to (9-5), (9-7), (9-9), (9-10) and (9-11) was too complicated to evaluate within a large meat plant simulation which might contain 25 rooms along with other models. The interactions between the variables in the full room model were such that a set of simultaneous equations had to be solved at each model evaluation and this would make the time required for a simulation excessive on the available computing equipment. Although it was considered impractical to implement the full model, it was desirable to achieve greater accuracy than was possible by using equations (9-6) and (9-8) to replace equations (9-2) to (9-4) (i.e. by assuming that the mass of air in the room did not change with room temperature).

The approach which was applied was to use equations (9-6) and (9-8) but to allow M_{air} to vary as an algebraic function of the room temperature, T. Although the resulting equations cannot be derived mathematically, it was considered that the accuracy might be better and should be no worse than following the simpler

approach of treating the air mass as constant. By making M_{air} a function of temperature, the change in buffering effect ($M_{air}\ C_{air} + W\ C_{water}$) with changing air mass was at least partially considered. It was hoped that the model testing described in Chapter 10 would indicate whether this compromise between the mathematical and physical approaches was justified.

9.3 RADSRoom

The *RADSRoom* model is derived from *Room* by removing the upper limit that *Room* places on absolute humidity. The equations describing *RADSRoom* are similar to those for Room, with the following differences:

1 the rate of water condensation (w_c) is set to zero for the *RADSRoom* at all times.

2 there is no equivalent to equation (9-11).

This model corresponds closely with the room model described by Cornelius (1991), except that the mass of air in the room is calculated as an algebraic function of air temperature (as for *Room*, section 9.2) rather than holding it constant as was done in the RADS simulation.

9.4 Door

The door model follows the equation of Tamm (1965) as modified by Cleland (1985*a*) to include a "protection factor" which accounts for the effects of high door opening frequency and door protection techniques (e.g. plastic strips, air curtains) on air flow through the door. The model of Tamm (1965) predicts that the volumetric flow rates in both directions through the door should be equal. In the Door model, this is slightly modified so that the mass flow rates in each direction are

equal. This modification implicitly accounts for air leakage as discussed in section 9.2.

The mass flow rate through the door is an algebraic function of the air temperature on each side and it therefore responds immediately to any changes in the conditions on either side of the door.

Door openings are assumed to occur at irregular intervals and follow a Poisson distribution. The times between each opening therefore follow an exponential distribution. The lengths of the door openings are assumed to follow a Gaussian distribution.

When an opening event is triggered, the closing time is calculated from:

$$t_{Close} = t_{Current} + \text{Normal}(t_{mo},\sigma)$$

where:

t_{Close} is the next closing time / s

$t_{Current}$ is the current time / s

t_{mo} is the mean length of openings / s

σ is the standard deviation of the door opening lengths / s

"Normal" returns a normally-distributed random deviate with mean t_{mo} and standard deviation σ. When a closing event is triggered, the next opening time is calculated from:

$$t_{Open} = t_{Current} + \text{Exponential}(t_{mb}) \qquad (9\text{-}13)$$

where:

t_{Open} is the next opening time / s

t_{mb} is the mean time between openings / s

"Exponential" returns an exponentially-distributed random deviate with mean t_{mb}.

The previously-reported door model of Cleland (1985*a*) which was included in the RADS simulation program, SIMUL, had evaluated the door opening times in a slightly different manner. At each ODE solver time step between the first opening time and last closing time of a door, SIMUL calculated a uniform pseudo-random deviate and opened the door for that time step if the deviate was less than the door

opening frequency. The method used by the RefSim door model is essentially equivalent to the RADS model in its selection of door opening times, although in most cases the RefSim model would require the generation of fewer random deviates. The lengths of the door openings predicted by the two models would differ, however, because the RADS model door opening lengths were always equal to the length of an ODE solver time step.

The RADS door model smoothed out intermittant door loads using a five-element boxcar filter so that the heat load passed to the room on any time step was the mean of the loads calculated during the present and the previous four time steps. The RefSim model does not implement such a smoothing method. Should such smoothing be necessary, then it may be achieved by the addition of another two models between the door and the main room model: a small *Room* model to represent the zone of air next to the door and an interface model (perhaps called "*Mixer*") between the door zone and the main room. The *Mixer* model would permit a limited rate of air interchange between the door zone and the room, thereby smoothing the intermittant door load using a mechanism similar to that which exists in practice. The concept is illustrated in Figure 9-5. It was hoped that Chapter 10 would show how important such a scheme would be to the accurate modelling of room conditions.

9.5 Ventilation

This model describes a fixed flow of air between one *Room* and another (or between a *Room* and an *Environment*). In essence, a *Ventilation* model is similar to a continuously open door. As was done for the *Door* model, the basic equations are modified so that there is an equal mass flow in each direction. While the ventilation is switched on, the enthalpy and water vapour flows between its adjoining models are

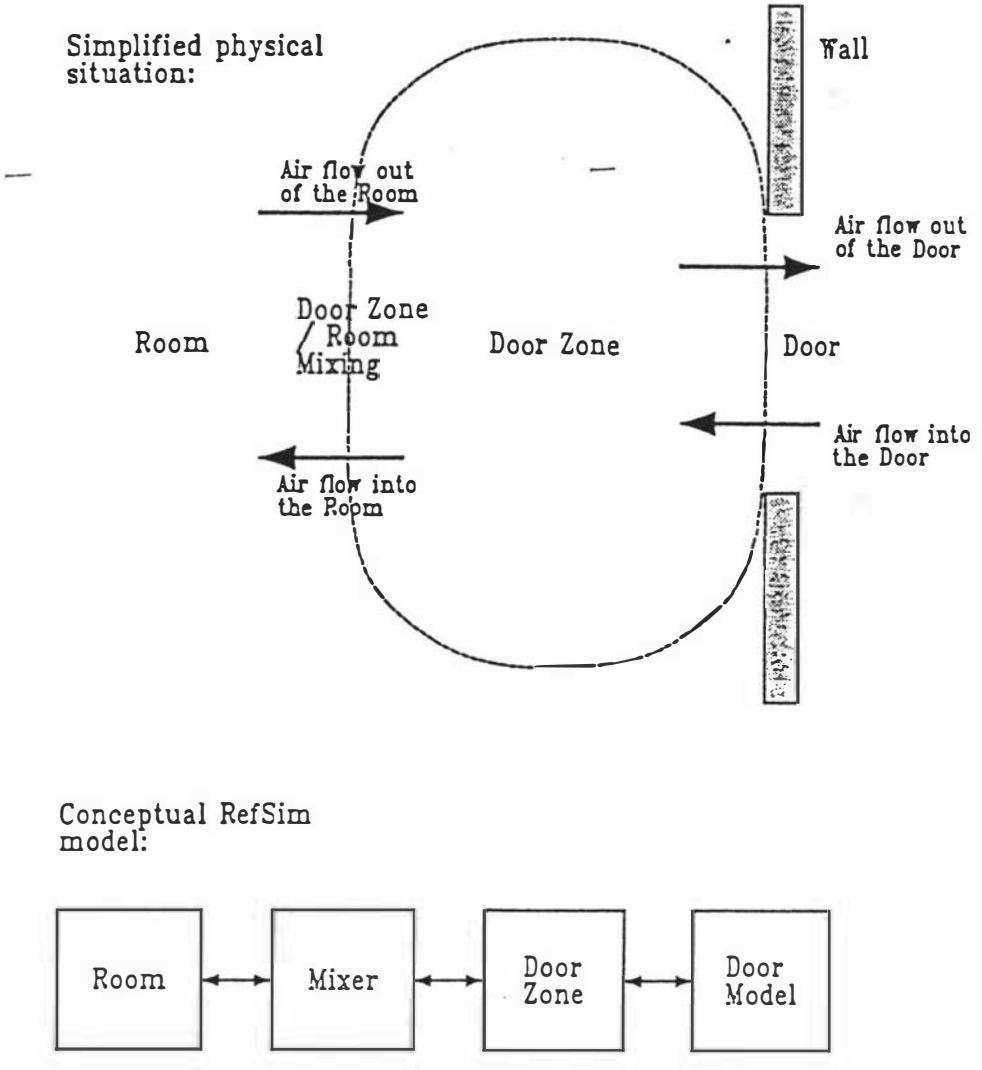


Figure 9-5 Conceptual diagram of a buffered door as it may be modelled in RefSim.

determined by:

$$\phi_{Vent} = Q_{air}\ \rho\ (\hat{H}_1 - \hat{H}_2) \qquad (9\text{-}14)$$

and:

$$w_{Vent} = Q_{air}\ \rho\ (h_1 - h_2) \qquad (9\text{-}15)$$

where ρ is the mean of the air densities in models 1 and 2. Ventilation is modelled in this manner so that it is only necessary to have one ventilation model per room rather than the two that would be required if the ventilation model passed flows in only one direction.

9.6 Wall

The rate of heat transfer through a *Wall* model is determined by the equation:

$$\phi_{Wall} = U\ A\ (T_1 - T_2) \qquad (9\text{-}16)$$

where: U is the overall heat transfer coefficient for the wall / W m^{-2} K^{-1}

$_{1\ 2}$ refer to either side of the wall

The intrinsic property U describes the convection heat transfer at both sides of the wall as well as the conductivity of the wall material. As with *ThermalObject*, this implies that the surface heat transfer coefficients on either side of the wall do not change during the simulation.

The *Wall* contains no heat capacity. If a wall has a significant heat capacity, then two of the ways in which it might be modelled are shown in Figure 9-6. Option (1) shows a wall with combined thermal capacity and conductivity properties. This is difficult to model without using a one-dimensional finite difference scheme — a method which was considered to require too much computation for the present work. Option (2) shows the same wall with the thermal capacity property separated from the conductivity property. This second option was used in the RADS simulation environment of Cleland (1985*b*) and the RADS approach was followed in the

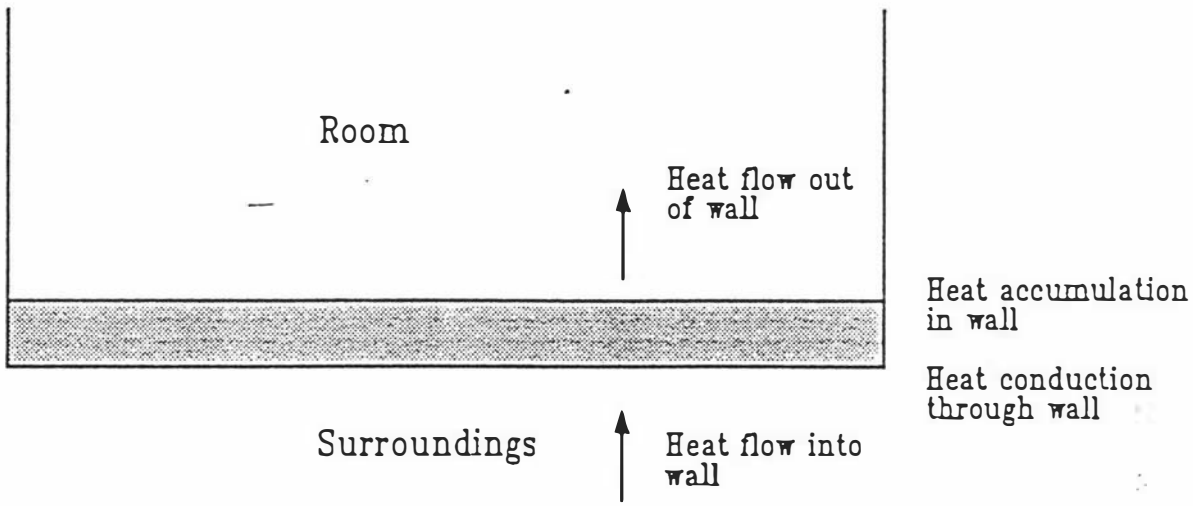
simulations carried out using RefSim. Where RADS used a "structures" model and a wall model, the simulations described in Chapter 10 represented such a wall using one *Wall* and one *ThermalObject*. For example, in section 9.1, the simple room simulation includes two models for the floor: Floor (a *Wall* model to represent the heat conduction property of the floor) and FloorObj (a *ThermalObject* model to represent the heat capacity of the floor).

The separated property approach was expected to be close to reality when the floor was well insulated. In the well insulated case, the FloorObj model would represent the concrete slab above the insulation layer and the Floor model would represent the insulation layer. If the floor was not insulated then a more complex model might be justified as there may be a more significant temperature gradient across the concrete slab.

9.7 ThermalObject

A thermal object is a solid component of a refrigeration application which may change in temperature during the simulation. The *ThermalObject* model follows the product heat load model described in Chapter 4 as extended by section 4.10 to cover structural materials as well as food product. Although the heat load provided by a thermal object is its most important characteristic, food product items have other characteristics which should be considered in a dynamic simulation. These secondary characteristics include weight loss (due to drying) and respiration heat load (due to biological processes in chilled product).

(1) Wall with combined thermal capacity and conductivity



(2) Wall with separated thermal capacity and conductivity

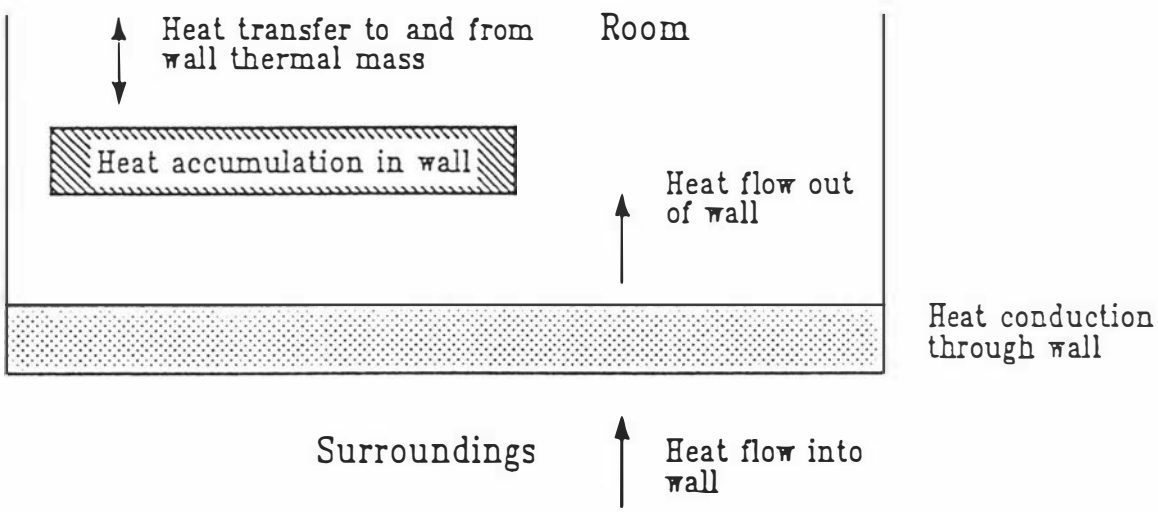


Figure 9-6 Two alternative conceptual diagrams for a Wall model.

The *ThermalObject* model includes consideration of product weight loss based on the model of Pham and Willix (1984). The rate of water vapour loss is given by:

$$w_{Prod} = \frac{k_Y A (h_{sat} - h)}{1.0 + \mathrm{Exp}(0.085 T_a) + k_Y R_v} \qquad (9\text{-}17)$$

where:

A is a surface area / m^2

$_{Prod}$ refers to the food product item

$_a$ refers to the ambient air conditions around the product

h_{sat} is the saturation mass basis absolute humidity at T_a

k_Y is a mass transfer coefficient which expresses the rate of mass transfer through the product/air boundary layer resistance / kg m^{-2} s^{-1}

R_v is a vapour diffusion resistance which expresses the rate of water vapour diffusion through a desiccated layer at the product surface / m^2 s kg^{-1}

The model estimates a water vapour loss rate (w_{Prod} / kg s^{-1}) for inclusion in the room water vapour mass change calculation of equation (9-7). Any heat required to evaporate w_{Prod} is deducted from the calculated sensible heat load of the *ThermalObject*.

ThermalObject does not maintain a dynamic variable for the mass of the product, nor does it change the vapour diffusion resistance during a simulation. Both of these factors would lower the accuracy of the weight loss model if product encountered substantially varying conditions during the simulation (e.g. freezing followed by storage for a long period).

Pham and Willix (1984) developed this model for the desiccation of food product in long term cold storage by assuming that the product was in hygrothermal equilibrium with the air. In freezing or chilling, the model departs from the physical reality due to the absence of the assumed equilibrium. Despite this problem, equation (9-17) should be more accurate than the alternative of assuming a constant

rate of weight loss during the process (as used by the RADS simulation software, Cornelius, 1991) because w_{Prod} has some dependence upon the air temperature and humidity.

If measured parameter values are not available, k_Y can be estimated approximately by using the Lewis relationship (Treybal, 1984, p.240) and R_v can then be calculated from equation (9-17) given a set of conditions and the average weight loss rate under those conditions. The utility program *App2Ref* uses this method to estimate these two parameters from the percentage weight loss value supplied by the RADS program APPLICS.

ThermalObject also includes a simple linear model for food product respiration heat load which follows that included in RADS (Cornelius, 1991):

$$\phi_{resp} = A + BT_{ma} \qquad (9\text{-}18)$$

This calculation is used where product is chilled or stored above 0°C. Chemical reaction theory indicates that the respiration heat load should show an exponential relationship with temperature over the temperature range of interest in a refrigeration system simulation, but the linear relationship of equation (9-18) was preferred because its predicted heat load was less sensitive to errors in the input data.

As discussed in section 7.5.4, *ThermalObject* treats surface heat transfer coefficient as one of its intrinsic properties, and so this does not change during a simulation even if the *ThermalObject* is moved from one room to another or if the air flow velocity in the room changes. The use of an interface model between the *ThermalObject* and the room model would resolve this problem should it be found to have a detrimental effect on the simulation accuracy.

9.8 RADSPipe

The pipeline model included in RADS is implicit within the RADS simulation environment. *RADSPipe* attempts to express this pipeline model explicitly.

RADSPipe is a thermal model. It has connections to (up to) three other models. Two models form the refrigerant source and destination respectively, while one forms the environment through which the pipeline passes.

RADSPipe accepts an energy flow from either or both ends. If one of the models to which it is connected is passive (such as a refrigerant vessel) and cannot provide a energy flow, it returns 0.0. The energy flow through the pipe is determined by the minimum magnitude of the energy flows at the ends, so that if the source supplies a larger energy flow than the destination can accept, then the energy flow through the pipeline is set by the destination.

The energy flow to the destination is the sum of the energy flow from the source and of any heat flowing into the pipe from the surrounding environment, which the *RADSPipe* calculates in the same way as the *Wall* model. Pressure is passed from the destination model to the source model after adding any pressure change which may occur in the pipeline. The *RADSPipe* model assumes that the pressure at the source end of a pipeline is higher than that at the destination by a fixed quantity which does not change with flow rate.

Some consideration was given to using more sophisticated pipeline models. In declining order of complexity, the alternatives considered were:

1 splitting the pipeline into several zones, each with ODEs derived from mass and energy balances
2 using a single zone with a lag function dependent upon flow rate
3 making pressure drop a simple algebraic function of mass flow rate

The full range of model complexities would be compatible with the RefSim environment. The first two alternatives had the advantage that it would not be necessary to specify their source and destination ends — they would be capable of determining the direction of flow from the end conditions. The third alternative did not have that advantage, but it had the potential to produce a superior pressure drop

estimate compared with the fixed pressure drop approach. In the end, the simplest approach (fixed pressure drop) was implemented for two reasons:

1 in the New Zealand meat industry plants of interest to the project, most refrigeration systems are pump-circulated. Wet suction lines would therefore make up about half of the pipelines in a simulation. Two-phase flow modelling is an active research area and there is little consensus on the best approach to follow. It can also be difficult to estimate the recirculation rate on a real plant.

2 the other models implemented within RefSim were of the "thermal" type and few were capable of making accurate estimates of the refrigerant mass flow passing through them. In the absence of accurate mass flow rate estimates, it was unlikely that any pipeline model could improve significantly upon a fixed pressure drop estimate.

If an hydrodynamic evaporator model (such as that developed by James, 1988, for instance) was implemented within the RefSim environment then it may be worth considering an enhanced pipeline model. Until then the benefits of a more detailed pipeline model were considered to be outweighed by the development and data acquisition effort required.

9.9 Header

The *Header* model solves the problem of connecting a common pipeline model to several refrigeration systems components. A pipeline model has only two ends so it cannot be connected to more than one component at each end. The *Header*, on the other hand, is connected to one pipeline which forms either its refrigerant source or destination. If the pipeline is the source, then energy and mass flows coming from the pipeline are divided equally into flows to the destination models. If the pipeline is the destination, then heat and mass flows from the source

models are added to calculate the energy and mass flows supplied to the pipeline. Pressures are passed to models in the reverse direction to flows after adding any fixed pressure change which may occur in the header.

Like the other refrigeration system component models implemented in RefSim, *Header* is a "thermal" model and does not consider hydrodynamics. The absence of pressure drop and flow calculations means that a feed header model (one with a pipeline as its refrigerant source) cannot easily determine how to split its outlet flows. The assumption that the outlet flows are all equal will be satisfactory if the components at the header outlet have similar capacities and they all operate at the same time. In particular, while it would be appropriate to use a Header to supply several evaporators, it would not usually be appropriate for a Header to serve as the suction model for a number of compressors. In the RefSim simulations of large meat processing plants which are discussed in Chapter 10, a *Header* is used as the refrigerant destination model of the condenser models, but not as the feed model for this reason.

9.10 RADSVessel

The RADS model for a refrigerant vessel is implemented in RefSim following the description of Cornelius (1991).

An energy balance over a refrigerant vessel may be approximated by:

$$\frac{dH}{dt} = \phi_{Vessel} \qquad (9\text{-}19)$$

where: ϕ_{Vessel} is the net energy flow into the vessel / W

Now, at any given time:

$$H_{Vessel} = \hat{H}_{vap} M_{vap} + \hat{H}_{liq} M_{liq} \qquad (9\text{-}20)$$

where: $_{vap}$ refers to refrigerant vapour in the vessel

$_{liq}$ refers to refrigerant liquid in the vessel

and:

$$V_{Vessel} = V_{vap} + V_{liq} \tag{9-21}$$

where V_{Vessel} is the fixed volume of the vessel. Now:

$$M_{vap} = V_{vap}\, \rho_{vap}(T_{Vessel}, P_{Vessel}) \tag{9-22}$$

and:

$$M_{liq} = V_{liq}\, \rho_{liq}\, (T_{Vessel}) \tag{9-23}$$

where: T_{Vessel} is the temperature in the vessel assuming that the vessel is well mixed / °C

P_{Vessel} is the pressure in the vessel / Pa

If it is assumed that both the liquid and vapour are saturated, then equation (9-22) simplifies to:

$$M_{vap} = V_{vap}\, \rho_{vap}(T_{Vessel}) \tag{9-24}$$

where T_{Vessel} is a function of $\hat{H}_{liq}$.

Cleland (1983,1985*a*) assumed that M_{liq} was constant and that $C_{liq}\, M_{liq}$ was much greater than $C_{vap}\, M_{vap}$. If the temperature in the vessel did not vary greatly, then ρ_{liq} and hence V_{liq} would also remain constant. Under these conditions, the temperature in the vessel is determined by the net energy flow into the vessel and the heat capacity of the liquid refrigerant mass:

$$\frac{dT}{dt} = \frac{1}{M_{liq}\, C_{liq}} \left(\sum_{\text{all models}} \phi_{model} \right) \tag{9-25}$$

Cleland (1985*a*) found that this model was satisfactory for the conditions encountered in a New Zealand meat industry refrigeration plant, so this model was adopted for implementation in RefSim. The *RADSVessel* model in RefSim differs from the RADS implementation in that it considers each refrigerant flow into and out of the vessel separately rather than treating them in combination as did RADS. By

separating treatment of the flows, the RefSim *RADSVessel* model was made more flexible than the RADS implementation. For example, it could be used as a condensed liquid receiver as described in section 9.12 in place of the dedicated model used for the purpose in RADS (Cornelius, 1991).

As was the case in RADS, each vessel has a designated liquid refrigerant source and it calculates the flow of liquid required from that source to maintain a refrigerant mass balance. Pressure is calculated as an algebraic function of refrigerant temperature using the thermodynamic property procedures in the Refrig module.

9.11 RADSCondenser

The *RADSCondenser* model follows that of Cleland (1985*a*). While the condenser model of Cleland (1985*a*) allowed liquid refrigerant to remain in the condenser, the present model requires a separate liquid receiver. A condensing arrangement similar to that of Cleland (1985*a*) may be achieved by connecting a group of *RADSCondenser* models in a loop so that they take refrigerant vapour from a vessel (named, perhaps, Vessel0) and return to that vessel refrigerant liquid at the same temperature. Compressors would feed vapour to Vessel0 and Vessel0 would be the liquid refrigerant source for other vessels or evaporators (via pipeline models). This system is shown in Figure 9-7.

9.12 Other models

All of the other refrigeration system models implemented in RefSim directly follow those of Cleland (1985*a*) as described by Cornelius (1991).

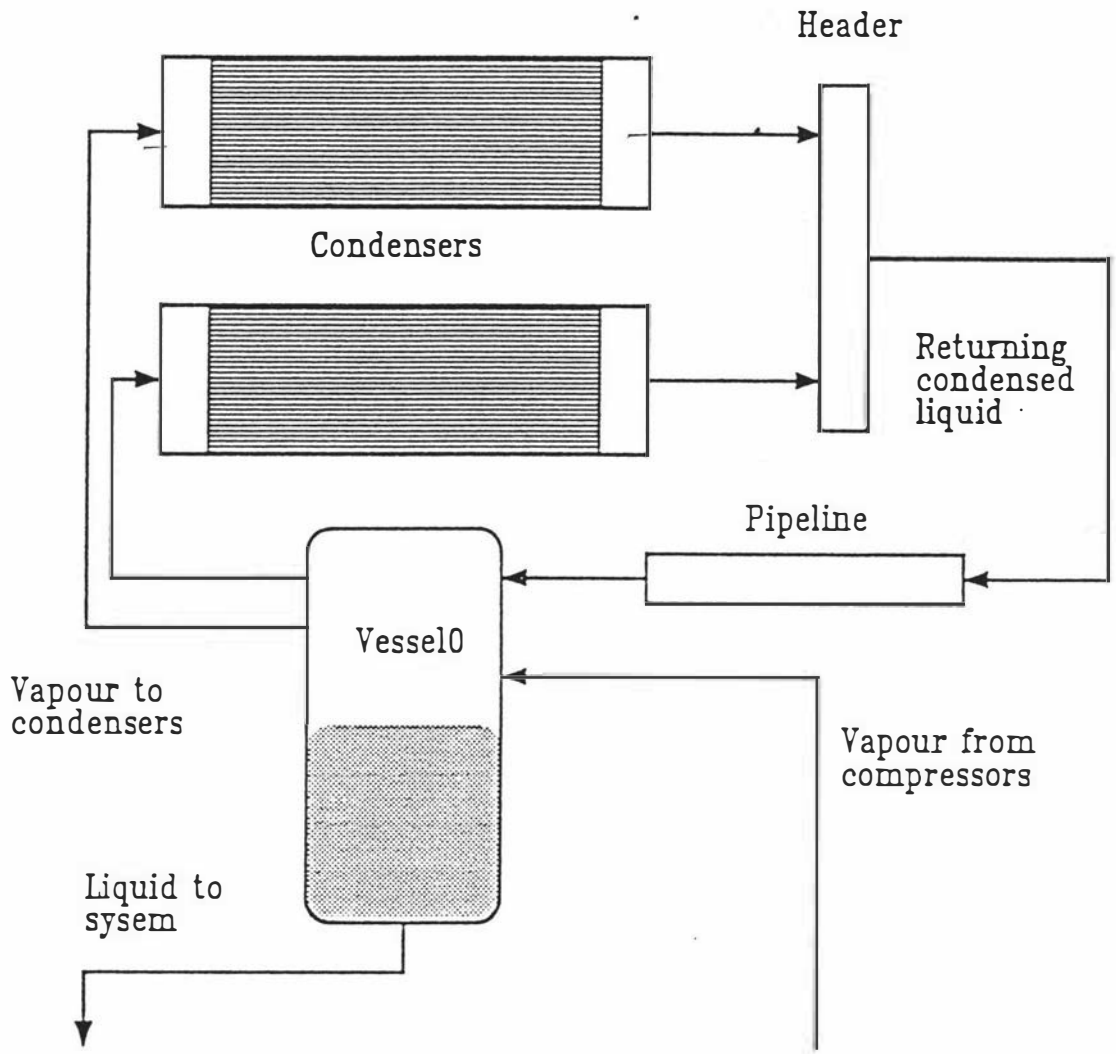


Figure 9-7 RADS condenser arrangement as it may be implemented in RefSim.

Chapter 10: Simulation Testing

This chapter describes three tests of the full RefSim simulation environment and its associated models. The testing approach followed that of Cleland (1985*a*) and consisted of monitoring temperatures around each plant as it operated for comparison with the temperatures predicted by the simulation program. The three plants from which the test data were obtained were:

- A water chiller plant. A pilot-scale water chilling system capable of working in both dynamic and steady-state modes.
- Alliance Ocean Beach. A large sheepmeat processing plant operated by the Alliance Freezing Company (Southland) Ltd.
- AFFCO Horotiu. A large beef and lamb processing plant operated by the Auckland Farmers' Freezing Co-operative.

10.1 Testing objectives

The prime purpose of this testing was to verify the correct operation of the RefSim simulation environment. Other objectives included:

- Identify model implementation errors. It was recognised in chapters 5 and 7 that measurements on a complex plant and their comparison with the results of a full plant simulation would not normally provide the detailed data required to completely validate individual models, but it was possible to identify major problems.
- Indicate whether or not the levels of model complexity used were appropriate to industrial refrigeration system simulation.

Other questions which were to be answered were as follows:

1 Can RefSim simulate meat industry refrigeration plants to at least the level of accuracy attained by existing modelling software?

2 How easy is it to describe a plant using RefSim when compared with existing modelling software?

3 How many distinct model types are needed to describe a large plant?

4 How flexible is the RefSim environment when adding new models or enhancing existing models?

5 Do the models implemented within RefSim have an appropriate level of complexity? Were simplifying assumptions and decisions justified? Was complexity added unnecessarily?

RefSim was compared with three other simulation environments:

1 A typical continuous system simulation language, ISIM (Hay and Crosbie, 1984).

2 A large refrigeration system simulation environment which was developed in the mid-1980s. RADS Version 2.2 was an enhanced version of the system described by Cleland (1985*b*) and it represented the state of the art in commercially-available refrigeration simulation software at the commencement of this project.

3 An enhanced version of the RADS simulation environment which was developed in parallel with this project. RADS Version 3.1 includes an early form of the food product heat load model described in Chapter 4. It also improved upon Version 2.2 by integrating all of the simulation ODEs in parallel (unlike the earlier version, which integrated application and refrigeration system ODEs separately). RADS Version 3.1 has been described by Cornelius (1991).

This chapter looks at how well each environment simulated the test plants when compared with the measured data. Chapter 11 compares other aspects of the simulation environments used in the testing process.

10.2 Water chiller

The food product heat load prediction method which was included in the ThermalObject model had been tested separately by the methods described in Chapter 6, so the first test case was selected to test other model types — especially those related to the refrigeration circuit. Darrow *et al* (1991) (reproduced in Appendix 4.3) had simulated a direct expansion R22 laboratory water chiller plant as it cooled a well-stirred tank of water and this plant seemed to be appropriate for the purpose.

10.2.1 Plant description

A diagram of the R22 water chill plant is shown in Figure 10-1. An alternate evaporator (APV Baker model M8-1000 shell and tube heat exchanger) and its Sporlan model FVE-5-CP100 thermostatic expansion valve also formed part of the plant and were used for steady state runs, but they are not shown on the diagram. All pipework was constructed of copper, and was sized for minimal pressure drop. The equivalent length of suction pipework was 7.26m including fittings.

The plant was capable of operating in three modes:

1 Steady state mode A. The plant was used to cool a stream of water flowing through the alternate evaporator. The plant was run in this mode to measure the compressor volumetric and isentropic efficiencies, and the condenser overall heat transfer coefficient. The water flowing through the shell and tube evaporator provided a constant load.

2 Steady state mode B. The plant was run with a steady stream of water flowing through the tank to maintain a constant temperature difference while the tank evaporator coil heat transfer coefficient was measured.

3 Dynamic mode. In this mode, the tank was filled with water at about 20°C, then the stirrer was started and the refrigeration system was started from a pumped-down state. The refrigerant evaporation, condensation and water tank temperatures were measured until the water temperature reached approximately 1°C.

The heat transfer coefficients and efficiencies measured in the steady state runs were correlated against temperature difference and pressure ratio respectively by using regression analysis. These regression equations provided the input data required by the models.

The initial steady-state measurements taken by Darrow (1990) were extended by Darrow *et al* (1991) to ensure that the regression curve fits accurately

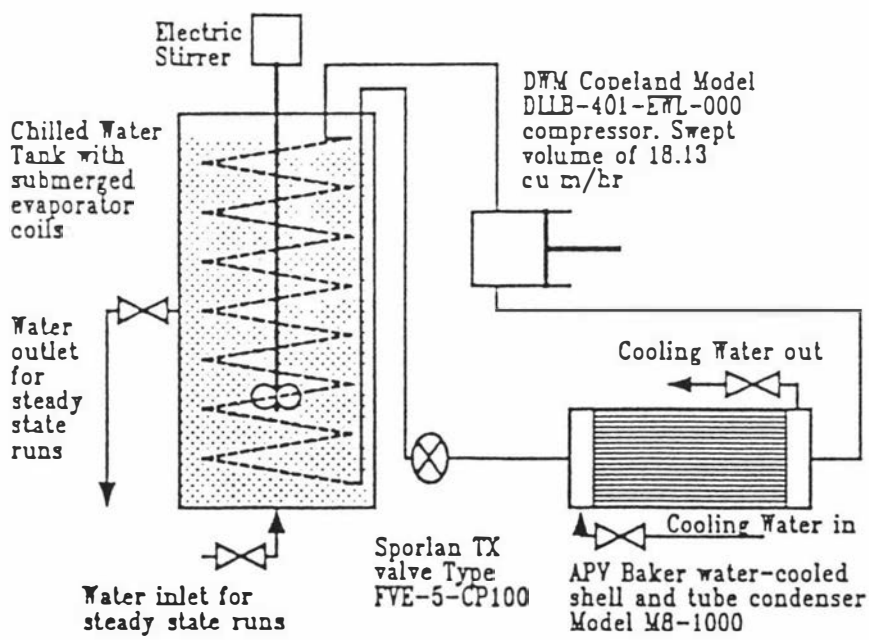


Figure 10-1 Laboratory Water Chiller Plant

characterized the water tank evaporator for the full range of temperature and pressure ratio reached during the dynamic runs. Darrow *et al* also carried out a dynamic run in which the temperatures were logged frequently (every 30 seconds) at the start of the run to identify the plant behaviour during start-up.

Table 10-1 shows a summary of the parameters for each of the three experimental runs which were simulated.

Table 10-1 Water chiller dynamic run parameters and initial conditions.

Item	Value		
	Run 1	Run 2	Run 3
Ambient temperature / °C	20.0	20.0	20.0
Condenser water temperature in / °C	16.4	16.1	16.6
Condenser water flow rate / kg s^{-1}	0.724	0.755	0.761
Condenser thermal capacity / J K^{-1}	44000	44000	44000
Evaporator thermal capacity / J K^{-1}	34000	34000	34000
Tank water mass / kg	325	339	341
Initial tank water temperature / °C	18.7	20.1	19.4
Initial condenser temperature / °C	22.2	22.2	22.2
Initial evaporator temperature / °C	-8.7	-8.7	-8.7

10.2.2 ISIM simulation

Darrow (1990) had originally used the continuous system simulation programming language ISIM (Hay and Crosbie, 1984) to implement thermal models of the water chiller components. Darrow *et al* (1991) found that simplified versions of Darrow's original models were adequate, but still retained the ISIM representation. The ISIM model for the first dynamic run is shown in Figure 10-2.

```
:Modifications by S J Lovatt, Original by J B Darrow
: Version 1.1 - Modified timestep (original was far too small)
:             - Modified TMER calculation (original used fixed value for
:               (gama-1)/gama
:             - Modified hco calculation so that all of AEER goes into vapour
:             - Changed output to include additional variables
:DYNAMIC ANALYSIS
:MODEL 1
CONSTANT tcwi=16.4          : Temp of condenser water in /C
CONSTANT ta=20              : Ambient temp /C
CONSTANT UAe=1166           : UA for evaporator / W/K
CONSTANT c=4180             : Heat capacity of water / J/kgK
CONSTANT mc=0.724           : Flow rate of condenser cooling water / kg/s
CONSTANT qs=0.0050361       : Compressor swept volume / m3/s
CONSTANT s=8.1125           : Mean superheat / C
CONSTANT mcc=44000          : Thermal mass of concenser / J/K
CONSTANT mce=34000          : Thermal mass of evaporator / J/K
CONSTANT mcewt=1358500      : Heat capacity of water in tank / J/K
t=0                         : Time from start / s
twt=18.7                    : Tank water temperature (initial) / C
tc=22.2                     : Condensation temperature (initial) / C
te=-8.7                     : Evaporation temperature (initial) / C
tfin=2000                   : Finishing time of simulation / s
cint=30                     : Calculation interval (time step) / s
algo=0                      : Algorithm is variable-step 5th-order RK
noci=2                      : No. of steps per communication interval
rerr=0.001                  : Relative integration error
:END OF SETTING UP
Sim                         : Call simulation model
Dynamic
UAc=364+1824*Mc                     : Regressed function for condenser UA
tcwo=tc-(tcwi-tc)*exp(-UAc/Mc/c)    : Condenser water out temp
dtm=-((tcwi-tc)-(tcwo-tc))/log((tcwi-tc)/(tcwo-tc))   : dtm is the LMTD
Pd=exp(21.25384-(2025.4518)/(tc+248.94))              : Discharge pressure
Ps=exp(21.25384-(2025.4518)/(te+248.94))              : Suction pressure
PR=Pd/Ps                                              : Comp ratio
tsat1=-2025.4518/(log(Ps)-21.25384)-248.94            : Suction sat. temperature
tsat2=-2025.4518/(log(Pd)-21.25384)-248.94            : Discharge sat. temp
dtc=tsat2-tsat1                                       : Chg in sat. temp
: Calculate gamma from here
g1=1.137423+(-1.50914E-3*tsat1)+(-5.59643E-6*tsat1**2)
g2=(-8.74677E-6*tsat1*dtc)+(-1.49547E-7*tsat1**2)
g3=(5.97029E-8*tsat1*dtc**2)+(1.41458E-9*tsat1**2*dtc**2)
g=(-4.5258E-4*dtc)+(g1+g2+g3)
g4=(3.68417E-4*s)+(-6.26076E-6*s**2)
g5=(1.45839E-5*s*tsat1)+(-1.65730E-7*s**2*tsat1)
gama=g*(1+g4+g5)
: Gamma is now calculated from Cleland's curve-fit
nv=0.920-0.0482*PR                                    : Regr. fn for vol effy
n=0.64                                                : Mean isentropic effy
: Calculation of saturated specific volume, V from here
vv=exp(-11.82344+2290.321/(te+273.15))
vb=1.01659+5.09433E-4*te-14.8464E-6*te*te
vc=vb-2.49547E-7*te**3
V=vv*vc
: Saturated specific volume now calculated from Cleland's curve-fit
vola=1.0+5.23275E-3*s-5.59394E-6*s*s
volb=3.45555E-5*te*s-2.31649E-7*te*s*s
volc=5.80303E-7*te*te*s-3.20189E-9*te*te*s*s
vol=V*(vola+volb+volc)
: Superheated specific volume now calculated from Cleland's curve-fit
qt=qs*nv                                              : Effective swept volume
MR=qt/vol                                             : Mass flow of refrigerant
TMER=(gama/(gama-1))*Ps*qt*((PR**((gama-1)/gama))-1)  : Theoretical energy req.
AEER=TMER/n                                           : Actual energy req.
: Calculating enthalpy of superheated vapour out of the evaporator
heoi=250027+367.265*te-1.84133*te*te-11.4556E-3*te**3
heoa=1.0-2.85446E-3*s-4.0129E-7*s*s
heob=7.27759E-6*te*s-8.11617E-8*te*s*s
heoc=14.1194E-8*te*te*s-9.53294E-10*te*te*s*s
heo=heoi*(heoa+heob+heoc)
: Finished calculating enthalpy from Cleland's curve-fit
hci=((AEER)/mr)+heo                                   : All heat into vapour
hco=44518+1170.36*tc+1.68674*tc*tc+5.2703E-3*tc**3
hei=hco
: Finished calculating compressed vapour enthalpy from Cleland's curve-fit
Tc'=(MR*(hci-hco)-UAc*dtm)/mcc                        : Condensation temp /C
Te'=(MR*(hei-heo)+UAe*(twt-te))/mce                   : Evaporation temp  /C
Twt'=-UAe*(twt-te)/mcewt                              : Tank water temp   /C
output t,tc,te,twt,gama,pr,nv                         : Output results
```

Figure 10-2 ISIM program to simulate water chiller run 1.

10.2.3 RefSim simulation

The opportunity was taken to test the flexibility of the RefSim environment by developing some specific models for the water chiller plant. Models for the liquid tank (*FluidTank*), a simple evaporator with general applicability (*GenEvaporator*) and a simple water-cooled condenser (*GenCondenser*) were written specifically for this application. The *RADSCompressor* model was modified to increase its generality.

10.2.3.1 Liquid tank model: FluidTank

This model represented the liquid in a well-stirred tank. *FluidTank* was derived from the *Application* base model and its properties were similar to those of the *Room* model described in section 9.2. There was no provision for humidity calculation and water condensation because it was assumed that the fluid was either pure or a solution which was far from its saturation conditions. For example, *FluidTank* could contain *ThermalObjects*, have walls through which heat could be conducted, and (unlike *Room*) it could have pipeline models connected to it through which heat could flow. This last feature allowed the *FluidTank* to act as a secondary refrigerant reservoir. The tank was assumed to have some air space above the liquid so that the mass of liquid would remain constant over time despite any density changes.

FluidTank had three input parameters:

Mass	-	Mass of fluid in the tank / kg
HeatCapacity	-	Specific heat capacity of the fluid / J kg^{-1} K^{-1}
Temperature	-	Initial temperature of the fluid / °C

The temperature of the liquid was described by:

$$\frac{dT_{liq}}{dt} = \frac{1}{M_{liq} C_{liq}} \sum_{\text{all models}} \phi_{model} \qquad (10\text{-}1)$$

where:
- T_{liq} was the fluid temperature / °C
- t was the time / s
- M_{liq} was the mass of fluid in the tank / kg
- C_{liq} was the specific heat capacity of the fluid / J kg^{-1} K^{-1}
- ϕ_{model} was the energy flow from each interacting model / W

10.2.3.2 Evaporator model: GenEvaporator

GenEvaporator provided a simple refrigerant evaporator model which allowed the user to specify parameters which could not be readily changed in the *RADSEvaporator* model. The evaporator was assumed to be surrounded by a well-mixed fluid so that the cooled fluid could be characterized by a single temperature. This avoided the necessity for the heat capacity of the cooled fluid to be a property of the evaporator model. *GenEvaporator* also represented the evaporator's associated expansion valve. *GenEvaporator* had several input parameters:

UA	-	Heat transfer coefficient by surface area product / W K^{-1}
MassHeatCap	-	Thermal mass of the evaporator / J K^{-1}
Temperature	-	Initial evaporation temperature / °C
Refrigerant	-	Refrigerant number
SuperHeat	-	Fixed refrigerant superheat on exit from the evaporator / °C
Source	-	Model which provides refrigerant liquid
Dest	-	Model which receives refrigerant vapour
Application	-	Model to be cooled
Multiple	-	Number of identical units represented by this evaporator

GenEvaporator estimated the evaporator energy load by:

$$\phi_{evap} = UA\ (T_a - T_{evap}) \qquad (10\text{-}2)$$

where:

T_a was the ambient temperature / °C

T_{evap} was the evaporation temperature / °C

ϕ_{evap} was the energy flow into the evaporator from the application / W

UA was the product (overall heat transfer coefficient) (heat tranfer area) / W K^{-1}

Refrigerant vapour superheat was assumed to be useful (i.e. the vapour absorbed its superheat from the cooled fluid rather than from some other part of the environment). *GenEvaporator* was a "thermal" model as hydrodynamics were not considered. It was assumed that the expansion valve worked as an equilibrium device so that at any instant the mass flow through the evaporator exactly matched the rate of vaporisation in the evaporator. The pressure drop in the evaporator was assumed to be negligible.

The refrigerant enthalpy flow rate out of the evaporator (ϕ_{out}) was determined by the component connected to the evaporator outlet if that model was active (e.g. a compressor or a pipeline connected to a compressor). If the outlet component was passive (e.g. a refrigerant vessel), the refrigerant enthalpy flow rate was determined by the evaporation rate. The evaporation temperature was then estimated by:

$$\frac{dT_{evap}}{dt} = \frac{\phi_{evap} - \phi_{out} + \phi_{in}}{MassHeatCap} \qquad (10\text{-}3)$$

where:

ϕ_{in} was the refrigerant enthalpy flow into the evaporator / W

ϕ_{out} was the refrigerant enthalpy flow out of the evaporator / W

10.2.3.3 Condenser model: GenCondenser

GenCondenser was a thermal model and analogous to *GenEvaporator*, except that it did not assume that the cooling fluid was well mixed. It also accepted an additional parameter:

CoolantMassFlow - The mass flow rate of cooling fluid / kg s^{-1}

GenCondenser assumed that the cooling fluid was liquid water. From this data, *GenCondenser* calculated the logarithmic mean temperature difference (LMTD) and then found the energy release rate from the equation:

$$\phi_{cond} = UA\ LMTD \qquad (10\text{-}4)$$

where: ϕ_{cond} was the energy flow out of the condenser into the cooling fluid / W

In place of the SuperHeat parameter of *GenEvaporator*, the condenser model accepted a SubCooling parameter which indicated a fixed amount of useful subcooling. The refrigerant energy flow into the condenser model was supplied by the compressor model connected to its inlet and it was assumed that the mass flow rate out of the condenser would equal the mass flow rate in.

10.2.3.4 Compressor model enhancement: RADSCompressor

The compressor model which was used in this simulation (and the others described in this chapter) was an enhancement of the RADS compressor model. The compressor model used by the RADS simulation program, SIMUL (Cornelius, 1991), assumed that its suction vapour was saturated. While this was found to be a satisfactory assumption for pump-circulated plants (Cleland, 1985*a*), it was not considered satisfactory for a direct expansion plant. Thus, the *RADSCompressor*

model as implemented in RefSim was modified to accept superheated suction vapour by having it request both refrigerant temperature and pressure from the model connected to its suction (where the original implementation requested only one and calculated the other), so this modification was made. This made the RefSim *RADSCompressor* model equivalent to the more sophisticated compressor model used in the RADS steady-state analysis programs.

10.2.3.5 Simulation input file

The RefSim input file for run 1 of the water chiller simulation is shown in Figure 10-3.

10.2.4 RADS V2.2 simulation

The RADS V2.2 simulation environment (Cleland, 1985*b*) had been designed to simulate large industrial refrigeration systems but many of its models were capable of application to a wider variety of refrigeration systems. An attempt was therefore made to simulate the water chiller plant using RADS V2.2.

The condenser and compressor data measured by Darrow *et al* (1991) were not in an appropriate form for processing by the RADS data preparation programs, so the engine room data file required by the SIMUL program was assembled manually. The application data file was generated by the APPLICS program with a Type 21 (liquid tank) application representing the water tank and a Type 5 (cooling coil or jacket) evaporator representing the water chiller evaporator. These application and evaporator types were described by Cornelius (1991).

The RADS V2.2 application and engine room data files which were used as input data for SIMUL V2.2 are not shown as they differed little from the V3.1 input files shown in Figures 10-4 and 10-5.

```
SIMULATION WaterChiller

{       This is a RefSim implementation of the R22 water chiller simulated by Darrow et al (1991). }

{ Data for Run 1 }

Run_length = 0.55556                { = 2000 seconds }
Output_every 8.33333E-3 to wch1.out
                                        { output every 30 s }
Report loud

MODEL CopelandDLLB ( RADSCompressor )
   SweptVol = 0.0050361             { cu m /s }
   HeatRetention = 1.0
   Refrigerant = 22
   TABLE IsEffCorr [    0.64 ]
                                    { Isentropic efficiency = 0.64 }
   TABLE VolEffCorr [   0.92    -0.0482 ]
                                    { Volumetric efficiency = 0.92 - 0.0482*PR }
   Suction = ChillEvap
   Discharge = APVM8-1000
   < Power PR >
   [ ChillEvap APVM8-1000 ]
END CopelandDLLB

MODEL APVM8-1000 ( GenCondenser )
   CoolantMassFlow = 0.724          { kg/s }
   UA = 1685                        { W/m²K }
   MassHeatCap = 44000              { J/K }
   Temperature = 22.2
   Refrigerant = 22
   Source = CopelandDLLB
   Dest = ChillEvap
   Coolant = CoolingWater
   < Temperature >
   [ CopelandDLLB CoolingWater ChillEvap ]
END APVM8-1000

MODEL ChillEvap ( GenEvaporator )
   UA = 1166                        { W/m²K }
   MassHeatCap = 34000              { J/K }
   SuperHeat = 8.1125               { deg C }
   Temperature = -8.7               { deg C }
   Application = WaterTank
   Refrigerant = 22
   Source = APVM8-1000
   Dest = CopelandDLLB
   < Temperature >
   [ APVM8-1000 WaterTank CopelandDLLB ]
END ChillEvap

MODEL CoolingWater ( Environment )
   Temperature = 16.4               { deg C }
   [ APVM8-1000 ]
END CoolingWater

MODEL WaterTank ( FluidTank )
   Mass = 325                       { kg }
   HeatCapacity = 4190              { J/kgK }
   Temperature = 18.7               { deg C }
   < Temperature >
   [ ChillEvap ]
END WaterTank

END WaterChiller
```

Figure 10-3 RefSim input file for water chiller run 1.

10.2.5 RADS V3.1 simulation

The RADS V3.1 application and engine room data files which were used to characterize dynamic run 1 are shown in Figures 10-4 and 10-5. The preparation required to simulate the water chiller using RADS V3.1 was almost identical to that required for RADS V2.2.

```
RRRRRRR
WATER CHILLENK                                    Y
    21      0      1      0
          .00000000          1.00000000        18.70000000          .00000000
       1000.00000000       4180.00000000          .32500000         1.00000000
          1.25000000
            .00100000         20.00000000         1.00000000
        150.00000000            .01000000
RRRRR
NT .7000000E+01 .1000000E+01 .0000000E+00 .2400000E+02
 .2400000E+02 .2400000E+02 .1200000E+01
    22      5
PY-.5000000E+01 .5000000E+00
 .1000000E+01 .4180000E+04
     1
WCHEVAP                                                5      3          .
NMT
 .0000000E+00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E+00
 .0000000E+00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E+00
 .0000000E+00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E+00
 .0000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E+00
 .0000000E+00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E+00
 .0000000E+00
 .0000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00
 .0000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00
 .0000000E+00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00
 .0000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00 .00000000E-00
 .0000000E-00 .00000000E-00 .00000000E-00 .00000000E-00

 .0000000E-00 .00000000E-00 .00000000E-00
 .1000000E-01
     0      0      0      1
 .1166000E+04
Y
    12
        1.000000        12.000000        20.000000          .010000
        1.000000        20.000000        20.000000          .010000
        1.000000          .000000        20.000000         1.000000
        1.000000         8.000000         1.000000
           1
```

Figure 10-4 RADS V3.1 application data file for water chiller run 1.

10.2.6 Simulation results

The results of the four simulation methods are compared with the data measured by Darrow *et al* (1991) for the three dynamic runs in Figures 10-6 to 10-8. Points indicate measured data. The lines marked "Darrow" show the output of the ISIM simulations performed by Darrow *et al* (1991).

The ISIM and RefSim simulations both predicted results which were close to the measured values for the condenser, tank water and evaporation temperatures. This was not surprising given the similarities of the models used by these simulations, but it was reassuring to see that the principles applied in RefSim of separating model concerns and restricting the spread of information between models did not have any detrimental effects upon the final results.

Possible reasons for the differences between predicted and measured temperatures were discussed by Darrow *et al* (1991) (Appendix 4.3).

The RADS V2.2 and V3.1 simulations predicted the water tank temperature almost as well as RefSim and Darrow *et al* (1991), but performed less well in predicting the evaporation and condensation temperatures — the RADS V3.1 prediction of the condensation temperature differing considerably from the measured

```
RRRR
WATER CHILLER COMPRESSOR
B
    22
     1
RRRRR
RRNNR
RRNNR
      1     1
      0     1
      1     0
      0     1     1
      1     0
      0     0
  22.200  -8.700
    .000
    .000
  10.000
     1
CopelandDLLB                                    NNNNNNN
 .5036100E-02 .6400000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
 .0000000E+00 .0000000E+00 .0000000E+00 .9200000E+00-.4870000E-01 .0000000E+00
 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .1500000E-04
 .7500000E-02 .1000000E-03 .2500000E-02 .2005180E+00 .1971050E-03 .2264120E-06
-.3327800E-03 .6669310E-03-.6500000E-02 .1000000E-02 .2000000E-02 .6000000E-02
 .1500000E-01 .1000000E-02 .6000000E-03 .1800000E-04 .1000000E-03 .1500000E-03
 .1000000E-01 .0000000E+00 .0000000E+00 .1000000E-01 .0000000E+00 .0000000E+00
 .1500000E-04 .0000000E+00
Reciprocating compressor on the laboratory water chiller plant
Swept volume = 0.0050361 cu m/s
Note - this data is derived from that of Darrow et al (1991), there is no .CPS
file for this compressor.

N
      1     1YNN
 .1640000E-02 .1640000E-02 .1640000E-02 .7240000E+00 .1000000E-01
 .3000000E-02 .3500000E-02 .9000000E-03 .1000000E-01 .1000000E-01
APVM8-1000                                          1
NMT .7240000E+00      1      2
 .1000000E-01 .1000000E+00 .1000000E-01 .1000000E-01 .1000000E-01 .7000000E-02
 .5500000E-02 .5000000E-01 .2500000E-02 .1000000E-01 .4000000E-02 .1000000E-01
 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
 .1685000E-02 .7240000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
 .0000000E+00
 .1000000E-01 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
 .1000000E-02 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
 .0000000E+00 .0000000E+00 .0000000E+00 .0000000E+00
Water cooled condenser: APVM8-1000
as run on the laboratory water chiller plant.
Note: this data is derived directly from that of Darrow et al (1991) and
so there is no .HTX file for this concenser.

RR
EN
     1
     1
     1
   5.000  30.000   0.555
N
```

Figure 10-5 RADS V3.1 engine room data file for water chiller run 1.

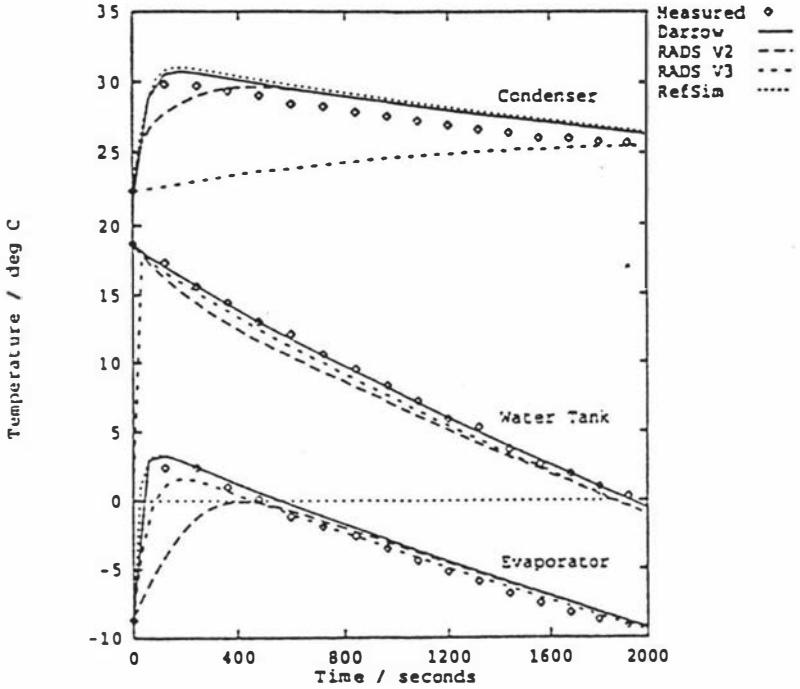


Figure 10-6 Water chiller run 1

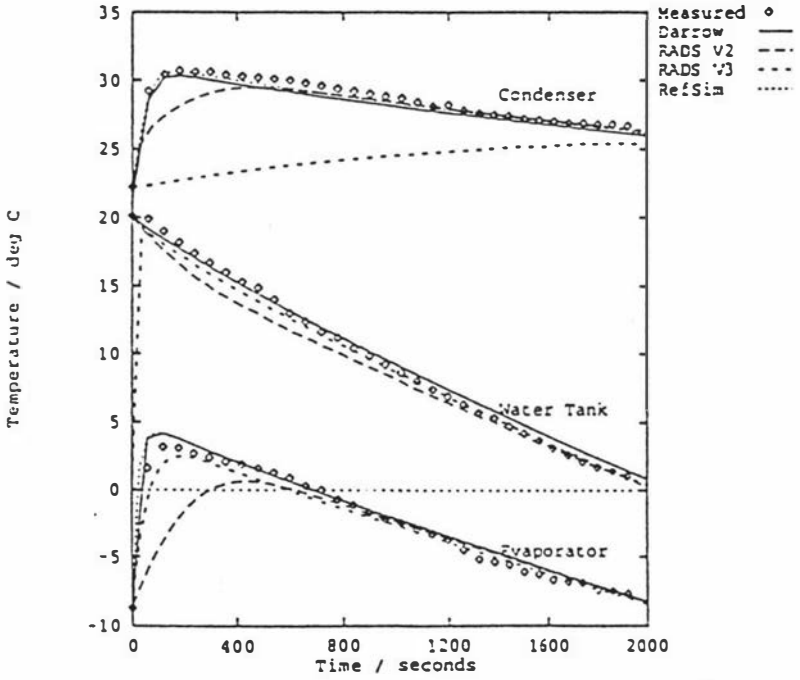


Figure 10-7 Water chiller run 2

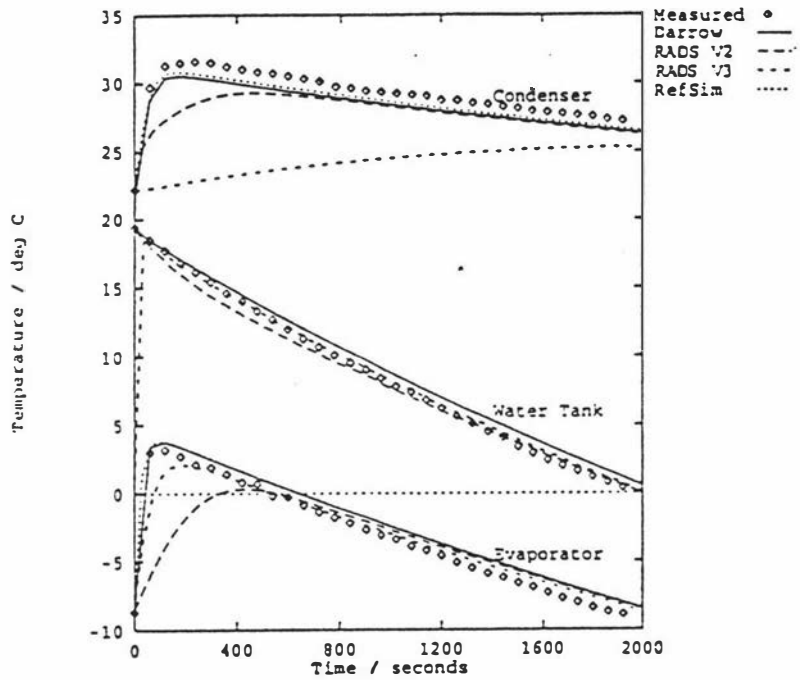


Figure 10-8 Water chiller run 3

data. Darrow *et al* (1991) had noted that it was not necessary to estimate the condenser and evaporator heat capacities precisely for the model to predict results close to the experimental measurements and that "there was no significant change in the simulated curve shapes until the heat capacity estimates were up to 50% smaller or larger than the initial estimates". Nevertheless, it appears that inappropriate estimates of these quantities were the major cause of the observed deviations in the RADS simulation results. Unlike RefSim and ISIM, the RADS simulations did not allow the user to supply these heat capacities as input data, but instead SIMUL made its own estimates of these quantities.

The evaporator heat capacity was estimated by SIMUL V2.2 at 479 kJ/K and by SIMUL V3.1 at 100 kJ/K (compared with the 34 kJ/K estimated by Darrow, 1990, from the evaporator material mass and estimated refrigerant charge). RADS V3.1 therefore performed better than RADS V2.2 but neither was as accurate as the RefSim or ISIM simulations which used the experimentally determined values.

The experimental estimate of the condenser heat capacity was 44 kJ/K and this was used in both the RefSim and ISIM simulations. The RADS V2.2 condenser heat capacity estimate was time-step dependent and for the time-step used for these simulations it was set at 36 kJ/K. The RADS V3.1 program differed from V2.2 in that this physically unrealistic dependency on time-step was removed and the condenser heat capacity was estimated using an heuristic which had been developed for application to large industrial condensers. When this scheme was applied to the small condenser on the water chiller, the heat capacity was estimated to be 1690 kJ/K, resulting in the observed sluggish response.

The accuracy of the RADS simulation evaporator and condenser heat capacities could be improved in an ad hoc manner by modifying the source code for the models and re-compiling, but for the purposes of this study it was decided to use the commercially-released executable code as a base-line. In addition, heat capacity considerations did not entirely explain the discrepancy between the RADS V2.2 prediction and the measured condenser temperature profile. This may have indicated a "bug" in the RADS V2.2 simulation program. Further investigation into the behaviour of the RADS programs was considered to be outside the scope of this project, so no further enquiries were undertaken.

10.3 Alliance Ocean Beach

The Alliance Freezing Company (Southland) Ltd. Ocean Beach meat processing plant was surveyed in April 1989 to gather data for testing full-plant simulations. This plant had previously been surveyed and simulated by Cleland (1983), although there had been considerable changes to the plant since that time.

At the time of the survey, the plant operated six lamb and mutton processing chains with approximately 18500 carcasses being processed each day. The majority of carcasses were frozen whole in ten batch blast freezers. Some butchering was done on the site, and this meat was frozen in a continuous carton freezer, along with

all offal products. Almost all product was exported frozen to various overseas markets, but a small quantity was chilled for local sale.

The ammonia refrigeration plant was capable of operating in several configurations, but predominantly ran in two stages with three surge pots at different temperature levels as shown in Figure 10-9. An array of ten condensers operated at a nominal temperature of 30°C. The air conditioning pot operated at a nominal -9°C, supplying air conditioning and chilling applications — a total load of up to about 1 MW. Three compressors generally operated on the high stage duty between the air conditioning pot and the condensers, using about 600 kW of electricity. The stores pot operated at a nominal -30°C and supplied coldstores and natural convection freezer rooms — a total load of up to 500 kW. Three reciprocating machines compressed vapour from the stores pot to the condenser, typically using about 250 kW of electricity. The freezer pot operated at a nominal -33°C and supplied both continuous and batch blast freezers — a total load of up to about 2.5 MW. Two compression pathways were used for the freezer pot: two screw compressors were used as boosters to the air conditioning pot; one screw and one compound reciprocating machine compressed directly to the condenser, making a total electrical load of about 600 kW.

The simulation of a whole meat processing plant refrigeration system was beyond the capability of ISIM, so the simulation environments used for this simulation and in section 10.4 were limited to RADS V2.2, RADS V3.1 and RefSim. The simulation input data for the full plant simulations are not listed in this thesis because of their bulk. For each of the three simulation environments, the simulation input data to represent each of the two meat plants totalled about 200kB per simulation.

The simulation input data comprised physical measurements of rooms, doors, insulation, evaporator sizes, counts and dimensions of product items. Compressor characteristics were derived from manufacturers' data. Evaporator characteristics were estimated from physical dimensions using correlations derived from the data of Edwards (1976) who tested a typical New Zealand meat industry evaporator. The

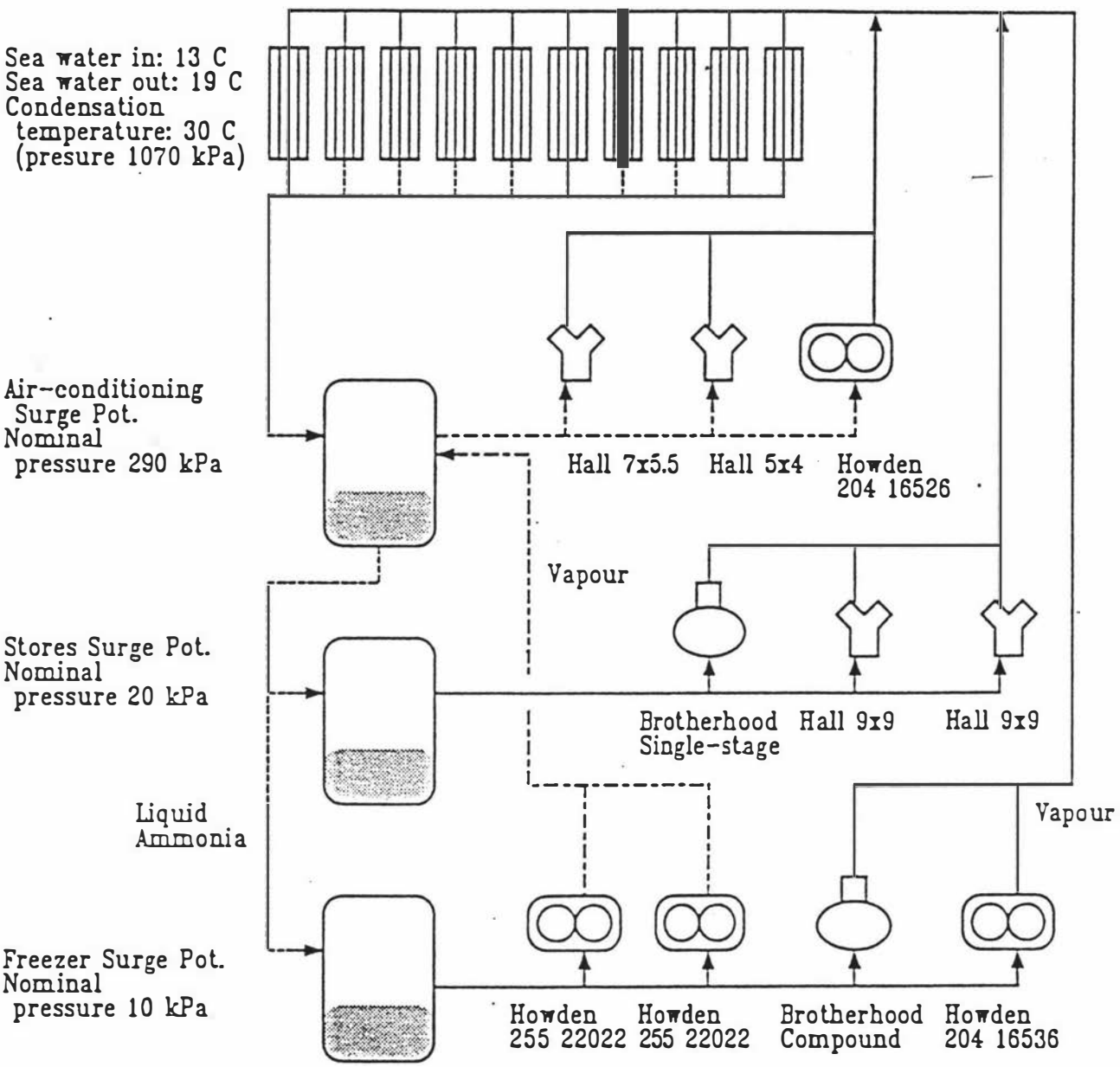


Figure 10-9 Alliance Ocean Beach plant engine room configuration

amount of work required to measure the performance of each refrigeration system component as had been done for the water chiller of section 10.2 would have been prohibitive for such a complex plant as Ocean Beach so some of the input data were estimated and average values assumed for other parameters. For example, while the sizes of doors were measured, their opening frequencies and total times open were estimated by observation rather than by timing door openings over long periods. The input data for the large plant simulations were therefore not as accurate as they had been for the water chiller and this could be expected to have a significant effect upon the overall simulation accuracy.

The simulation performed in each case was of the plant operations during April 10 1989 — a day which was chosen because it was typical of the conditions encountered during the survey. While application temperature data was logged automatically, the engine room temperature data were collected manually by the plant operators. This manual logging process resulted in temperatures being logged inconsistently, with no data being recorded at all during some shifts. The engine room temperature data was therefore only indicative and was not used for comparison purposes.

10.3.1 RADS V2.2 simulation

Ocean Beach was first simulated using RADS V2.2. The simulation of the chosen day used 25 applications input files and one engine room file. The RADS simulation program, SIMUL assumed that the plant operated in a cyclical manner with each set of conditions repeating every 24 hours. This corresponded well with the measured data for some applications, but not for all.

The low pressure surge pot operated a split suction compression system with two compressors discharging into the condensers and two booster compressors discharging into the high pressure surge pot. RADS V2.2 was capable of representing this operating mode but not the strategy which was employed to control the temperature of this vessel. When the load on the low pressure pot was small, the

operators turned off the two low pressure ratio machines in preference to shutting down the high pressure ratio machines.

This control strategy was used due to electrical supply constraints, even though it was expected to have been less energy-efficient than shutting down the high pressure ratio machines first. RADS V2.2 did not support this unusual control strategy. To simulate Ocean Beach it was necessary to patch SIMUL V2.2 so that it shut down the low pressure ratio machines first when the refrigeration load was small. This patched version of SIMUL V2.2 could then only be used to simulate Ocean Beach.

10.3.2 RADS V3.1 simulation

RADS V3.1 was developed after the Ocean Beach survey and RADS V2.2 simulations had been carried out. It therefore included a modified form of the split suction vessel control strategy which allowed the strategy used at Ocean Beach to be simulated directly. Version 3.1 used an early version of the product heat load model described in Chapter 4 for the product (although not for other thermal objects in the manner of RefSim, section 9.7) and integrated all of its ODEs in parallel rather than integrating application and engine room ODEs separately as had been done in RADS V2.2.

As with the RADS V2.2 simulation, the RADS V3.1 simulation of the chosen day used 25 applications input files and one engine room file.

10.3.3 RefSim simulation

In order to make the RefSim simulation directly comparable to the RADS simulations, the RefSim input file was largely generated automatically using the program *App2Ref* described in section 8.4.1. This meant that while RefSim was capable of simulating the non-cyclical behaviour observed in the plant, the plant description generated from the RADS input data assumed a 24 hour operating cycle

and so that feature of RefSim was not used. Even if a non-cyclical simulation had been desired, some of the plant data that would have been needed for a non-cyclical simulation were not available as the survey was carried out while RefSim was in the earliest stages of design.

The RefSim simulation of Ocean Beach comprised 729 separate models and required the solution of 309 ODEs.

10.3.4 Comparison

There were some difficulties involved in comparing the non-cyclical plant measurements with the cyclical simulation output. The most important non-cyclical applications were batch freezers. Of the twelve batch freezer rooms in the plant, six would typically be used on any one day for freezing fresh product, with the remainder finishing AC&A (section 5.3) production runs with cycle times of more than 24 hours, temporarily storing product or empty. A freezer room used one day for freezing would often be storing product, or be empty the next day. The lowest load on these batch freezers and on the whole system was typically at about 05:00 hours each morning, so it was decided that the most appropriate comparison with the simulation runs could be made by using the data from 05:00 on 10 April to 05:00 on 11 April 1989.

The temperatures predicted by the three simulation programs are compared with the measured data for a selection of simulated components are plotted in Figures 10-10 to 10-16. The measured data discontinuity at 05:00 hours on some of the plots indicates where the plant did not correspond to the cyclical assumption.

One feature of RADS V2.2 was that it did not simulate applications while the refrigeration supply to them was turned off. This improved the simulation accuracy for those applications because they were always re-started at the correct temperature when the refrigeration was turned back on. It also improved the simulation rate because a number of applications were not simulated during part of each day. It did mean that the RADS V2.2 simulations of specific applications could not be compared

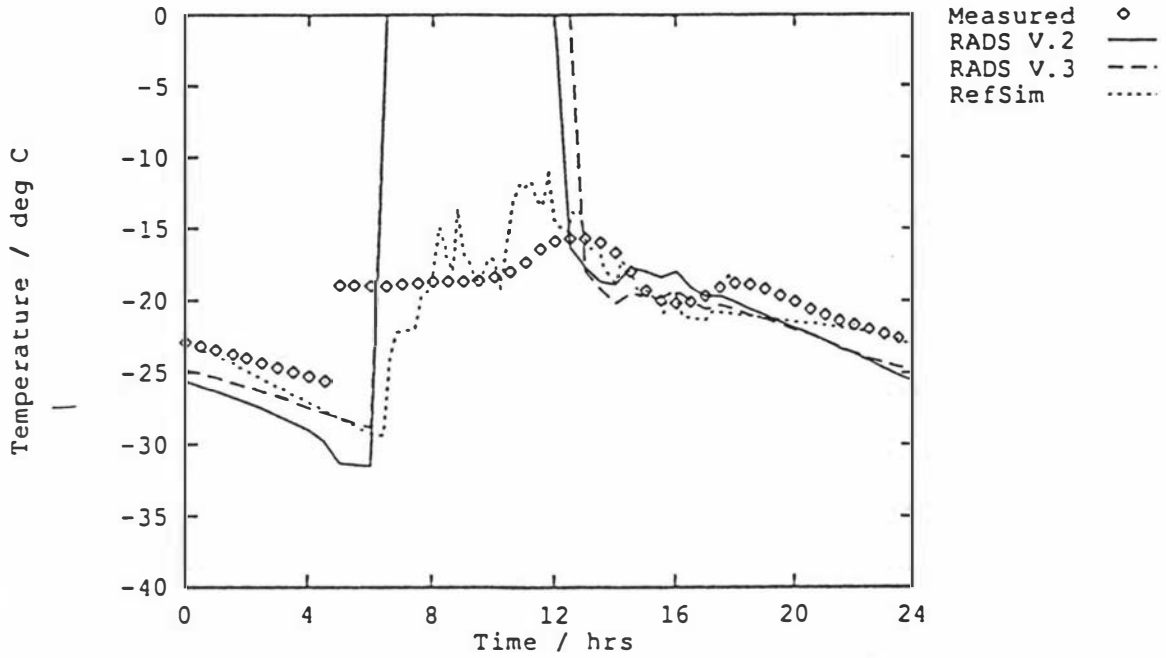


Figure 10-10 Ocean Beach freezer room 11

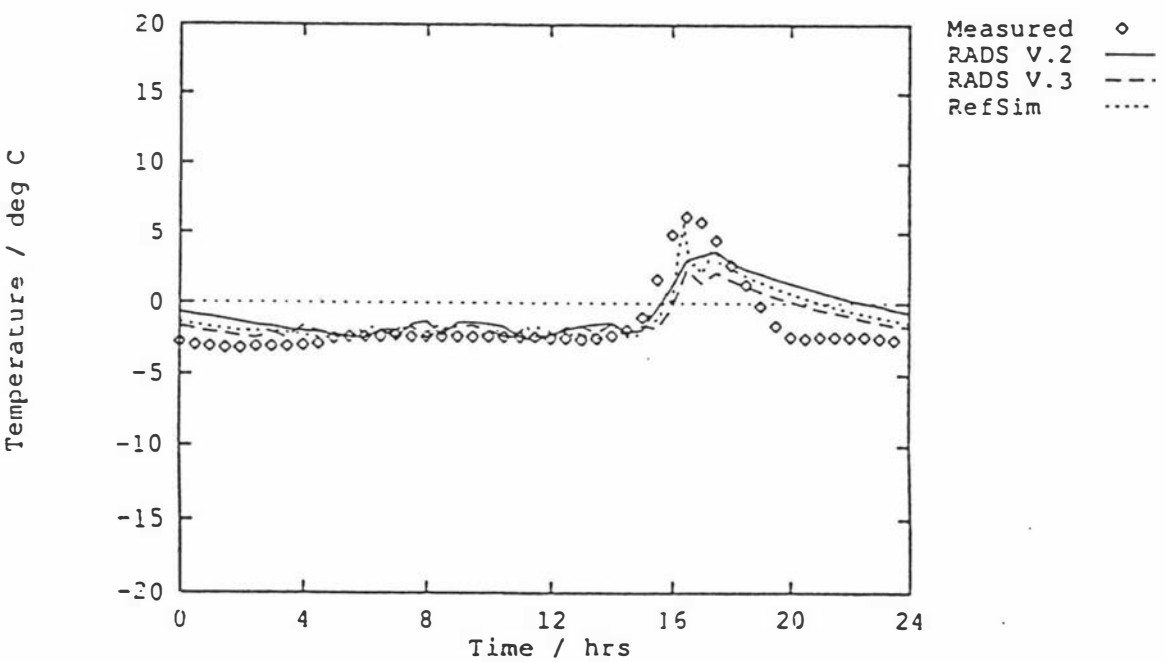


Figure 10-11 Ocean Beach chiller room 16

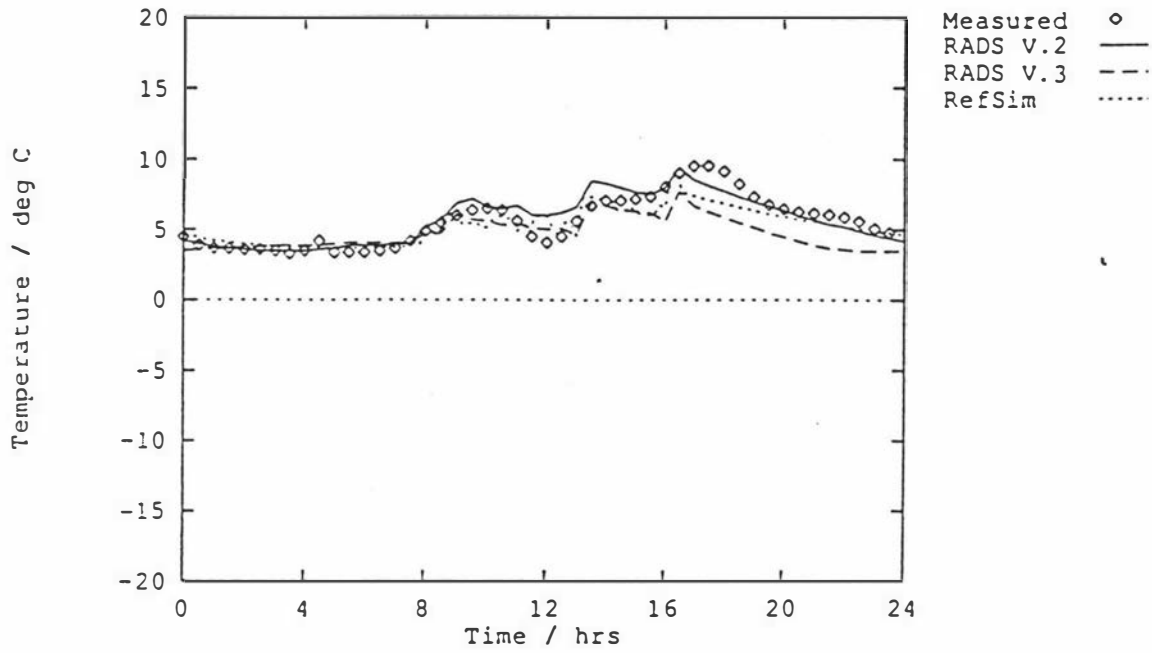


Figure 10-12 Ocean Beach chiller room 20

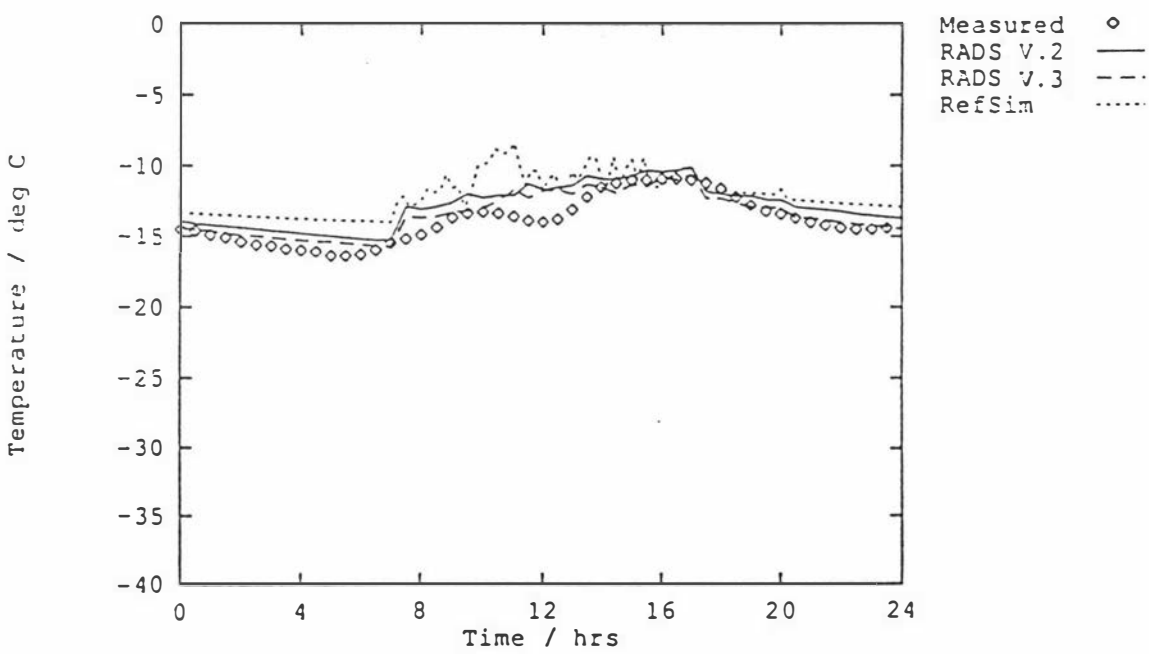


Figure 10-13 Ocean Beach pallet store

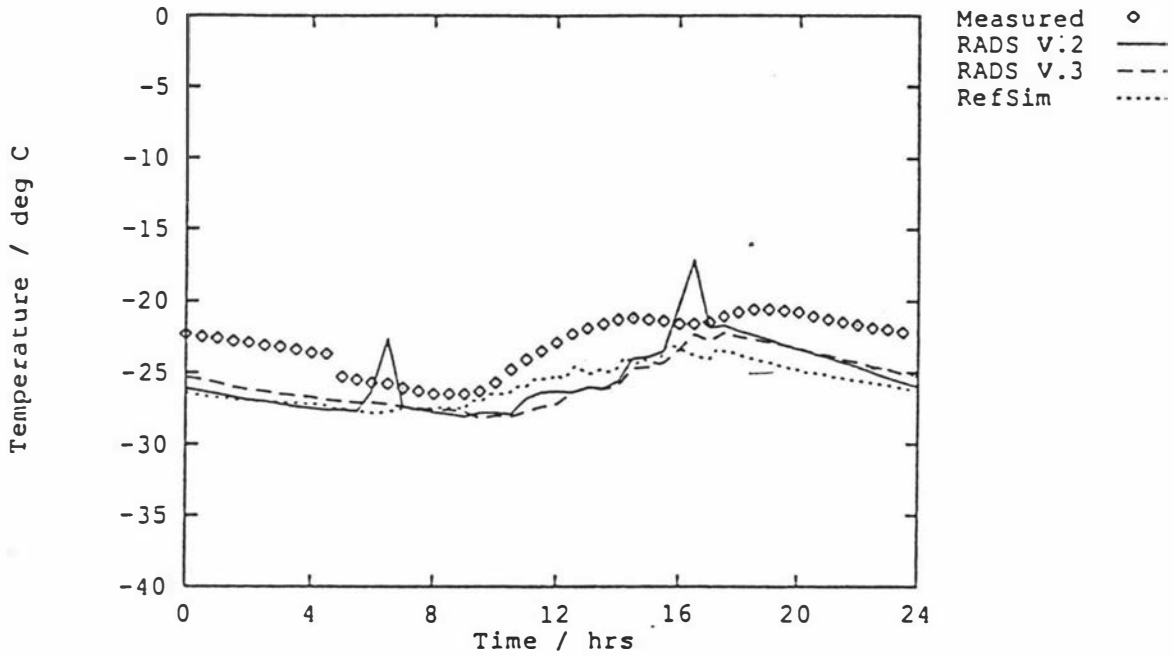


Figure 10-14 Ocean Beach carton freezing tunnel

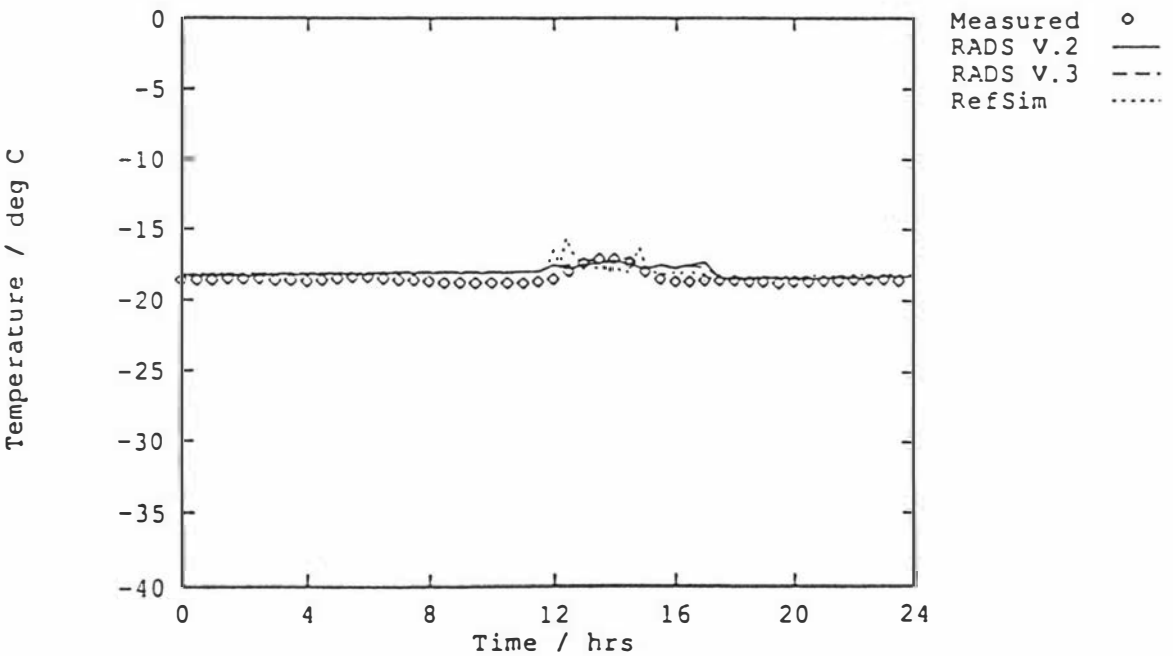


Figure 10-15 Ocean Beach carton store

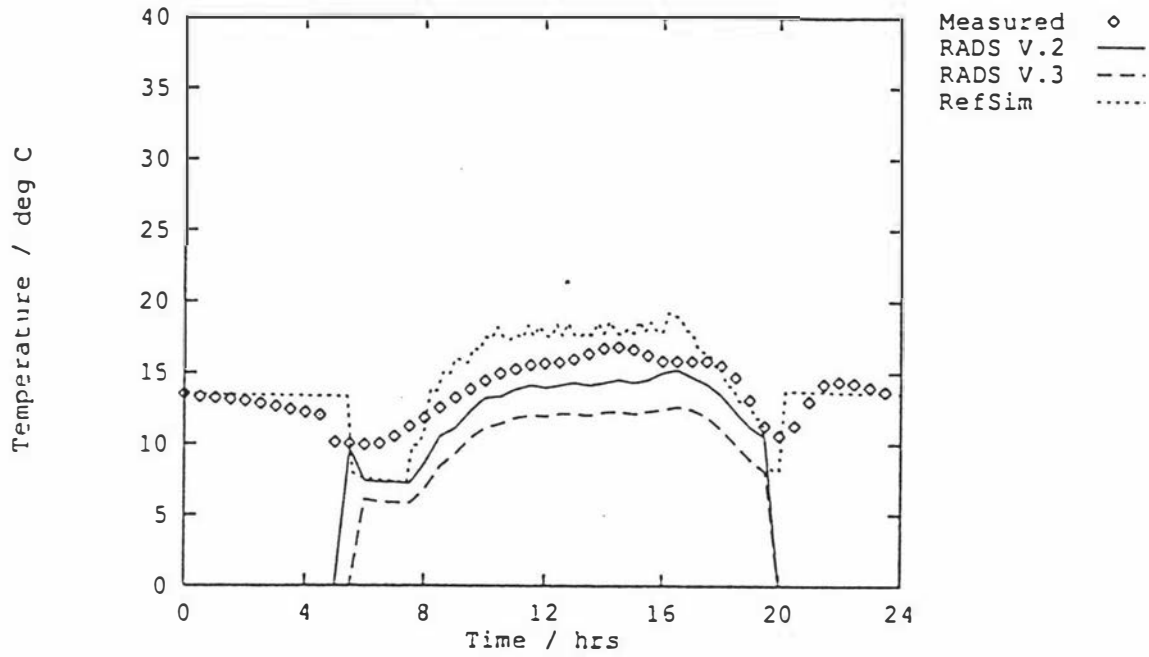


Figure 10-16 Ocean Beach cooling floor

with the real measurements and RefSim simulations while those applications were turned off. This problem could have been resolved with RADS V3.1 by using the alternative application file generator program ROOM rather than APPLICS to create the application files. ROOM-based applications simulated under RADS V3.1 were simulated throughout the day whether or not they were refrigerated. Unfortunately, ROOM was not compatible with RADS V2.2, so converting ROOM-generated applications for use by the RADS V2.2 SIMUL program would have been very difficult. APPLICS was therefore used to generate application files for both RADS versions.

The RADS V2.2 simulation results file omitted any applications which were off during an output period. The RADS V3.1 results file noted "Application is off" for any applications which were off during an output period. In the graphs plotted in this chapter, applications which were off are shown as having temperatures of 0°C.

RefSim simulated the applications while they were not supplied with refrigeration. The air temperatures predicted by RefSim were less accurate when the

applications were turned off, probably because the assumption of well mixed air was less appropriate when the evaporator fans were turned off.

The compressors and condensers in the Ocean Beach plant were controlled manually and therefore the control strategy tended to vary with the operator on duty. This variation and the judgement used by the operator in making control decisions was difficult to represent using the control strategies available in either of the RADS simulation programs. While RefSim was flexible enough to add novel control strategies without difficulty, the process of developing those strategies so that they adequately characterised the operator's behaviour would have required extensive work in the area of expert systems. The RefSim simulation therefore retained control strategies similar to those used by the RADS simulations.

10.3.4.1 Blast freezers

The temperatures in batch freezers (e.g. Figure 10-10) were predicted better by RADS V3.1 and RefSim than by RADS V2.2, almost certainly due to the new product model. In particular, the predictions at the start and end of the process were both improved. The initial temperature change in batch freezers after loading product was predicted well by RefSim, probably due to the integration error control exercised by RefSim.

RefSim predicted the air temperature during the latter half of the freezing process well due to its sophisticated treatment of the sub-cooling product heat load. The RefSim prediction of the air temperature during the period that the freezer was switched off was surprisingly good — despite the *Room* model assumption of a well-stirred air volume being inappropriate at that time due to the evaporator fans being switched off.

The effect of door openings on the air temperature in freezer room 11 was exaggerated by RefSim. While the freezer was switched off, this was understandable due to the assumption of a well-stirred air volume, but the effect after the room was started was still greater than that observed in the measured data. This indicated that

there may have been a deficiency in the interaction between the *Door* and *Room* models.

10.3.4.2 Chiller rooms

The temperature profile of chiller room 16 (Figure 10-11) was generally predicted well by RefSim. Both the initial peak temperature at loading and the rate of temperature drop after loading were underpredicted.

Some of the assumptions made in developing the chilling stage of the new ODE food product heat load model were identified in section 4.5 as being inappropriate during the early part of the chilling process. During testing, the model was found to predict poorly at the start of the process (section 6.3.1). The unavailability of accurate values for the E parameter during chilling (section 4.5) may also have contributed to model inaccuracy under these circumstances.

The temperature profile of chiller room 20 (Figure 10-12) was predicted better than that of chiller room 16 (Figure 10-11). The air temperature in chiller room 20 was higher than that in chiller room 16 with the result that the product chilling heat load would have been relatively smaller than in room 16 and would therefore have had a lesser effect upon the air temperature dynamics.

10.3.4.3 Natural convection cold stores

The pallet store (Figure 10-13) was the most dynamic of the natural convection cold stores in the Ocean Beach plant. The air temperature profile was not simulated well by any of the simulation environments.

The pallet store was not adequately refrigerated and it was the opinion of the plant operators that its suction line may have had a large, but unknown, pressure drop. Many of its natural convection coils appeared to be starved of refrigerant during the daytime. This situation could not be simulated using the "thermal" model type.

There were other difficulties in obtaining accurate input data for this application. The heat flows through the insulation panels were calculated from the published properties of the insulating materials, but a multiplying factor had to be applied for heat which bypassed the insulation and for reduced insulating effectiveness due to water-logging. For insulation which was known to be in bad condition, multiplying factors of up to 6 were used but there was considerable uncertainty about the most appropriate values.

RefSim again exhibited problems with the interaction between the *Door* and *Room* models.

10.3.4.4 Continuous carton freezing tunnel

The trends in the carton tunnel air temperature profile (Figure 10-14) were generally well simulated. The average temperature level was, however, consistently underpredicted.

The air temperature in this freezer was consistently within about 10°C of the nominal freezer pot temperature. A small difference between the predicted and actual vessel temperatures would have had a disproportionate effect on the simulated air temperature. For example, if the simulations had underpredicted the vessel temperature by 2°C, the effective refrigeration capacity in the carton freezer would have been approximately 20% greater than had actually been the case. Since the carton freezer temperature was uncontrolled, this would have tended to hold down the carton freezer air temperature as well. Errors in predicting the vessel temperature were quite possible in view of the difficulty of exactly representing human operator decisions with the automatic control algorithms available in the simulation environments. The general underprediction of the air temperature was therefore considered to be less important than the trend in this case.

10.3.4.5 Forced convection coldstores

The carton store air temperature profile shown in Figure 10-15 was typical of several of the cold stores at Ocean Beach. They were well supplied with refrigeration and well controlled. There were therefore no notable features in the temperature profile.

10.3.4.6 Cooling floor

The air temperature profile of the dressed lamb cooling floor (Figure 10-16) was not well predicted. Sensitivity analysis showed that the profile was dominated by the dynamics of its heavy concrete floor. The differing treatments of that floor between the simulation environments caused the observed differences between the measured and simulated air temperature profiles. Apart from this effect (which caused a vertical shift in the predicted air temperature profile when the capacity of the evaporators in this room was exceeded), the trends in the air temperature profile were approximately correct.

RefSim predicted the starting temperature of the concrete floor by simulating it continuously from the initial estimate provided in the input data while the RADS programs reset the concrete temperature to the input data value at the start of each day. RefSim probably overestimated the concrete temperature because it ignored the existence of some cold stores underneath the concrete floor which cooled the floor from that side (see section 9.6).

The difficulties with the chilling model discussed in section 10.3.4.2 may also have contributed to the poor accuracy of the predicted air temperature profile.

10.4 AFFCO Horotiu

The Auckland Farmers' Freezing Co-operative Horotiu meat processing plant was surveyed during May 1990 (in association with the Meat Industry Research

Institute of New Zealand refrigeration group) to gather further data for testing full-plant simulations.

At the time of the survey, AFFCO Horotiu was a combined beef and sheepmeat processing facility, serving both export and local demand. One beef chain and two lamb/mutton chains killed and dressed product which was predominantly chilled and then boned or cut. Some lamb was frozen in carcass form in a continuous carcass freezer, but most of the product was packaged in cartons of about 27 kg capacity and frozen in a continuous carton freezer. Small quantities of product were shipped out as chilled fresh meat for local sale and for air-freight overseas.

The bulk of the plant refrigeration load was supported by the ammonia refrigeration plant shown in Figure 10-18. The plant was normally run as a three stage system, with the LP and LLP pots coupled together to make a two stage system during the weekend when the carton freezer load was smaller than during the working week. A total of 14 compressors were available for use on various compression pathways. Ten condensers of various types maintained a condensing temperature of about 30°C. The HP pot operated at a nominal -8°C to supply the lamb cooling floor, beef quartering floor, beef and abattoir chillers. The LP pot operated at a nominal temperature of -28°C to supply the cold stores, continuous lamb carcass freezer, and the four batch "fast freezers" (which were, in fact, usually used for lamb chilling). The LLP pot nominally operated at temperature of -40°C and supplied only the two continuous carton freezers.

Two glycol secondary refrigerant systems were cooled by ammonia from either the HP or LP pots and in turn cooled the beef boning room (beef glycol system) and the lamb cutting room and the lamb cutting room chillers (lamb glycol system).

The experience developed during the earlier Ocean Beach survey meant that more care was taken to ensure that all of the important data was collected during the whole period. In addition, the opportunity was taken to automatically log all available refrigerated room temperatures so there were more measured temperatures for comparison with the simulated results.

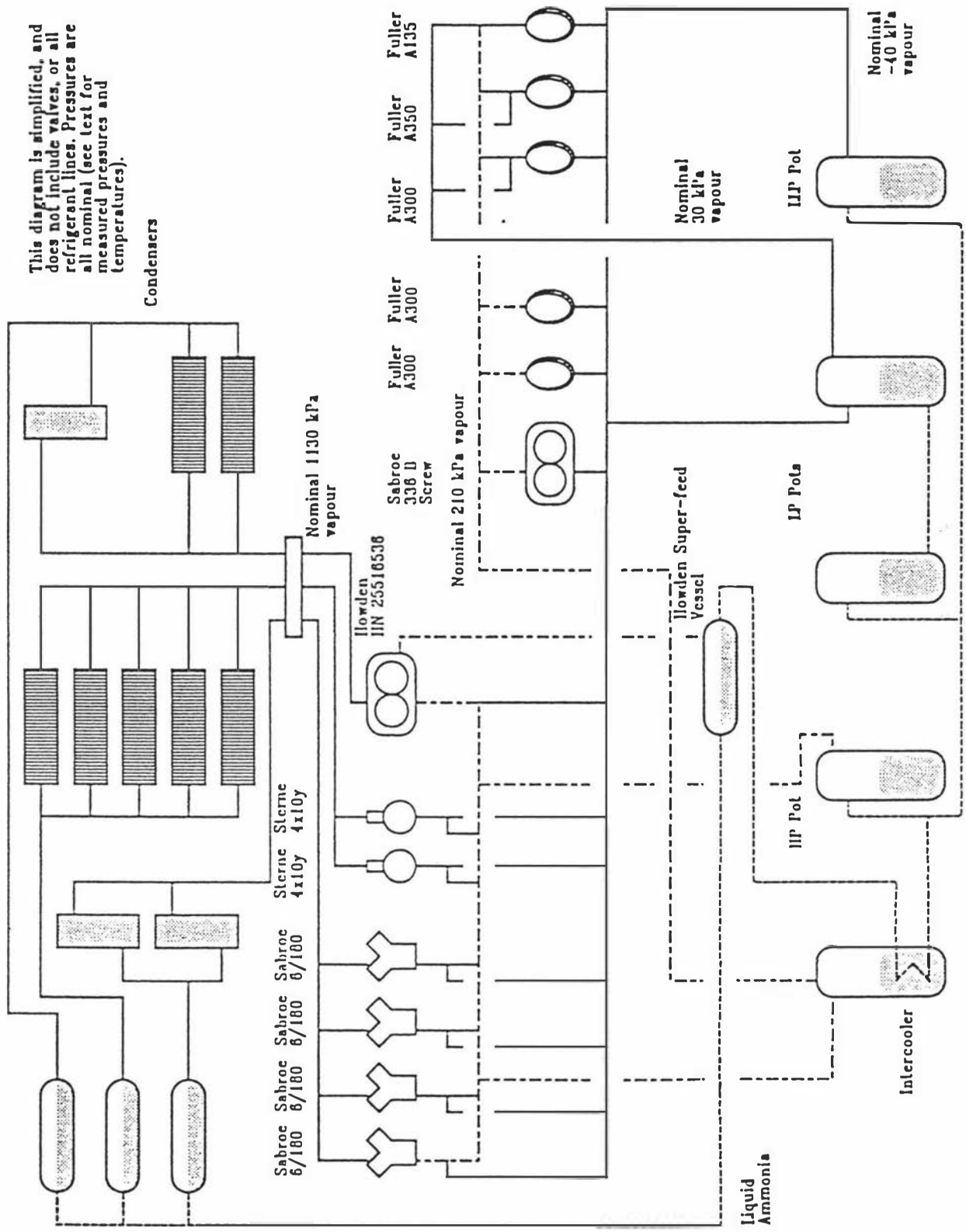


Figure 10-17 AFFCO Horotiu plant engine room configuration

As in section 10.3, the plant was simulated using RADS V2.2, RADS V3.1 and RefSim. The data required to specify these simulations were similar in quantity to those for Ocean Beach. The differences between the two plants made a simulation of AFFCO Horotiu a substantially different test of the simulation programs from Ocean Beach. In addition, fully automatic logging made the data gathered at Horotiu more reliable and less subject to human error. In particular, the engine room temperature data were all available for Horotiu during the whole period of the survey, unlike the Ocean Beach engine room data.

10.4.1 RADS V2.2 simulation

AFFCO Horotiu was first simulated using RADS V2.2. Some difficulties were encountered when setting up models of the beef carcass chillers. These chillers were run all day but they were loaded at one or more discrete times. The RADS room type which most closely resembled this mode of operation was the Type 15 Combined chiller and coolstore, so the beef chillers were modelled using this room type. The RADS APPLICS simulation set-up program did not accept that the specified operating behaviour was appropriate for the Type 15 room and so did not produce a simulation file for input to SIMUL. After attempting to model the beef chillers with other room types, the difficulty was finally resolved by patching the source code of APPLICS so that it would accept the beef chiller operating regime.

Few other difficulties were encountered and the simulation executed as expected once the beef chiller problem was resolved.

10.4.2 RADS V3.1 simulation

The RADS V3.1 simulation of AFFCO Horotiu executed with fewer problems than the V2.2 simulation. In particular, data preparation for the beef chiller rooms was not hindered by the V3.1 APPLICS program as it had been by the V2.2 APPLICS.

10.4.3 RefSim simulation

The RefSim models described up to this point could simulate all but one of the component types in the AFFCO Horotiu plant secondary refrigerant system. The ammonia-secondary refrigerant heat exchangers and secondary refrigerant reservoirs could be adequately represented by the *GenEvaporator* and *FluidTank* models. The pipeline model was general enough to represent secondary refrigerant as well as refrigerant pipelines. The only components which could not be adequately represented were the air-brine heat exchangers used to cool the air in some air-conditioning and chilling applications, so the RADS brine-refrigerated air cooler model was implemented under the name *RADSBrCooler*.

The RefSim simulation of AFFCO Horotiu comprised 702 separate models and required the solution of 354 ODEs. The larger number of ODEs when compared with the Ocean Beach simulation resulted from the fact that Horotiu included two continuous carton freezers and one continuous carcass freezer. The simulation input file prepared by *App2Ref* assumed that the product in a continuous freezer could be adequately represented by a set of ten batches which were loaded at even intervals during the loading period of the freezer each day. Each of the continuous carton freezers therefore required the solution of 40 ODEs for the product alone (one for the enthalpy and one for the frozen depth of each product batch) as they had 48 hour residence times, while the continuous carcass freezer (with a 24 hour residence time) required 20 ODEs.

10.4.4 Comparisons

The period of the survey which was used for comparison with simulation results stretched from 05:00 on 3 May 1990 to 05:00 on 4 May 1990. Unlike the Ocean Beach survey, the condenser and surge pot temperature data were collected for the same period and can therefore be compared directly with the simulation output.

Comparisons between measured data and simulation outputs are shown in Figures 10-18 to 10-24.

Some difficulties were again encountered because the plant operation was not truly cyclical, even though the simulations assumed it to be so. As in section 10.3, no attempt was made to simulate non-cyclical operation using RefSim in order to maximise comparability with the RADS simulations.

10.4.4.1 Refrigerant vessels

The condensing temperature (Figure 10-18) was well controlled on this plant. The RADS V.2 simulation exhibited some instability in its predictions, but RefSim and RADS V.3 predicted the temperature profile well.

The HP pot (Figure 10-19) was even more tightly controlled than the condenser. The tight control resulted in frequent control action as actually occurred on the plant. This was well simulated by RefSim due to its use of state events and integration error control to isolate the effects of state changes, but the sudden changes in control action caused some instability in both of the RADS simulations.

The temperature profile of the LLP pot is shown in Figure 10-20. Both the LP and LLP pots were controlled loosely in practice and they were allowed to drift up and down according to individual operators' strategies which could not be adequately represented by any simple automatic controller of the sort that could be simulated. The pot temperature was therefore controlled to a typical minimum temperature and allowed to rise if the compressor capacity was exceeded at any time.

In fact, the simulated compressor capacity was not exceeded and the temperature remained steady for most of the simulation, unlike the actual situation. This indicated that the model compressors performed better than the compressors in the actual plant. It may be that, due to wear, the manufacturers' data from which the compressor model input data was obtained did not accurately represent the characteristics of the compressors once they had been in use for many years. It was impractical to carry out a complete compressor test for each of the compressors on

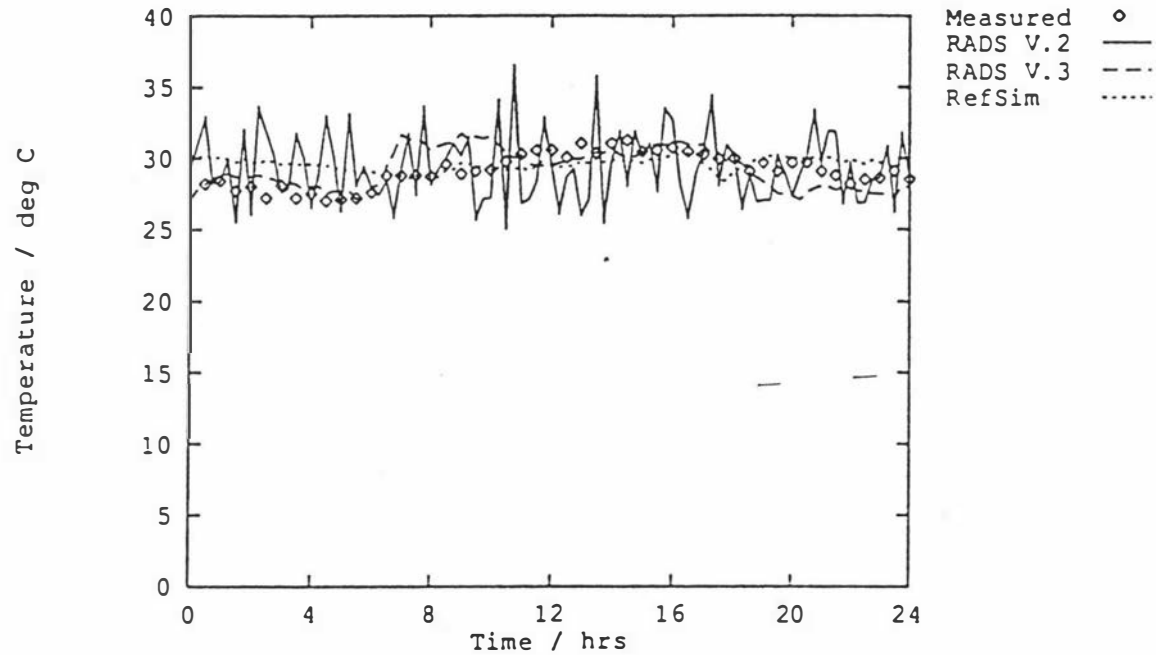


Figure 10-18 Horotiu condensing temperature

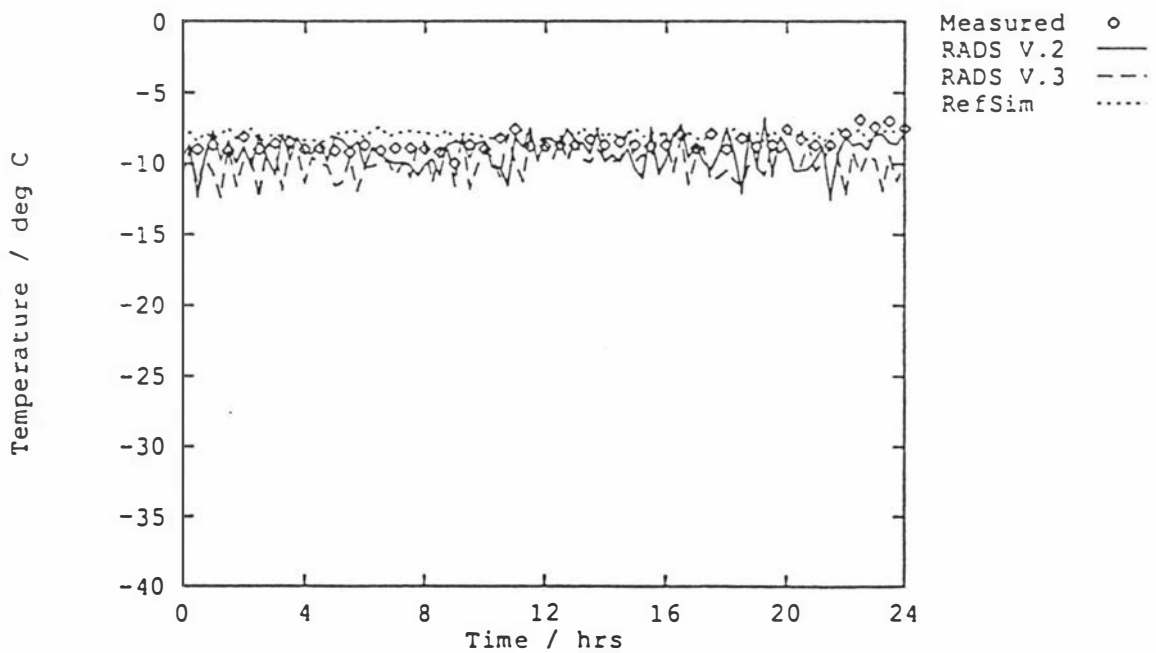


Figure 10-19 Horotiu HP pot

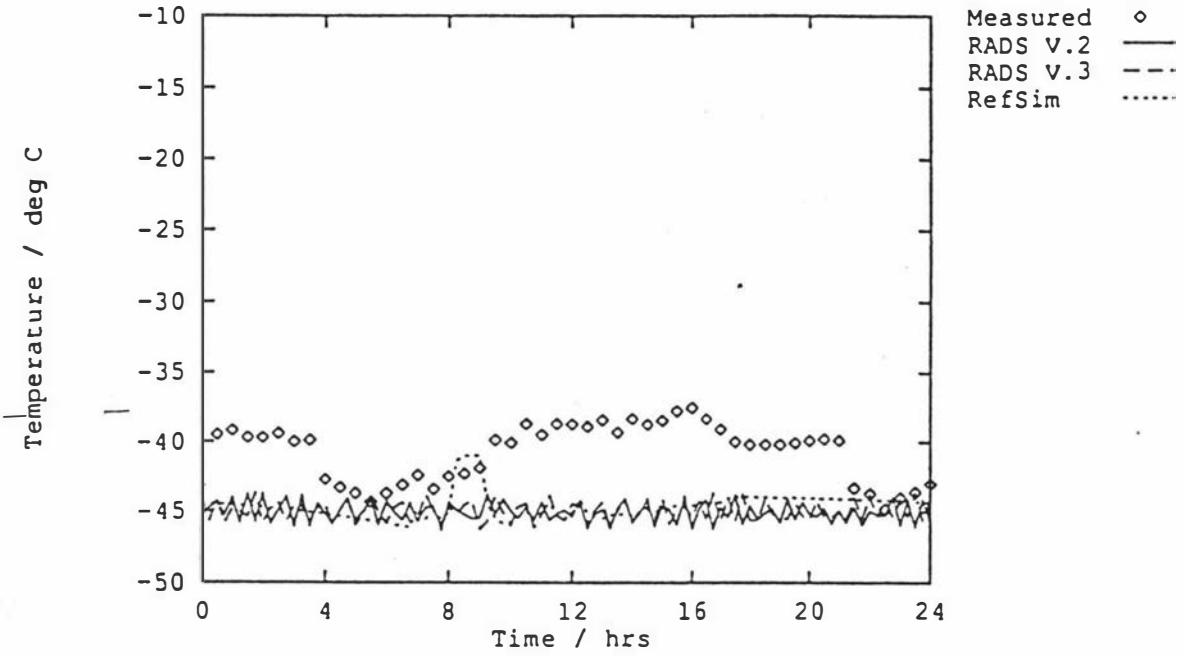


Figure 10-20 Horotiu LLP pot

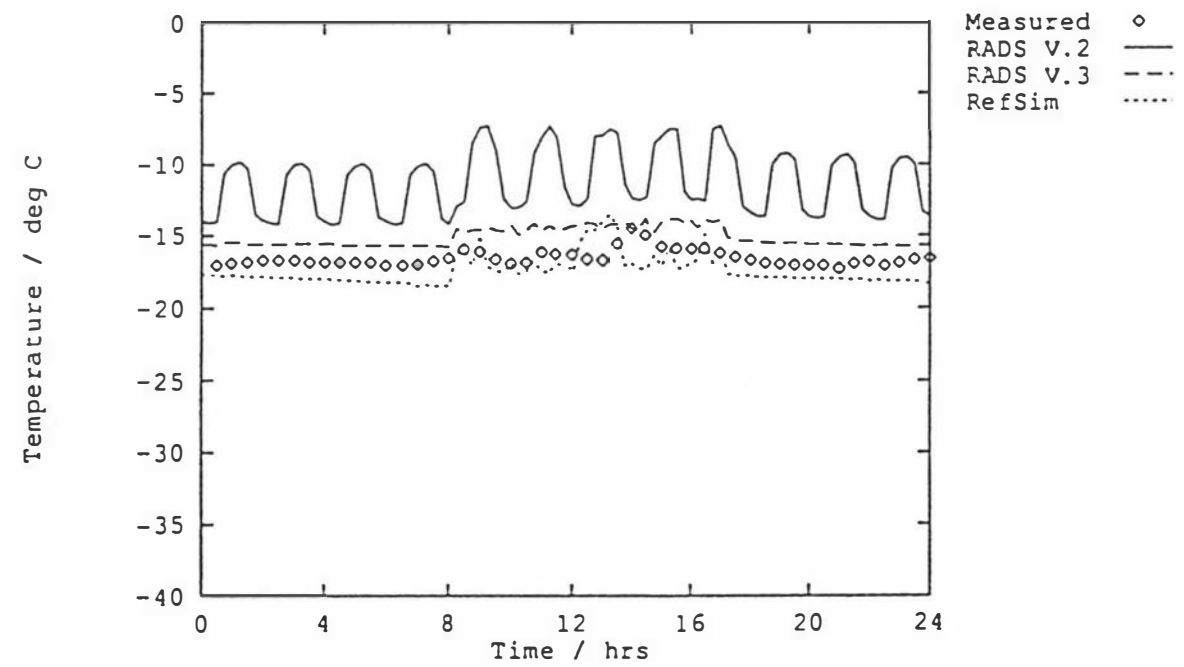


Figure 10-21 Horotiu '69 store bottom floor

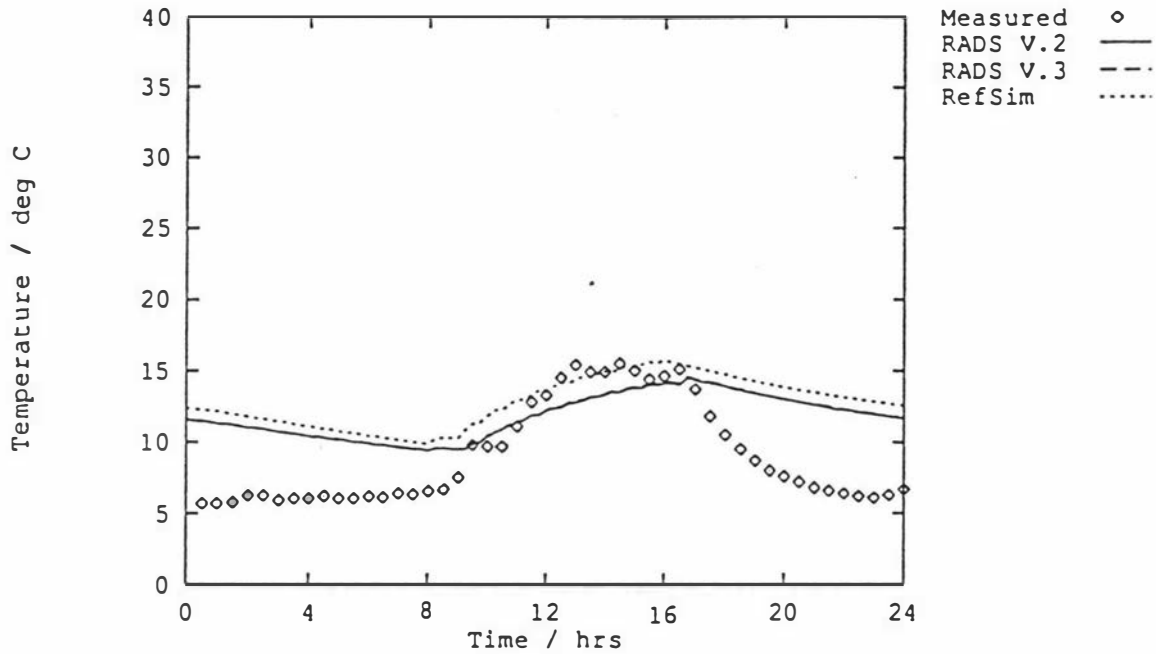


Figure 10-22 Horotiu cooling floor

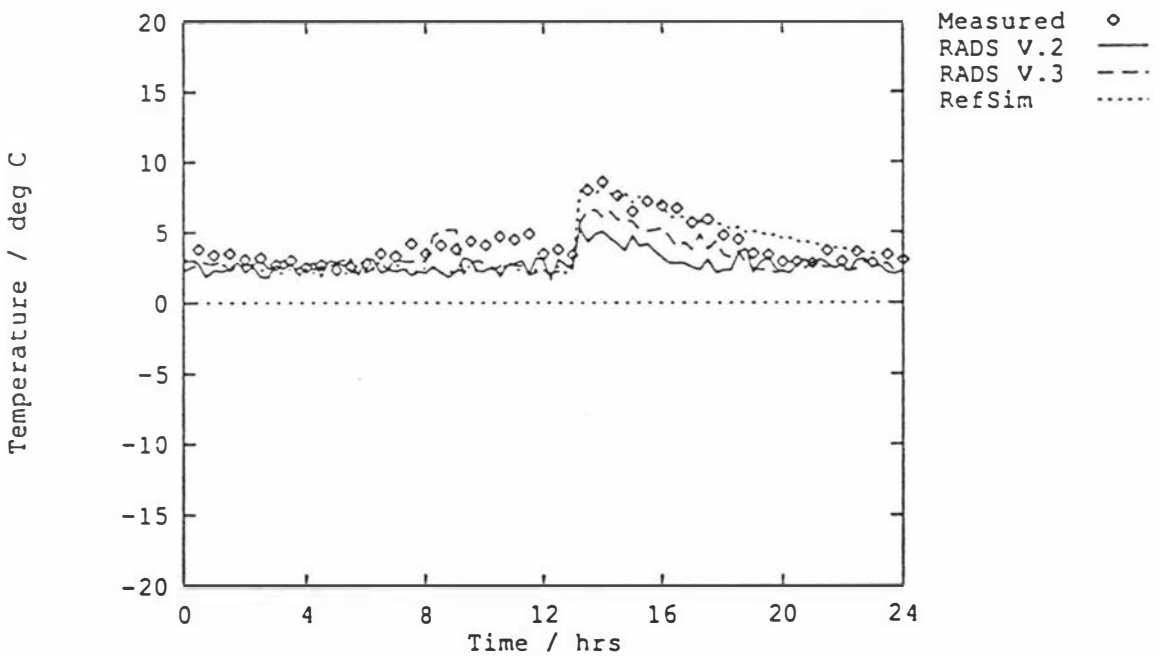


Figure 10-23 Horotiu beef chiller 1A

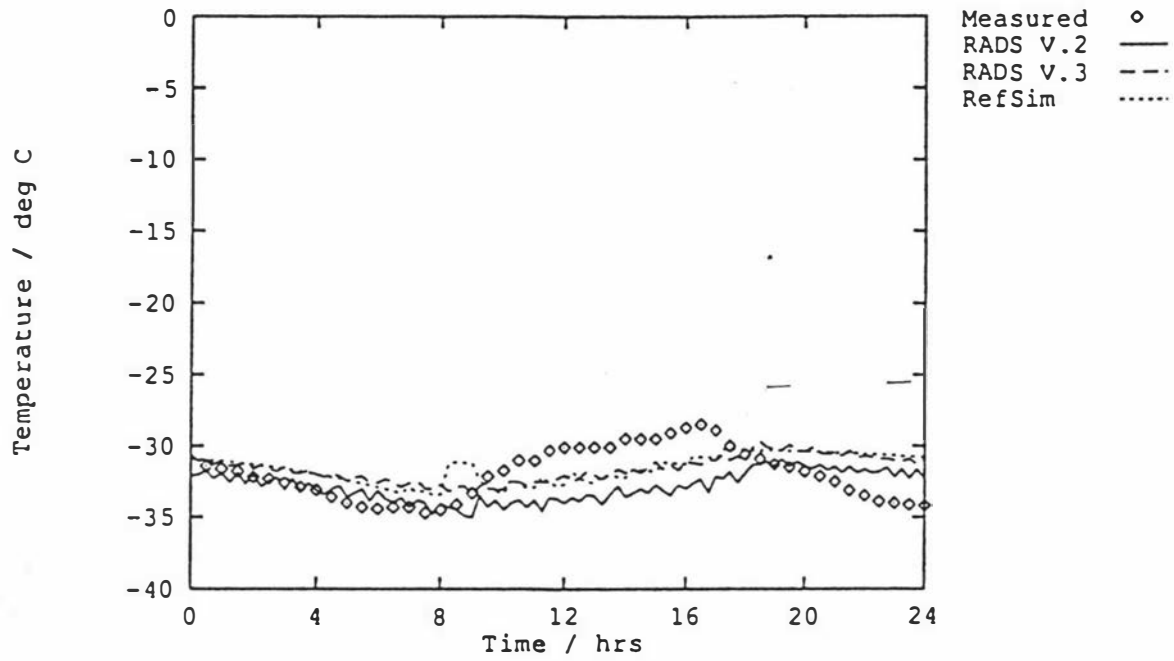


Figure 10-24 Horotiu continuous carton freezer 2

the plant and the only other alternative would have been to apply an arbitrary de-rating factor to the compressor characteristics so the simulation was based upon the manufacturers' data despite its apparent problems.

10.4.4.2 Cold stores

The temperature profile of the '69 store bottom floor (Figure 10-21) was typical of the forced convection cold stores at AFFCO Horotiu. RefSim predicted the temperature profile well. The effects of door openings on the temperature profile were comparable to those measured during the day of interest. This provides an interesting comparison with the Ocean Beach pallet store which was predicted poorly during periods of door load due to the inappropriateness of the well-stirred volume assumption to the natural convection store. The profile predicted by RADS V.2 for '69 store showed instability.

10.4.4.3 Cooling floor

The air temperature profile of the Horotiu dressed lamb cooling floor is shown in Figure 10-22. All of the simulation environments predicted poorly. The problem appears similar to that experienced with Ocean Beach chiller room 16 (section 10.3.4.2) with the simulated air temperature peak lower than the measured peak during loading, and the subsequent simulated air temperature remaining higher than the measured value.

In contrast with the Ocean Beach cooling floor (section 10.3.4.6), in which carcasses only spent about three hours before being transferred to freezers, the carcasses on the Horotiu cooling floor were held overnight before cutting or freezing the next day. The large number of lamb carcasses in the Horotiu cooling floor meant that there was a significant product cooling load through the whole day and the chilling product was always a major influence on the air temperature profile. The weaknesses of the product heat load model chilling stage were therefore made particularly apparent in this application.

10.4.4.4 Beef chiller

Figure 10-23 shows the air temperature profile in beef chiller 1A. This was typical of the eight beef carcass chillers. The loading peak was well predicted by RefSim in spite of the possible weaknesses in the product chilling heat load model discussed in section 10.3.4.2.

One significant difference between the Horotiu and Ocean Beach chillers was the E factor of the product. For the Ocean Beach chiller (which contained lamb carcasses) it was found in section 6.4.2 that $E = 1.48$ (assuming that E for chilling equalled E for freezing), and for the Horotiu chiller (which contained beef sides) it was estimated that $E = 1.25$. It may be that the estimated E factor used for the Horotiu chiller happened to be closer to the correct chilling E than was the freezing E used for the Ocean Beach chiller.

Once the air temperature was back under control, it was controlled by turning the chiller fans on and off. As with the tight refrigerant vessel control discussed in section 10.4.4.1, this induced some instability in the RADS simulations.

10.4.4.5 Continuous carton freezer

The temperature profile of carton freezer 2 (Figure 10-24) was predicted adequately by all of the simulation environments. Other than product heat load, the principle influence on the air temperature in this room was the temperature of refrigerant in the LLP pot. The simulated pot temperature was often lower than the measured temperature for the reasons discussed in section 10.4.4.1. One consequence of this was that the predicted air temperatures were also lower, as discussed in section 10.3.4.4.

Chapter 11: Simulation Test Analysis

This chapter compares and contrasts the refrigeration system simulation methodology of the RefSim environment described in this thesis with three other methodologies represented by the ISIM simulation of Darrow *et al* (1991), the RADS simulation systems (versions 2.2 and 3.1) of Cleland (1985*b*) and the Flexible Modelling Environment (FME) of James (1988). Comparisons with the FME are less detailed because no simulations were carried out with that environment. Comparisons are made in four areas:

- Model expression.
- Simulation results.
- Simulation flexibility.
- Ease of model testing.

11.1 Methods for comparing simulation programs

A quantitative and fair comparison between simulation programs was found to be difficult to achieve. The quantities of data required to simulate the large plants of sections 10.3 and 10.4 were considerable (about 200 kB per simulation) and the different simulation environments required those data to be in significantly different formats. For the comparison to be fair it was essential that the simulated plants not only resemble the real plant as closely as possible, but that the simulation data sets resemble each other very closely. It was important that differences in simulation outputs should result from differences between the simulation programs rather than from differences between the plant representations.

The method used for sections 10.3 and 10.4 was to use the RADS V3.1 simulation as a base case. From this base, the RADS V2.2 simulation was automatically generated by the use of the two utility programs *Enr3-2* (which converted RADS V3.1 engine room description files into V2.2 format) and *App3-2*

(which performed the same function for application files). These programs were written specifically for this task and are not described further. Similarly, RADS V3.1 application files were converted into RefSim input files by the *App2Ref* utility described in section 8.4.1. RADS V3.1 engine room files were converted into RefSim input by hand.

The assumption that the refrigeration plants operated on 24 or 48 hour cycles was fundamental to the operation of the RADS programs. Although RefSim was not bound to cyclical operation, one disadvantage of automatically converting the simulation data from the RADS V3.1 format into the RefSim format was that the conversion perpetuated this assumption. This had the same detrimental effect on the ability of RefSim to accurately simulate parts of the plant which did not operate on strict cycles as it did on the performance of the RADS simulations under the same circumstances.

The fairness of comparisons between simulation methodologies also depended upon the cases chosen for testing. Models and simulation environments are not generally developed to cover the whole range of cases which may be simulated and if a model is applied outside its intended range of operation, it cannot be expected to produce good results. Under those circumstances, the best that may be hoped is that the simulation environment will warn the user of the difficulty and then attempt to produce what results it can. This was done to some degree by each of the simulation environments considered in this comparison and a methodology should not be judged harshly if it is applied outside its declared range of accuracy.

Finally, in some cases, it was not possible to measure the system input data required to drive the simulations to the desired level of accuracy. Where input data were estimated, the overall quality of the simulation results depends as much upon the quality of the input data estimate as upon the quality of the simulation environment or its component models. The problems encountered with the RADS Ocean Beach cooling floor simulations (discussed in section 10.3.4.6) exemplified this difficulty.

11.2 Model expression

When a model is developed for research purposes, it is essential that the model be easy to understand, develop and enhance. A modelling methodology should facilitate the development of models with these attributes.

Models are comprised of two parts. Firstly, a model is defined by the mathematical expressions which relate its inputs to its outputs (the algorithm). Secondly, a general model is made specific to a particular problem by the definition of system input variable values and initial conditions (the data). The ways in which these two parts are expressed have an important bearing upon the clarity of the overall model.

In the RefSim and FME environments, the algorithmic part of the model was described in a structured programming language (RefSim in TopSpeed Modula-2, FME in Vax Pascal). The system input and initial condition parameters for a RefSim simulation were held in a separate simulation input file. By separating these two parts of the model description, the part of the model common to all model instances was kept clearly separate and unchanged while the differing instances were defined in the simulation input file. In contrast, the ISIM input file combined both the input data and algorithmic model description. The ISIM approach facilitated very simple simulations while making complex simulations confusing and difficult to understand. This factor alone precluded the use of ISIM for simulations which were much more complex than the water chiller of section 10.2.

The RADS programs produced two sorts of data files for each model description — a "results" file for the user and a data file which was used by the simulation environment as input — so a direct comparison of the RADS data files with the RefSim and ISIM files was not possible. The RADS algorithmic model descriptions were largely separate from their data (one exception was the condenser heat capacity discussed in section 10.2.6) but they were written in relatively unstructured FORTRAN-77 (ANSI, 1978), with some models intertwined with each other in the source code. This made it difficult to isolate individual models.

11.3 Simulation results comparison

RefSim performed comparably to RADS V3.1 programs under most circumstances. This was expected due to the similarities between many of the models used in these two environments. For batch freezers, RADS V3.1 and RefSim performed better than RADS V2.2 due to their use of the improved product heat load model described in Chapter 4. RefSim performed better than RADS V3.1 due to its more sophisticated treatment of the product sub-cooling stage. During application start-up, RefSim performed better than either RADS V3.1 or RADS V2.2 due to its ODE integration step size control methods (which were more sophisticated than those of the RADS programs) and its handling of state events such as controller mode switching.

RefSim performed significantly worse than the RADS programs where room dynamics were dominated by door openings. The RADS V3.1 and V2.2 door models included implicit heat load smoothing which made the RADS room temperature behaviour less sensitive to any inadequacy in the stirred-tank model for air temperature. The solution to this problem in RefSim would be to use a group of models to simulate a buffered door as described in section 9.3. For rooms where the air volume was well stirred, RefSim performed almost as well as the RADS programs in the presence of door heat loads.

11.4 Simulation flexibility

Several problems were encountered with both RADS versions because they had limited flexibility when modelling plant components or their interactions. The evaporator and condenser heat capacity problems identified in section 10.2 for both RADS versions, and the difficulties experienced with RADS V2.2 in modelling the Ocean Beach engine room control strategy (section 10.3.1) and the Horotiu beef chiller operating mode (section 10.4.1) demonstrated this lack of flexibility. Both RADS versions performed well when faced with simulation problems for which they

were designed (i.e. most of sections 10.3 and 10.4), but performed less well when dealing with problems outside their design parameters.

The FME was more flexible than the RADS environments. It allowed the user considerable freedom in describing the plant and all of the model parameters could be readily altered by the user. The FME was, however, designed for simulating less complex plants than those of sections 10.3 and 10.4. For complex plants, the descriptive freedom available to the user may have made the plant description prohibitively complex.

RefSim was found to be more flexible than RADS at two levels. Firstly, all of the parameters of the RefSim models were provided in the input file. While each RefSim model provided default values for most parameters, the parameter values may all be set as the user wishes. This resolved the problem encountered with both RADS versions in section 10.2 where the condenser heat capacity could not be altered by the user. In addition, while most RefSim models carried out some range-checking on input parameters, they did not reject parameter values unless the use of those values would put the model outside its applicable range. The RefSim models did not attempt to make judgements upon the appropriateness of parameter values beyond this range-checking function. For this reason, RefSim was more suitable for experienced users than for naive users.

Secondly, where a component could not be simulated by an existing RefSim model, or by linking a group of general RefSim models together, a new RefSim model could be designed, written and included in the environment with no disruption to the existing models and no interference with the underlying simulation utilities. A new model need not duplicate any of the simulation utilities, and if it is similar to some existing model the new model could also use the inheritance feature of the object-oriented implementation language to access the common model functionality without code duplication.

This second aspect of RefSim's flexibility was used four times (*GenCondenser*, *GenEvaporator*, *FluidTank* and *RADSBrCooler*) in chapter 10 to quickly add new models to the RefSim environment when it was discovered that they

were required. When they were carefully designed, even models which were added ad hoc could be sufficiently flexible to be re-used for quite a different purpose. This was shown by the way in which the *GenEvaporator* and *FluidTank* models developed for the water chiller simulation were re-used in the AFFCO Horotiu simulation (section 10.4.3) with no source code changes — only changes to input parameters.

11.5 Ease of model testing

The RefSim, FME and ISIM environments provided significant benefits over the RADS environments when individual models were being tested. In ISIM, all of the variables used in any model were available for output to the user. In RefSim and the FME, this was true for almost all variables. RefSim also provided the *Environment* model types which were used to set up simulated "test-rigs" in which other models could be run with well-defined inputs independent of any other active models. It was therefore not necessary to set up a complex simulation with a large number of other models in order to test one of them. The FME could have carried out "test-rig" simulations if it was provided with *Environment*-type models.

In contrast, the RADS environments provided limited facilities for model testing. The user could simulate one or more applications without the refrigeration system by assuming that there was an infinite refrigeration capacity available at a declared evaporation temperature. It was more difficult to test individual models. For example, to test an alternative product model in RADS, it would have been necessary to simulate a room and evaporator, and perhaps a temperature controller if the test conditions required a constant ambient temperature. RADS V2.2 also required that at least one wall be simulated for each room. This would have made individual model testing substantially more complicated than was the case in RefSim, FME or ISIM.

11.6 Conclusions

RefSim has met its objectives of being a flexible environment which is general enough to simulate a wide range of refrigeration systems. When simulating industrial refrigeration systems, RefSim produced simulation results as accurate as the best alternative environment, RADS V3.1 (with which it shares many models) and superior to RADS V2.2. RefSim was found to be considerably more flexible in several respects than either of the RADS versions, and comparable in flexibility to the FME of James (1988) and ISIM (Hay and Crosbie, 1984). At the same time, RefSim could deal with plants which were more complex than those which could be handled by either the FME or ISIM.

Work in two areas would be required to improve the accuracy with which RefSim predicts measured data. Firstly, simulation input data designed specifically for RefSim (rather than for comparing several simulation programs) would not necessarily be cyclical, and so the input data would represent the plant more accurately. Secondly, although enhancing the product heat load model produced significantly better correspondences with measured data, some difficulties remain with the following models:

- Product heat load chilling stage. Some of the assumptions made when developing the chilling stage of the product heat load model reduced the accuracy of product heat load prediction early in the chilling process. In addition, difficulties in obtaining an accurate E factor for chilling may cause problems for this part of the model when dealing with particular shapes.
- Room air mixing. For rooms where there was little air movement, the assumption that the air behaves like a well-stirred tank was poor. This caused difficulties with the effect of door loads on the air temperature.
- Room structures. The assumption that room surfaces with significant heat capacities could be treated as being wholly inside the room was poor

in some cases. For these cases, a *Wall* model which combined a conductivity component with a heat capacity might be more appropriate.

Chapter 12: Conclusions

12.1 Product heat load model

A purpose-designed ordinary differential equation model was developed for predicting the heat load vs. time profile of chilling and freezing food product.

For regular geometric shapes, and for a wide range of practical ambient conditions, the heat load predicted by the ODE model corresponded well with that predicted by finite difference methods during most of the freezing process. Some discrepancies were noted during the sub-cooling stage, but these were not of practical importance as the total heat load at that time was small. The heat load was generally better predicted during the freezing stage than during chilling.

A differential thermocouple method was shown to be superior to alternative methods for measuring the heat load profiles of freezing of meat carton shapes, lamb and sheep carcasses. Heat balances achieved using this method were within the estimated range of experimental error (±10%) for high (1.4 ms^{-1}) air velocity runs, although the heat balances for low (0.35 ms^{-1}) velocity runs were less satisfactory.

The heat load predicted for lamb carcasses by the ODE model corresponded to the measured loads within the range of experimental error. For the meat carton shape, it was necessary to configure both the FD and ODE methods to take position-variation of the heat transfer coefficient into account. Once this was done, the measured heat load was predicted within the accuracy of the measurements. This also demonstrated that the ODE method could be used in some complex heat transfer situations through the use of simple extensions.

12.2 Simulation environment

An object-oriented methodology was developed to map refrigeration system components onto mathematical models and to unite these models into simulations of

complete refrigeration plants. This methodology was successfully used to develop an environment for simulating complete industrial refrigeration systems.

The principle of encapsulation allowed the implementation of flexible models with well-defined boundaries — each as an object within the environment. The principle of inheritance reduced the extent of code replication when compared with previous simulation environments and encouraged the identification of common attributes which are shared between models. Polymorphism encouraged the identification of functionality which is common to many models.

New models were easily implemented and included in the simulation environment without making changes to any other model. The concept of testing models by setting them up in simulated "test-rigs" was found to be useful and this was facilitated by the methodology.

By having general models which were stripped down to their essential features, it was possible to model complex situations by using the flexible model linking protocols to combine existing general models rather than by implementing new specific models for the purpose.

12.3 Simulation testing

When tested against data measured on working refrigeration plants, RefSim was found to produce simulation results superior to those of RADS V2.2 and comparable to those of RADS V3.1 and the custom-built ISIM simulation. In addition, RefSim was found to be more flexible than either RADS or ISIM when modifying simulation input data, using existing models in unexpected ways, and adding new models to the simulation environment.

In simulations of the two large meat plants that were surveyed during the project, some temperatures were predicted poorly. This was partly due to deficiencies in the input data provided to the simulations. In particular, the simulation results indicated that the thermal conductivities of some coldstore walls were higher than the accepted values for their insulating materials by a large but

uncertain factor. The information available on door opening frequencies and durations was also inadequate for some doors.

The new ODE product heat load model was found to be of real benefit in characterizing the behaviour of blast freezers. The temperature profiles of these applications were predicted much better with the new heat load model when compared with the older model used by RADS V2.2. The model was less successful when simulating chilling product. This may have been improved if the E shape factor for chilling was known more accurately.

The improvements to the *Room* model made in RefSim were found to be of little benefit despite its apparently more realistic treatment of the application humidity. The RefSim *Door* model was found to predict door behaviour less well than the buffered door used in RADS V3.1. This suggested that the assumption of a well-mixed room was not accurate for some applications rather than indicating a deficiency in the *Door* model.

Overall, the disagreements between measurements and predictions were more attributable to data uncertainties than model weaknesses. Some improvements could nevertheless be made in the models of chilling food product, structural material heating and cooling, and air mixing in rooms.

Notation

a	A constant in equation (4-25)
A	Area / m^2
A	A constant in equation (9-18)
A_s	Surface area / m^2
A_x	Cross-sectional area of the body at the distance x from the centre / m^2
AC&A	Accelerated Conditioning and Aging
AFFCO	The Auckland Farmer's Freezing Co-operative Ltd.
APPLICS	A refrigeration application analysis and steady state simulation program. Part of the RADS package (Cleland, 1985*b*)
AWK	A string processing programming language (Aho *et al*, 1988)
b	A constant in equation (4-25)
B	A constant in equation (9-18)
Bi	The Biot number ($Bi = h\ X\ /\ k$) (dimensionless)
c	A constant in equation (4-25)
C	A volumetric specific heat capacity / J m^{-3} K^{-1}
C_{air}	The volumetric specific heat capacity of air / J m^{-3} K^{-1} (or, in section 9.2, the mass-basis specific heat capacity of air / J kg^{-1} K^{-1})
C_{liq}	The mass-basis specific heat capacity of refrigerant liquid (section 9.2) or fluid in a *FluidTank* (section 10.2.3.1) / J kg^{-1} K^{-1}
C_{vap}	The mass-basis specific heat capacity of refrigerant vapour / J kg^{-1} K^{-1} (section 9.2)
C_{water}	The mass-basis specific heat capacity of liquid water / J kg^{-1} K^{-1} (section 9.2)
C_l	Unfrozen volumetric specific heat capacity / J m^{-3} K^{-1}
C_s	Frozen volumetric specific heat capacity / J m^{-3} K^{-1}
DYSAN	A dynamic refrigeration system simulation environment (Nowotny, 1983)
e	Error factor (section 8.4.4.2)

E	Equivalent Heat Transfer Dimensionality (section 4.3.1)
E_{ref}	E for a reference shape (section 4.3.1)
F	Total volumetric flow rate of air through a room / $m^3\ s^{-1}$
FD	Finite Difference method for solving systems of PDEs
FE	Finite Element method for solving systems of PDEs
FME	Flexible Modelling Environment (James, 1988)
Fo	The Fourier number (dimensionless)
FORTRAN	FORmula TRANslation programming language (ANSI, 1978)
$FracOK$	Desired fraction of successful substeps in each ODE solver major step.
h	Surface heat transfer coefficient / $W\ m^{-2}\ K^{-1}$
h	ODE solver step size (section 8.4.4)
h	Absolute mass-basis humidity (section 9.2) / $kg\ kg^{-1}$
h_0	Desired ODE solver step size (dimensionless)
h_1	ODE solver step size during the last substep (dimensionless)
h_1, h_2	Absolute mass-basis saturation humidity at either end of a ventilation duct (section 9.5) / $kg\ kg^{-1}$
h_{sat}	Absolute mass-basis saturation humidity (section 9.2) / $kg\ kg^{-1}$
h_{temp}	Temporary ODE solver step size (section 8.4.4.2)
H	Total enthalpy / J
$\hat{H}$	Volumetric specific enthalpy / $J\ m^{-3}$
$\hat{H}_1$, $\hat{H}_2$	Volumetric specific enthalpy of air at either end of a ventilation duct (section 9.5) / $J\ m^{-3}$
$\hat{H}_{base}$	Volumetric specific enthalpy at T_{base} / $J\ m^{-3}$
$\hat{H}_f$	Volumetric specific enthalpy at T_f when the material is unfrozen / $J\ m^{-3}$
$\hat{H}_{fg}$	Difference in mass-basis specific enthalpy between water vapour and liquid (or solid if $T < 0°C$) (section 9.2) / $J\ kg^{-1}$
$\hat{H}_{vap}$, $\hat{H}_{liq}$	Mass-basis specific enthalpy of refrigerant vapour and liquid respectively in a refrigerant vessel (section 9.10) / $J\ kg^{-1}$
H_{Vessel}	Total enthalpy in a refrigerant vessel / J

I	Accumulated error integral (section 8.4.4.2)
ISIM	A specialised dynamic simulation programming language (Hay and Crosbie, 1984)
k	Thermal conductivity of a body / W m^{-1} K^{-1}
k_l	Unfrozen thermal conductivity of a body / W m^{-1} K^{-1}
k_s	Frozen thermal conductivity of a body / W m^{-1} K^{-1}
k_Y	A mass transfer coefficient / kg m^{-2} s^{-1}
K_{Cl}	A constant (Cleland, 1986*b*)
K_I	Integration gain (section 8.4.4.2)
K_p	Proportional gain (section 8.4.4.2)
L	Latent heat of freezing / J m^{-3}
LF74	A state-space and heat balance network modelling environment (Glockner and Findeisen, 1984)
LMTD	Logarithmic mean temperature difference / °C
m_{air}	Mass flow rate of humid air into a room / kg s^{-1}
M	Mass / kg
M_{air}	Mass of humid air in a room / kg
$M_{dry\ air}$	Mass of dry air in a room / kg
M_{liq}	Mass of fluid in a *FluidTank* / kg (section 10.2.3.1)
M_{vap}, M_{liq}	Mass of refrigerant vapour and liquid respectively in a refrigerant vessel (section 9.10) / kg
M_{vapour}	Mass of water vapour in a room / kg
MassHeatCap	Thermal mass of a device / J K^{-1}
MIRINZ	Meat Industry Research Institute of New Zealand (Inc.)
Modula-2	A general purpose programming language (Wirth, 1982)
MS-DOS	Microsoft Disk Operating System (Microsoft, 1991)
n	A shape factor (section 4.5)
N	A shape factor (section 4.6)
Nsteps	Number of successful substeps in this major ODE solver step.

ODE	Ordinary Differential Equation
p	The number of skip list nodes with levels higher than a given level as a fraction of the number of nodes on the that level (Pugh, 1990) (section 8.4.7)
P	Proportional (with reference to process controllers)
P	Proportional factor (section 8.4.4.2)
P_{Cl}	A constant (Cleland, 1986*b*)
P_{Vessel}	Pressure in a refrigerant vessel / Pa
Pascal	A general purpose programming language (Jensen and Wirth, 1985)
PDE	Partial Differental Equation
PI	Proportional-Integral (with reference to process controllers)
PID	Proportional-Integral-Derivative (with reference to process controllers)
PROSIM	A steady state heat pump simulation environment (Thorbergsen, 1985)
Q_{air}	Volumetric flow rate of air / $m^3\ s^{-1}$
R_{Cl}	A constant (Cleland, 1986*b*)
R_v	A vapour diffusion resistance (section 9.7) / $m^2\ s\ kg^{-1}$
RADS	Refrigeration Analysis, Design and Simulation software package (Cleland, 1985*b*)
RH	Relative humidity / fractional
RKF	Runge-Kutta-Fehlberg, a method for the numerical solution of ordinary differential equations (Fehlberg, 1969)
S	ODE solver step size adjustment safety factor.
S_0	ODE solver step size adjustment safety factor for the next set of steps.
S_1	ODE solver step size adjustment safety factor during the last set of steps.
SIMUL	A dynamic refrigeration system simulation environment. Part of the RADS package (Cleland, 1985*b*)
Smalltalk	A general purpose object-oriented programming language and environment (Lalonde and Pugh, 1990)

STASAN	A steady state refrigeration system simulation environment (Nowotny, 1983)
t	Time / s
t_{Close}	Time at which a door will close (section 9.4) / s
$t_{Current}$	Current time (section 9.4) / s
t_f	Freezing time / s
t_{mb}	Mean time between door openings / s
t_{mo}	Mean length of door open periods / s
t_{Open}	Next door opening time / s
T	Temperature / °C
T_1, T_2	Temperatures on either side of a wall (section 9.6) / °C
T_a	Ambient temperature / °C
T_{base}	Base temperature for enthalpy calculation / °C
T_{evap}	Evaporation temperature / °C
T_f	Freezing temperature / °C
T_{ff}	Pure material freezing temperature of the component which actually freezes within a body (e g. water for most food products) / °C
T_i	Initial temperature / °C
T_{liq}	Fluid temperature in a *FluidTank* (section 10.2.3.1) / °C
T_{ma}	Mass average temperature / °C
T_s	Surface temperature / °C
T_{Vessel}	Temperature in a refrigerant vessel / °C
TEV	Thermostatic Expansion Valve
TotFail	Number of failed substeps in this ODE solver major step
TRNSYS	A modular dynamic simulation environment (Bullock *et al*, 1983)
U	Overall heat transfer coefficient / W m^{-2} K^{-1}
UA	The product (overall heat tranfer coefficient) (heat transfer area) / W K^{-1}
V	Volume of a body / m^{-3}

V_f	Unfrozen volume of a body / m^3
V_{vap}, V_{liq}	Volume of refrigerant vapour and liquid respectively in a refrigerant vessel (section 9.10) / m^3
w_c	The mass flow rate of water vapour into the humid air of a room from condensing or evaporating water / kg s^{-1}
w_{liq}	The mass flow rate of water liquid into a room / kg s^{-1}
w_{model}	The mass flow rate of water vapour into the humid air of a room from the specified model / kg s^{-1}
w_{Prod}	The net mass flow rate of water vapour into the humid air of a room from a product item / kg s^{-1}
w_{Room}	The net mass flow rate of water vapour into the humid air of a room / kg s^{-1}
w_{Vent}	The net mass flow rate of water vapour through a ventilation duct / kg s^{-1}
W	Mass of condensed water in a room / kg
x	A position / m
x_f	Freezing front position / m
x_{mCl}	"Mean radius" of Cleland (1986*b*) / m
X	The critical depth of a body to its thermal centre / m
Y_{ma}	Mass average fraction unaccomplished temperature change

Greek symbols

α	Thermal diffusivity / m^2 s^{-1}
β_i	The *i*th root of: $\beta \cot \beta + (Bi - 1) = 0$ (section 4.5)
Δ_0	Desired ODE solver substep tolerance (dimensionless)
Δ_1	ODE solver substep tolerance during the last substep (dimensionless)
ΔH_{10}	The change in enthalpy between the freezing temperature, T_f, and -10°C / J m^{-3}

ΔT_{air}	The change in temperature of the air passing over the product / °C
Θ_{max}	Maximum allowable ODE solver step size increase factor (section 8.4.4.2)
ρ_{air}	Density of air / kg m^{-3}
ρ_{vap}, ρ_{liq}	Density of refrigerant vapour and liquid respectively in a refrigerant vessel (section 9.10) / kg m^{-3}
τ	Time constant of a room (section 9.2) / s^{-1}
ϕ	Energy flow / W
ϕ_c	Energy flow from changes to the water mass in a room (due to water condensing or evaporating) (section 9.2) / W
ϕ_{chill}	Energy flow during chilling / W
ϕ_{cond}	Energy flow out of a condenser into the cooling fluid / W
ϕ_e	Electrical energy flow / W
ϕ_{ess}	Electrical energy flow during steady state operation / W
ϕ_{evap}	Energy flow into an evaporator from the application / W
ϕ_{freeze}	Energy flow during freezing / W
ϕ_{in}	Energy flow into an evaporator from the refrigerant inlet (section 10.2.3.2) / W
ϕ_{model}	Energy flow from the specified model into a room (section 9.2) / W
ϕ_{out}	Energy flow out of an evaporator through the refrigerant outlet (section 10.2.3.2) / W
ϕ_p	Energy flow from the product / W
ϕ_{resp}	Energy flow into the humid air in a room due to respiring product / W
ϕ_{Room}	Net energy flow into the humid air in a room / W
$\phi_{subcool}$	Energy flow during subcooling / W
ϕ_{Vent}	Energy flow through a ventilation duct / W
ϕ_{Vessel}	Net energy flow into a refrigerant vessel / W
ϕ_{Wall}	Energy flow through a wall / W
Q	Rejection margin (section 8.4.4.2)

References

Aho, A. V., Kernighan, B. W., Weinberger, P. J. (1988). *The AWK Programming Language*, Addison-Wesley, Reading, Massachusetts

ANSI (1978) *American National Programming Language FORTRAN*, American National Standards Institute, New York

ANSI/IEEE (1985) *IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Standard 754-1985*, The Institute of Electrical and Electronic Engineers Inc., New York.

Beckey, T. J. (1986). Modelling and verification of a vapour compression heat pump, *IIR Commission B1, B2, E1, E2, Purdue*, **1986-1**, 175-183

Belth, M. I., Tree, D. R. (1986). Design and preliminary analysis for measuring transient mass rate of flow in unitary heat pumps, *ASHRAE Transactions*, **92, 1B**, 843-853

Blankespoor, H. J., Brok, S. W., Touber, S. (1976). An interactive compressor simulator, in *Simulation of Systems*, 8th AICA Congress, Delft, The Netherlands, 897-906

Bonte, A., Veldhoven, B. v. (1983). Dynamic behaviour of a refrigerating plant, *16th International Congress of Refrigeration*, II, 845-851

Bullock, C. E., Wroblewski, D. E., Groff, G. C. (1983). A dynamic simulation model for residential air-to-water heat pump systems, *16th International Congress of Refrigeration*, V, 337-345

Cellier, F. E. (1979). Combined continuous/discrete system simulation languages — Usefulness, experiences and future development, in *Methodology in Systems Modelling and Simulation*, Zeigler *et al* (eds), North-Holland, Amsterdam, 201-220

Chan, C. Y., Haselden, G. G. (1981*a*). Computer-based refrigerant thermodynamic properties — Part 1, *International Journal of Refrigeration*, **4**, 7-12

Chan, C. Y., Haselden, G. G. (1981*b*). Computer-based refrigerant thermodynamic properties — Part 2, *International Journal of Refrigeration*, **4**, 52-60

Chan, C. Y., Haselden, G. G. (1981*c*). Computer-based refrigerant thermodynamic properties — Part 3, *International Journal of Refrigeration*, **4**, 131-134

Cheney, W., Kincaid, D. (1985). *Numerical Mathematics and Computing, Second Edition*, Brooks/Cole, Monterey, California

Chi, J., Didion, D. (1982). A simulation model of the transient performance of a heat pump, *International Journal of Refrigeration*, **5**, 176-184

Cleland, A. C. (1983). Simulation of industrial refrigeration plants under variable load conditions, *International Journal of Refrigeration*, **6**, 11-19

Cleland, A. C. (1985*a*). Experimental verification of a mathematical model for simulation of industrial refrigeration plants, *International Journal of Refrigeration*, 8, 275-282

Cleland, A. C. (1985*b*). RADS - a computer package for refrigeration analysis, design and simulation, *International Journal of Refrigeration*, **8**, 372-372

Cleland, A. C. (1986*a*). Computer subroutines for rapid evaluation of refrigerant thermodynamic properties, *International Journal of Refrigeration*, 9, 346-351

Cleland, A. C. (1986*b*). Use of dynamic simulation in the design of food freezing equipment, in *Food Engineering and Process Applications, Volume 2*, Le Maguer and Jelen (eds), Elsevier Applied Science Publishers, London, 55-65

Cleland, A. C. (1990). *Food Refrigeration Processes - Analysis, Design and Simulation*, Elsevier Science Publishers, London.

Cleland, A. C., Earle, R. L. (1976). A new method for prediction of surface heat transfer coefficients in freezing, *Refrigeration Science and Technology*, **1**, 361-368

Cleland, A. C., Earle, R. L. (1982*a*). A simple method for prediction of heating and cooling rates in solids of various shapes, *International Journal of Refrigeration*, **5**, 98-106

Cleland, A. C., Earle, R. L. (1982*b*). Freezing time prediction for foods — a simplified procedure, *International Journal of Refrigeration*, **5**, 134-140

Cornelius, M. (1991). *Refrigeration Analysis, Design and Simulation Package: "RADS" — Notes for users (revised), Release 3.1*, Food Technology Research Centre, Massey University, Palmerston North, New Zealand

Cox, B. J. (1986). *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley, Reading, Massachusetts

Crosbie, R. E. (1984). Continuous and discontinuous-change models: concepts for simulation languages, in *Simulation and Model-based Methodologies: an integrative view*, Oren *et al* (eds), Springer-Verlag, 337-356

Darrow, J. B. (1990). Development of a dynamic simulation model for a refrigeration system, *Bachelor of Technology Final Year Research Project Report*, Department of Biotechnology, Massey University, Palmerston North, New Zealand

Darrow, J. B., Lovatt, S. J., Cleland, A. C. (1991). Assessment of a simple mathematical model for predicting the transient behaviour of a refrigeration system, *18th International Congress of Refrigeration*, (in press) (included in this thesis as Appendix 4.3)

Edwards, B. F. (1976). Evaluation of a commercial finned evaporator, MIRINZ 493, Meat Industry Research Institute of New Zealand.

Eldredge, D. L., McGregor, J. D., Summers, M. K. (1990). Applying the object-oriented paradigm to discrete event simulations using the C++ language, *Simulation*, 50, 83-91

Ellis, M., Stroustrup, B. (1990). *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, Massachusetts

Elzas, M. S. (1979). What is needed for robust simulation?, in *Methodology in Systems Modelling and Simulation*, Zeigler *et al* (eds), North-Holland, Amsterdam, 57-91

Elzas, M. S. (1984). System paradigms as reality mappings, in *Simulation and Model-based Methodologies: an integrative view*, Oren *et al* (eds), Springer-Verlag, 41-67

Fehlberg, E. (1969). Klassiche Runge-Kutta formeln funfter und siebenter ordnung mit schrittweitenkontrolle, *Computing*, **4**, 93-106

Findlay, I. C., Smith, J. (1967). Response of a single/two pass liquid-liquid heat exchanger to disturbances in flow rate, *Journal of Mechanical Engineering Science*, **9**, 3, 211-217

Fleming, A. K. (1969). Calorimetric properties of lamb and other meats, *Journal of Food Technology*, **4**, 199-215

Fleming, A. K. (1976). Refrigeration demands for meat processing, *Refrigeration Science and Technology*, **1**, 285-292

Fleming, A. K., Earle, R. L. (1967). Cooling and freezing of lamb and mutton carcasses — 2. Weight loss during cooling, *Food Technology*, **22**, 100-104

van Gerwen, R. J. M. (1988). Modelling temperature distribution in products moving in cold chains, *IIR Commission B1, B2, C2, D1, D2, D3, Wageningen*

Glockner, G., Findeisen, F. (1984). The computing program LF74: A software solution for typical simulation problems of air and refrigeration engineering, *IIR Commission B2, Dresden*, 1984-2, 231-236

Gunther, E., Kinne, L., Najork, H. (1984). Dynamic simulation of the start behaviour of hermetic refrigerant compressors operating in refrigerant plant, *IIR Commission B2, Dresden*, 1984-2, 177-185

Gustafsson, K., Lundh, M. Soderlind, G. (1988). A P.I. stepsize control for the numerical solution of ordinary differential equations, *BIT — Computer Science and Numerical Mathematics*, 28, 2, 270-287

Hay, J. L., Crosbie, R. E. (1984). ISIM — A simulation language for microprocessors, *Simulation*, 43, 133-136

Hasegawa, Y. (1965). Analogue computer solution of passenger air car air conditioning process, *SHASE Transactions, Japan*, **2**, 1-6

Higuchi, K., Hayano, M. (1982). Dynamic characteristics of thermostatic expansion valves, *International Journal of Refrigeration*, **5**, 216-220

Hossain, Md. M., Cleland, D. J., Cleland, A. C. (1992*a*). Prediction of freezing and thawing times for foods of two-dimensional irregular shape using a semi-analytical geometric factor, *International Journal of Refrigeration* (in press)

Hossain, Md. M., Cleland, D. J., Cleland, A. C. (1992*b*). Prediction of freezing and thawing times for foods of three-dimensional irregular shape using a semi-analytical geometric factor, *International Journal of Refrigeration* (in press)

Hrones, J. A. (1942). The analysis of a continuous process by a discontinuous step method, *Transactions of the American Society of Mechanical Engineers*, November 1942, 753-757

James, K. A. (1988). *Dynamic mathematical modelling of refrigeration systems and heat pumps*, Ph.D. thesis, Institute of Environmental Engineering, South Bank Polytechnic, London.

James, K. A., Dunn, A., James, R. W. (1987*a*). A data structure for flexibly linking models of refrigerant components, *Institute of Environmental Engineering Research Memorandum No. 104*, South Bank Polytechnic, London.

James, K. A., James, R. W. (1986*a*). Dynamic analysis of a heat pump using established modelling techniques, *Institute of Environmental Engineering Technical Memorandum No. 98*, Polytechnic of the South Bank, London.

James, K. A., James, R. W. (1986*b*). Transient analysis of thermostatic expansion valves, *Institute of Environmental Engineering Technical Memorandum No. 99*, Polytechnic of the South Bank, London.

James, K. A., James, R. W., Dunn, A. (1987*b*). Development of a thermodynamically-based mathematical model of a novel heat pump, *17th International Congress of Refrigeration*, B, 629-634

James, K. A., James, R. W., Dunn, A. (1986*c*). A critical survey of dynamic mathematical models of refrigeration systems and heat pumps and their components, *Institute of Environmental Engineering Technical Memorandum No. 97*, Polytechnic of the South Bank, London.

James, K. A., Wong, A. K. H., James, R. W. (1986*d*). Pressure, flow and temperature transients in refrigeration systems, *International Journal of Refrigeration*, **9**, 200-205

James, S. J., Bailey, C. (1981). Measurement of product heat load during cooling, freezing and thawing of meat product, *Proceedings of the Institute of Refrigeration*, 78, 33-41

Jensen, K., Wirth, N. (1985). *Pascal: user manual and report*, Springer-Verlag, New York

Jowitt, R. (ed) (1983). *Physical Properties of Food*, Applied Science Publishers, London,

JPI (1991). *TopSpeed Modula-2 Reference (Version 3.01)*, Jensen and Partners International, 1101 San Antonio Road, Suite 301, Mountain View, California

Kreutzer, W. (1986). *System simulation: programming style and languages*, Addision-Wesley, Wokingham, England.

Lalonde, W. F., Pugh, J. R. (1990). *Inside Smalltalk: Volume I*, Prentice Hall, Englewood Cliffs, New Jersey

Latimer, D. M., Marsden, R. (1983). Microprocessor control of refrigeration compressors, *Procedings of the Institute of Refrigeration*, 4, 1-7

Loeffen, M. P. F., Earle, R. L., Cleland, A. C. (1981). Two simple methods for predicting food freezing times with time-variable boundary conditions, *Journal of Food Science*, 46, 1032-1034

Loeffen, M. P. F., Carrie, A. (1986). Computerized freezer control to protect meat tenderness, *Transactions of the Institution of Professional Engineers, New Zealand*, **13**, 2/EMCh, 113-118

Lovatt, S. J., Pham, Q. T., Cleland, A. C., Loeffen, M. P. F. (1991). Prediction of product heat release as a function of time in food cooling - Part 1: Theoretical considerations, *Journal of Food Engineering*, (in press) (included in this thesis as Appendix 4.1)

Lovatt, S. J., Pham, Q. T., Loeffen, M. P. F., Cleland, A. C. (1991). Prediction of product heat release as a function of time in food cooling - Part 2: Experimental Testing, *Journal of Food Engineering*, (in press) (included in this thesis as Appendix 4.2)

MacArthur, J. W. (1984). Analytical representation of the transient energy interactions in vapour compression heat pumps, *ASHRAE Transactions*, **90, 1B**, 982-996

McNabb, A., Wake, G. C., Hossain, Md. M. (1990*a*). Transition times between steady states for heat conduction. Part I - General theory and some exact results. *Occasional Papers in Mathematics and Statistics*, No. 20, Massey University, New Zealand

McNabb, A., Wake, G. C., Hossain, Md. M., Lambourne, R. D. (1990*b*). Transition times between steady states for heat conduction. Part II - Approximate solutions and examples. *Occasional Papers in Mathematics and Statistics*, No. 21, Massey University, New Zealand

Marshall, S. A., James, R. W. (1975). Dynamic analysis of an industrial refrigeration system to investigate capacity control, *Proceedings of the Institution of Mechanical Engineers*, 189, 44/75, 437-445

Meffert, H. F. Th., van Beek, G. (1988). Basic elements of a physical model for refrigerated vehicles, II — Temperature distribution, *IIR Commission B, C2, D, Wageningen*

Mellor, J. D. (1983). Critical evaluation of thermophysical properties of foodstuffs and outline of future developments, in *Physical Properties of Food*, Jowitt (ed), Applied Science Publishers, London, 331-352

Miles, C. A., van Beek, G., Veerkamp, C. H. (1983). Calculation of thermophysical properties of foods, in *Physical Properties of Food*, Jowitt (ed), Applied Science Publishers, London, 269-313

Microsoft (1991). *Microsoft MS-DOS Operating System version 5.0 User's Guide and Reference*, Microsoft Corporation, One Microsoft Way, Redmond, West Virginia

Moreno, J. (1987). Air interchange rate through a cold store open door, *17th International Congress of Refrigeration*, D, 138-143

Murphy, W. E., Goldschmidt, V. W. (1984). Measurement of transient temperatures downstream of the evaporator in an insulated duct, *ASHRAE Transactions*, 90, 1A, 245-261

Nowotny, S. (1983). Long-term experiences with the application of mathematical models to analyze the steady state and dynamic behaviour of refrigerating systems and components — general aspects to establish a catalogue of computer programs, *16th International Congress of Refrigeration*, II, 835-844

Oren, T. I. (1979). Concepts for advanced computer assisted modelling, in *Methodology in Systems Modelling and Simulation*, North-Holland, Amsterdam, 29-55

Oren, T. I., Zeigler, B. P., Elzas, M. S. (eds) (1984). *Simulation and Model-Based Methodologies: An Integrative View*, Springer-Verlag, Berlin

Pala, M., Devres, Y. D. (1988). Computer simulated model for power consumption and weight loss in a cold store, *IIR Commission B, C2, D, Wageningen*

Perry, R. H., Green, D. (1984). *Perry's Chemical Engineers' Handbook*, 6th Edition, McGraw-Hill, Singapore

Pappas, C. H., Murray, W. H. (1988). *80386 Microprocessor Handbook*, Osborne McGraw-Hill, Berkeley, California

Pflug, I J, Blaisdel, J L, Kopelman, J (1965). Developing temperature-time curves for objects that can be approximated by a sphere, infinite plate, or infinite cylinder, *ASHRAE Transactions*, 71, 1, 238-248

Pham, Q. T. (1984). Extension to Plank's equation for predicting freezing times of foodstuffs of simple shapes, *International Journal of Refrigeration*, 7, 377-383

Pham, Q. T. (1985). Analytical method for predicting freezing times of rectangular blocks of foodstuffs, *International Journal of Refrigeration*, **8**, 43-47

Pham, Q. T. (1986*a*). Freezing of foodstuffs with variations in environmental conditions, *International Journal of Refrigeration*, **9**, 290-295

Pham, Q. T. (1986*b*). Simplified equation for predicting the freezing time of foodstuffs, *Journal of Food Technology*, **21**, 209-219

Pham, Q. T. (1987*a*). Calculation of bound water in frozen food, *Journal of Food Science*, **52**, 1, 210-212

Pham, Q. T. (1987*b*). A converging-front model for the asymmetric freezing of slab-shaped food, *Journal of Food Science*, **52**, 10, 795-800

Pham, Q. T. (1989). Approximate formulae for heat load during freezing and thawing, Personal Communication, 16 August 1989 (included in this thesis as Appendix 1)

Pham, Q. T. (1990). Effect of Biot number and freezing rate on the accuracy of some food freezing time prediction methods, *Journal of Food Science*, **55**, 1429-1434

Pham, Q. T. (1991). Temperature vs. enthalpy data for Tylose MH1000, Personal Communication, February 1991 (included in this thesis as Table 6-2)

Pham, Q. T., Oliver, D. W. (1983) Infiltration of air into cold stores, *16th International Congress of Refrigeration*, 4, 67-72

Pham, Q. T., Willix, J. (1984). A model for food dessication in frozen storage, *Journal of Food Science*, 49, 1275-1281,1294

Pham, Q. T., Willix, J. (1987). Heat transfer coefficients in the air blast-freezing of rows of cartons, *17th International Congress of Refrigeration*, C, 350-357

Plank, R. (1913). Die Gefrierdauer von Eisblocken, *Zeitschrift fur die gesamte Kalte-Industrie*, 20 (6), 109-114

Plank, R. (1941). Beitrage zur Berechnung und Bewertung der Gefriergeschwindikeit von Lebensmitteln, *Beiheft zur Zeitschrift fur die gesamte Kalte-Industrie Reihe 3*, Heft 10, 1-16

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vettering, W. T. (1986). *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, England

Pugh, W. (1990). Skip lists: a probabilistic alternative to balanced trees, *Communications of the ACM*, 33, 6, 668-676

Rajendran, N., Pate, M. B. (1986). A computer model of the startup transients in a vapor-compression refrigeration system, *IIR Commissions B1, B2, E1, E2, Purdue*, 1986-1

Reid, D. S. (1983). Fundamental physicochemical aspects of freezing, *Food Technology*, April 1983, 110-113,115

Reid, R. C., Prausnitz, J. M., Poling, B. E. (1987). *The Properties of Gases and Liquids, Fourth Edition*, McGraw-Hill, New York

Riedel, L. (1960). Eine Prufsubstanz Fur Gefrierversuche, *Kaltetechnik* 12, 8, 222-225

Reynoso, R. O., De Michelis, A. (1988). Simulation of cryogenic batch freezers, *International Journal of Refrigeration*, 11, 6-10

Sbrana, F., Rossi, M. (1988). Low energy consumption cold stores for frozen foods: optimising of the refrigerating cycle, *IIR Commissions B, C2, D, Wageningen*

Schwartzberg, H. G. (1976). Effective heat capacities for the freezing and thawing of food, *Journal of Food Science*, 46, 153

Sokulski, M. B. (1972). Sharp freezer room calculation, *Refrigeration Science and Technology*, 2, 343-353

Szczechowiak, E., Rainczak, J. (1987). The cooling chamber and the air cooler operation in unsteady states, *17th International Congress of Refrigeration*, B, 864-869

Tamm, W. (1965). Air flow within air curtains to protect cold rooms, *11th International Congress of Refrigeration*, 2, 1025-1033

Thorbergsen, E. (1985). PROSIM, A program system simulating heat pump plants, *IIR Commission B2, Trondheim*, 1985-3

Touber, S. (1984). Principles and methods for mathematical modelling the steady state and dynamic behaviour of refrigeration components and installations, *IIR Commission B2, Dresden*, 1984-2

Treybal, R. E. (1984). *Mass Transfer Operations: Third Edition*, McGraw-Hill, Tokyo

Vidmar, V., Gaspersic, B. (1987). Dynamic behavior of draft condenser, *17th International Congress of Refrigeration*, E, 534-539

Wade, N. L. (1984). Estimation of the refrigeration capacity required to cool horticultural produce, *International Journal of Refrigeration*, 7, 358-366

Wang, H., Touber, S. (1987). The prediction of air flow patterns in cold stores based on temperature measurements, *17th International Congress of Refrigeration*, D, 52-60

Wang, H. W., Touber, S. (1988). Simple non-steady state modelling of a refrigerated room accounting for air flow and temperature distributions, *IIR Commission B, C2, D, Wageningen*

Wang, H., Touber, S. (1990). Distributed dynamic modelling of a refrigerated room, *International Journal of Refrigeration*, 13, 214-222

Wang, H., Touber, S. (1991). Distributed and non-steady-state modelling of an air cooler, *International Journal of Refrigeration*, 14, 98-111

Wong, A. K. H., James, K. A., James, R. W. (1985). The influence of transients in refrigeration systems, *Institute of Environmental Engineering Technical Memorandum No. 98*, Polytechnic of the South Bank, London.

Wong, A. K. H., James, R. W. (1988). Control strategies of an intelligent controller for a liquid chilling plant, *IIR Commission B, C2, D, Wageningen*

Wong, A. K. H., James, R. W. (1988). A simple dynamic liquid chilling plant model, *Institute of Environmental Engineering Technical Memorandum No. 108*, South Bank Polytechnic, London.

Welty, J. R. (1978). *Engineering heat transfer — SI. Version*, Wiley, New York

Wirth, N. (1982). *Programming in Modula-2*, Springer-Verlag, Berlin

Xiao Dagang, Yu Yongzhang (1987). A new mathematical model of screw refrigeration system with an economizer, *17th International Congress of Refrigeration*, B, 739-744

Yasuda, H., Touber, S., Machielsen, C. H. M. (1983). Simulation model of a vapour compression refrigeration system, *ASHRAE Transactions*, **89, 2A**, 408-425

Zeigler, B. P. (1979). What is modelling and simulation methodology?, in *Methodology in Systems Modelling and Simulation*, North-Holland, Amsterdam, xi-xv

Zeigler, B. P., Elzas, M. S., Klir, G. J., Oren, T. I. (eds) (1979). *Methodology in Systems Modelling and Simulation*, North-Holland, Amsterdam

Appendix 1: Personal communication from Pham (1989): "Approximate formulae for heat load during freezing and thawing"

This Appendix reproduces a personal communication to the author (Pham, 1989) which described a food product freezing heat load formula similar to that developed independently by the author at the same time (described in section 4.6 of this thesis).

APPROXIMATE FORMULAE FOR HEAT LOAD DURING FREEZING AND THAWING

Q.T.Pham

IERS. COMM AUG 16 1989

ABSTRACT

In the design of freezing equipment the two parameters of utmost interest are the freezing time and the heat load. The freezing time is controlled by the slowest-freezing part of the product while the peak heat load is controlled by the fastest-freezing part. A two-parameter equation is proposed for use in the simulation of freezing processes, to calculate both freezing time and heat load curve.

NOMENCLATURE

A	Area, m^2
Bi	Biot number, 2hX/k
h	heat transfer coefficient, W/m^2K
G	unfrozen fraction
k	thermal conductivity, W/m K
K	k(T_a-T_f)/(#LX^2)
n	freezing time parameter (eq.1)
N	Heat load parameter (eq.2)
T	temperature, ^C
T_a, T_c, T_f, T_i	Environment, final center, freezing point and initial object temperatures, ^C
V	Volume, m^3
x	coordinate along smallest dimension, m
x_f	x-coordinate of freezing front, m
X	smallest half-dimension, m
#a1	ratio of smallest to next smallest dimension
#a2	ratio of smallest to largest dimension
#b1	1/#a1
#b2	1/#a2
#L	latent heat per unit volume, J/m^3

INTRODUCTION

Process time and product heat load must be calculated during the design

or simulation of freezing and thawing equipment. To avoid the numerical solution of partial differential equations, which can be time consuming for multidimensional shapes, it is desirable to develop approximate ordinary differential equations (Cleland, RADS).

The calculation of freezing and thawing times can often be accurately calculated by approximate arithmetic formulae that may take into account the shape of the product. The same formulae, suitably rearranged, are then used for heat load calculations.

On reflection, this practice may lead to significant errors. Consider, for example, an elongated shape such as a rectangle with height 0.16 m and width 0.56 m. Such a shape can, for the purpose of freezing time calculation, be replaced by a slab. This is because freezing time is governed by what happens in the middle section of the rectangle while the "ends" have little influence. On the other hand, the initial or peak heat load is mainly dependent on what happens at the "ends", which freeze quickly, rather than at the middle. As a general rule:

- Freezing time is influenced by the SLOWEST freezing parts

- Peak heat load is influenced by the QUICKEST freezing parts.

Thus the heat load curves for the slab and rectangle above show quite different features although their freezing times are within 2-3% of each other. The peak heat loads, averaged over half-hour intervals, differ by 25% and the final heat loads by 60%. (Differences are more marked still if instantaneous heat loads are taken.) The above example is not an hypothetical situation: it shows what happens when rows of cartons, 0.16 m thick x 56 m wide x 2 m long, are frozen in a typical air blast freezer.

In this paper, general expressions for the heat load curve of products undergoing freezing or thawing will be suggested.

Table 1. Heat load curves for slab, rectangle and brick. Htc = 15 W/m2^C. See also chart HEAT.PLT.

Time	Heat load, W		
hr	Slab	Rectangle	Brick
0.25	5.18	6.50	6.73
1.75	4.66	5.89	6.30
2.75	4.46	5.58	5.94
3.75	4.23	5.22	5.60
4.75	4.06	4.90	5.25
5.75	3.90	4.59	4.88
6.75	3.76	4.30	4.57
7.75	3.64	4.10	4.33
8.75	3.54	3.90	4.11
9.75	3.43	3.68	3.87
10.75	3.34	3.45	3.62
11.75	3.25	3.27	3.38
12.75	3.17	3.07	3.16
13.75	3.11	2.86	2.88
14.75	3.03	2.63	2.59
15.75	2.96	2.39	2.29

17.75	2.61	1.75	1.61
18.75	2.24	1.38	1.23

THEORY

The following assumptions will be made:

1. Sensible heat effects are negligible. This is the quasi-steady state assumption, since in the absence of sensible heat the term dT/dt in the heat balance equation disappears.

2. A fixed thermal centre exists.

3. The temperature profile along the shortest dimension through the thermal centre, between the surface and the freezing front, is given by

$$dT/dx = A\, x^{(-n)} \qquad (1)$$

The parameter n indicates the curvature of the temperature profile caused by geometry. For slabs n = 0 and the temperature profile is linear. For infinite cylinder n = 1 and for sphere n = 2, but for other shapes n can take any other value. It will ber assumed however that n remains constant throughout the phase-change period, or at least that some time-averaged value can be defined.

3. The volume of the unfrozen fraction (in a freezing process) is related to the relative distance from the thermal centre to the freezing front x_f as a power law,

$$G = (x_f/X)^N \qquad (2)$$

The parameter N gives the dimensionality of the object. For example, N = 1 for slabs, 2 for infinite cylinders and 3 for spheres. Again, N can take on any other value for other shapes if the freezing front does not remain similar in shape to the surface, though it will be assumed that N is constant throughout the freezing period.

4. Newton's cooling law applies at the product's surface:

$$k(dT/dx)_{x=X} = h(T_a - T_s) \qquad (3)$$

By integrating eq.(1), the temperature profile along x is given by

$$\frac{T - T_s}{T_f - T_s} = \frac{x^{(1-n)} - X^{(1-n)}}{x_f^{(1-n)} - X^{(1-n)}} \qquad (4)$$

The freezing front speed is

$$\frac{dx_f}{dt} = \frac{-k(dT/dx)_{x=x_f}}{\#L}$$

$$= \frac{k\,(n-1)\,(T_s - T_f)/\#L}{x_f - X^{(1-n)}\, x_f^{n}} \qquad (5)$$

The temperature gradient at the surface is

$$(dT/dx)_{(x=X)} = \frac{(T_f-T_s)(1-n)}{X} \cdot \frac{1}{G^{[(1-n)/N]} - 1} \qquad (6)$$

From eqs.(3) and (6):

$$T_f-T_s = \frac{(G^{[(1-n)/N]} - 1)\ (T_f-T_a)}{2(n-1)/Bi + G^{[(1-n)/N]} - 1} \qquad (7)$$

Substituting for (T_f-T_s) into eq.(5) gives, after simplification:

$$\frac{dx_f}{dt} = \frac{-(n-1)Nk(T_f-T_a)/\#L}{x_f^{(2-N)} + [2(n-1)/Bi-1]\ x_f^{(1+n-N)}} \qquad (8)$$

and from eq.(2)

$$\frac{dG}{dt} = \frac{-(n-1)NK}{G^{(2/N-1)} + [2(n-1)/Bi-1]\ G^{[(1+n-N)/N]}} \qquad (9)$$

where

$$K = \frac{k(T_f-T_a)}{\#LX^2} \qquad (10)$$

Integration gives

$$t = \frac{(n+1)(G^{2/N} - 1) + [4(n-1)/Bi-2]\ [G^{(n+1)/N} - 1]}{2(1-n^2)K} \qquad (11)$$

Note the particular case n = 1 for which eqs.(9) and (11) become

$$dG/dt = \frac{N K G^{1-2/N}}{\ln G - 2N/Bi} \qquad (12)$$

$$t = \frac{G^{2/N} \ln G}{2KN} + \frac{(1-G^{2/N})\ (1+4/Bi)}{4K} \qquad (13)$$

The total freezing time t_f is found by putting G = 0 in eq.(11):

$$t_f = \frac{1 + 4/Bi}{2(1+n)K} \qquad (14)$$

Thus t_f is independent of N, but the heat load $\#L.dG/dt$ depends on N. Note that the shape factor EHTD (ratio of slab and object freezing times) is simply 1 + n.

PRACTICAL APPLICATIONS

Equations (9) and (11) can be used to calculate the freezing rate and heat load during the simulation of freezing processes, such as done by RADS. The following problems have to be solved:

1. In practice materials may not have a sharp freezing point, and sensible heat effects are usually not negligible.

Pham (1986?) lumped sensible heat (appropriately weighted) with latent heat, and defined a "mean freezing point": these measures enable ideal-case freezing equations to be applied to practical cases.

Note: for heat load calculations these measures may not be enough to account for the initial cooling peak! What has Simon done on this? My 1986 eqn may need further mods?
Maybe N can be adjusted to fit data?

2.How to find n

Since the total freezing time depends only on n but not on N, n can be found by comparing the freezing time of the object with that of a slab, infinite cylinder, sphere, or any other shape for which n is already known or can be calculated.
For ellipses and ellipsoids n can be found from (Pham, 1989)

$$n = \left(\frac{F - 1}{\#a1 + \#a2}\right)^p \left(\#a1^q + \#a2^q\right)$$

where

$$p = \frac{1}{1 + Bi}$$

$$q = \frac{1 + Bi/2}{1 + Bi/4}$$

For #b1, #b2 =< 2, eq.(?) simplifies to

$$n = \#a1^q + \#a2^q$$

For rectangles and bricks n can be found from

$$n = r\,(\#a1^s + \#a2^s)$$

where

$$r = (1 + 0.35\,Bi) / (1 + 0.55\,Bi)$$

$$s = (1 + 2.00\,Bi) / (1 + 0.70\,Bi)$$

Alternatively use any existing EHTD equation since n = EHTD-1.
(see Don Cleland's thesis p.181...)

(Equation above gives +-6% error compared with Don Cleland's numerical results (see program EHTDBRICK.FOR);
Cleland & Earle (1982) eqn gives -20 to +8% error;

Donald's eqn gives +-5% error but is VERY complicated.
EHTD depends on Bi, T_i, T_c, T_a anyway and even numerical values can vary by 10% for the same shape.)

l and Bi w.

3. How to find N

Except for the basic shapes, N is likely to vary during the process rather than staying constant. However, if we are primarily interested in the PEAK heat load which occurs right at the beginning, then a suitable value for N can be found as follows.

At time 0, there is no internal resistance to heat transfer and the heat load is controlled by the surface resistance:

$$\left(\frac{dG}{dt}\right)_{G=1} = -\frac{hA(T_f-T_a)}{V\rho L}$$

$$= -K\,Bi\,AX/(2V)$$

From eq.(9), putting G = 1:

$$\left(\frac{dG}{dt}\right)_{G=1} = -K\,Bi\,N/2$$

Equating eqs.(?) and (?):

$$N = AX/V$$

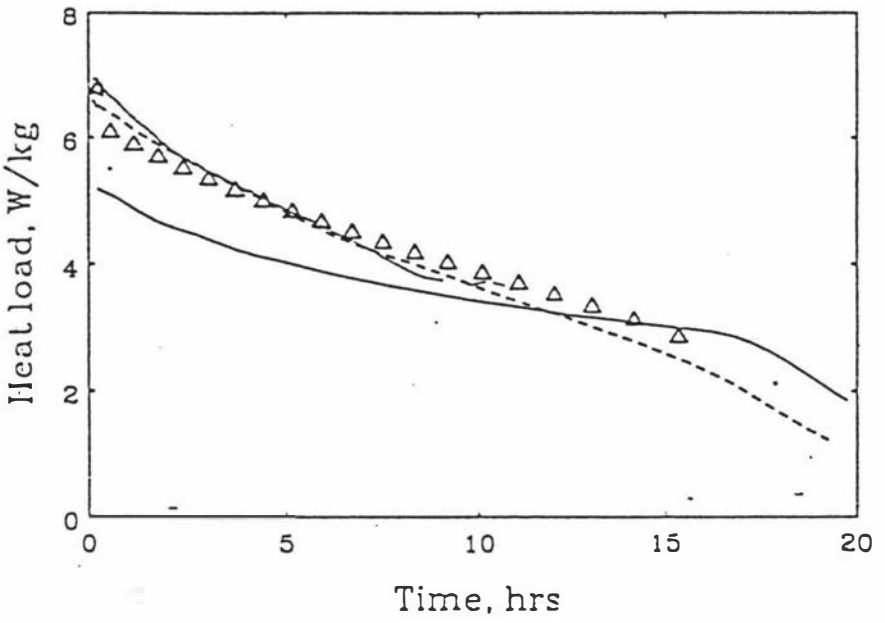
For example, in the case of the rectangle in the introduction, AX/V = 1.29. Thus N = 1.29 and n = 0 reproduce realistically the heat load curve. (Table below; also, graph [TUAN.FREEZE]HEATLOAD.PLT.)

Table ?. Heat load curve from eqs.(11) and (12) for cartons of Tylose, with N = 1.29, n = 0.

Time hr	Heat load W/kg
0.57	6.08
1.16	5.88
1.77	5.69
2.40	5.50
3.06	5.32
3.73	5.15
4.43	4.98
5.16	4.81
5.90	4.65
6.68	4.48
7.48	4.32
8.32	4.16
9.18	4.00
10.08	3.84
11.02	3.68
12.01	3.50
13.05	3.32
14.15	3.11
15.34	2.83
16.72	1.56

This approach is very simple but needs to be checked.
Also, it needs to be use judiciously. Very thin parts of the object
(such as the shanks of a carcass)
will freeze very quickly, causing a sharp peak at the start,
but will not affect the overall heat load curve to any extent.
Thus, these parts should be ignored in the calculation of AX/V
unless it is crucial that the sharp peak heat load be catered for,
which is unlikely.

WORK PLAN: Calculate heat load from complex shapes by f.e. methods,
or measure, and correlate by eqs above.


Heat load, W/kg
0
2
4
6
8
0
5
10
15
20
Time, hrs

Appendix 2: Extended Backus-Naur Form definition of the RefSim input language

This appendix defines the syntax of the RefSim input language in terms of the Extended Backus-Naur Form (EBNF) of Jensen and Wirth (1985, Appendix D). The EBNF notation is described first (following Jensen and Wirth, 1985, Appendix D), followed by the RefSim input language description using that notation.

A2.1 Extended Backus-Naur Form

The EBNF notation is commonly used to define the grammar of computer programming languages. EBNF consists of a collection of rules and symbols which show how to form any of the sentences which may be written in that language. Each rule consists of a non-terminal symbol (or "meta-identifier") and an EBNF expression which defines that symbol. EBNF is context-free, so context-dependent features of the language are defined more loosely by supplementary information in section A2.2.2.

The meta-symbols from which an EBNF expression may be constructed are as follows:

Meta-symbol	*Meaning*
=	is defined to be
|	alternatively
.	end of rule
[*X*]	0 or 1 instance of *X*
{*X*}	0 or more instances of *X*
(*X Y*)	a grouping which consists of *X* followed by *Y*
"XYZ"	the terminal (literal) symbol XYZ
MetaIdentifier	the non-terminal symbol *MetaIdentifier*

A2.2 RefSim input language definition

A2.2.1 Context-free input language grammar

SimulationInput	=	*HeaderSection* {*ModelDefinition*} *TrailerSection*
HeaderSection	=	"SIMULATION" *SimName* {*RunLength* | *Output* | *Report*}
TrailerSection	=	"END" *SimName*
SimName	=	*Identifier*
RunLength	=	"Run_Length" "=" *Number*
Output	=	"Output_every" *Number* ["to" *Filename*]
Filename	=	*Identifier*
Report	=	"Report" ("loud" | "quiet" | "silent")
ModelDefinition	=	*ModelHeader* {*ModelStatement*} *ModelTrailer*
ModelHeader	=	"MODEL" *ModelName*
ModelTrailer	=	"END" *ModelName*
ModelStatement	=	(*RealAssignment* | *StringAssignment* | *TableAssignment* | *OutputList* | *LinkList*)
RealAssignment	=	*VariableName* "=" *Number*
StringAssignment	=	*VariableName* "=" *Identifier*
TableAssignment	=	"TABLE" *VariableName* "[" {*Identifier* | *Number*} "]"
OutputList	=	"<" {*VariableName*} ">"
LinkList	=	"[" {*Identifier*} "]"
VariableName	=	*Identifier*
Identifier	=	*Character* {*Character*}
Character	=	any ASCII character but *WhiteSpace*
WhiteSpace	=	*WhiteSpaceChar* {*WhiteSpaceChar*}
WhiteSpaceChar	=	*Punctuation* | "{" | "}"
Punctuation	=	LF | FF | CR | HT | " " | "," | ";"

Number	=	[*Sign*] *UnsignedInteger* ["." *UnsignedInteger*] ["E" [*Sign*] *UnsignedInteger*]
UnsignedInteger	=	*Digit* {*Digit*}
Sign	=	"+" | "-"
Digit	=	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
Comment	=	"{" {*Character* | *Punctuation* | *Comment*} "}"

A2.2.2 Context-dependent input-language definition information

1 All terminal syntactic items must be separated by *WhiteSpace*.

2 Any *WhiteSpace* may optionally include *Comment* statements which are stripped by the lexical analyzer before parsing (and are therefore not explicitly mentioned in the definition of *WhiteSpace*).

3 In *HeaderSection* and *TrailerSection*, *SimName* must be the same identifier.

4 In *ModelHeader* and *ModelTrailer*, *ModelName* must be the same identifier.

5 In *OutputList*, each of the *Identifier* non-terminals must be the name of a variable which is valid for the current model type.

6 In *LinkList*, each of the *Identifier* non-terminals must be the name of a model instance which is defined somewhere in the input file.

7 In each of *RealAssignment*, *TableAssignment*, and *StringAssignment*, the *VariableName* must be one of the input variables of that type which is valid for input to the current model type.

8 There is an implementation limit of 16 characters on the length of each *Identifier*.

Violation of condition 1 will typically cause the lexical analyser to produce an error report indicating that condition 8 has been violated. Violation of conditions 3, 4, 5 and 7 will result in a parser error while reading the affected model. Violation of condition 6 will produce an error during model linking.

Appendix 3: Definition of the RefSim output format

While the RefSim input language could be largely described using the Extended Backus-Naur Form, this is not the case for the output. The overall format of the output table is context-dependent and therefore it may not be adequately described in EBNF. The following description is therefore less formal than that of section A2.2.1.

A3.1 Output format description

The output from RefSim is in the form of a table in MS-DOS text format (i.e. only the ASCII characters in the range 32-126, with each line terminated by the character pair CR LF (character 13 followed by character 10)). The first line of the output file is of the form:

FirstLine = "SIMULATION : " *SimName*

where *SimName* corresponds to that of section A2.2.

This line is followed by one blank line, and then by a series of column headings left-justified in 16 character fields with each field separated from the next by one space (" ") character. Each heading is in three lines. The first line shows the model name from which the output value is obtained (e.g. "Room1"). The second line shows the name of the output variable (e.g. "Temperature"). The third line shows the units in which the output value is expressed (e.g. "deg.C").

The column headings are followed by one blank line and then for each column heading there is a column of Real numbers indicating the value of that variable at each output time. These numbers are right-justified in 16 character fields with each field separated from the next by one space (" ") character. There is one line of numbers in the table for each output time, starting from time 0. Lastly, there

is one line showing the value of each variable at the end of the simulation. This last line may duplicate the second to last time if the output interval evenly divides the simulation length.

The first column in any simulation output is always the system time with units of seconds and this need not (and cannot) be specified by the user.

The names included in each field of the column headings follow the convention that they should not include any spaces. This convention together with the overall output format allows the output table to be processed easily with field-oriented text processing programs. During the project, simulation output files were processed using a set of UNIX-like utility programs ported to MS-DOS such as AWK, tail, sed, cut, paste and sh. Graphs were plotted using the program gnuplot. All of these programs are widely available and freely distributable.

Typical output files for large simulations reached about 500 fields wide.

Appendix 4: Work published

This Appendix reproduces manuscripts which are currently in press and describe aspects of the work reported in this thesis.

A4.1 A New Method of Predicting the Time-Variability of Product Heat Load During Food Cooling — Part 1: Theoretical considerations

This manuscript is referenced in the thesis text as Lovatt *et al* (1992*a*). References in this manuscript to Hossain *et al* (1991*a*,*b*) correspond to the references Hossain *et al* (1992*a*,*b*) in the thesis text.

A New Method of Predicting the Time-Variability of Product Heat Load During Food Cooling - Part 1: Theoretical Considerations

S J Lovatt, Q T Pham, A C Cleland, M P F Loeffen
Massey University, Palmerston North, New Zealand, and Meat Industry Research Institute of New Zealand, Hamilton, New Zealand.

ABSTRACT

A new method has been developed to predict the way in which the heat load placed upon a refrigeration system by cooling food product varies with time. This information is important to refrigeration system designers and plant operators as capital and running costs may depend strongly upon the heat load sustained by the system. The new method utilizes ordinary differential equation models of both chilling and phase change processes and requires only one parameter in addition to those required by existing freezing and chilling time prediction methods. The new method has been tested against finite difference (FD) calculations under a wide range of conditions for those product shapes to which FD methods may be applied. It was found to predict the product heat load to within 10% of the FD estimate for all test cases, except at the very start and at the end of the cooling process. The method requires much fewer computational resources than FD, and is capable of extension to shapes not easily handled by FD methods.

NOTATION

A_s	Surface area of body (m²)
A_x	Heat conduction area at distance x from thermal centre (m²)
a,b,c	Empirical coefficients in the freezing product enthalpy function
Bi	Biot number
C	Constant of Integration
C_p	Product specific heat capacity (J/m³)

C_u	Unfrozen product specific heat capacity (J/m³)
C_f	Frozen product specific heat capacity (J/m³)
E	Equivalent Heat Transfer Dimensionality
E_{ref}	E for reference shape in the method of Cleland (1986)
FD	Finite Difference
FE	Finite Element
Fo	Fourier number
H	Enthalpy (J/m³)
$H_{T_{ref}}$	Product enthalpy at T_{ref} (J/m³)
H_f	Product enthalpy at T_f (J/m³)
h	Surface Heat Transfer Coefficient (W/(m² K))
K_α	Constant in Cleland's (1987) object volume ODE
k	Thermal conductivity (W/(m K))
k_u	Unfrozen product thermal conductivity (W/(m K))
k_f	Frozen product thermal conductivity (W/(m K))
L	Latent heat of freezing (for food products, the precise definition may vary, depending on circumstances) (J/m³)
N	Heat release geometry factor
n	Freezing time geometry factor (= E - 1)
ODE	Ordinary Differential Equation
P_α, R_α	Constants in Cleland's (1987) freezing front ODE.
T	Temperature (°C)
T_a	Ambient temperature (°C)
T_{ref}	Temperature at which the product enthalpy is fixed at 0 J/m³ (°C)
T_f	Freezing temperature (°C)
T_i	Initial temperature (°C)
T_m	Mass average temperature of body (°C)
T_s	Surface temperature of body (°C)
t	Time from start of process (s)
V	Volume of body (m³)
V_f	Unfrozen volume of body (i.e. inside x_f) (m³)
X	Critical dimension of body (m)

x	Distance from thermal centre of body (m)
x_f	Distance of freezing front from thermal centre (m)
x_{ma}	"mean radius" in Cleland's (1987) freezing front ODE (m)
Y_m	Mass average fractional unaccomplished temperature change
α	Thermal diffusivity (m^2/s)
β	Chilling boundary equation root
ΔH_{10}	Enthalpy change between T_f and -10°C (J/m^3)
ϕ	Product heat load (W)

INTRODUCTION

Refrigeration plants in the New Zealand meat industry are characterised by their large size (1 - 6 MW), complexity, and highly variable heat load (Fleming, 1976). The control of refrigeration costs is a major concern for the industry, so there is a strong motivation towards developing ways of predicting heat load variations over time and the response of the refrigeration system to those load variations. If accurate predictions can be made, a more precise design of the plant itself and of the control strategy to be employed can enable significant cost savings to be made.

The advantages to be gained can best be realized by developing models for inclusion in dynamic plant simulations (Cleland, 1986). Simulation models allow the engineer to test refrigeration and control system designs without the need to build the plant first, and without relying upon subjective experience. The whole area of food refrigeration process simulation has recently been reviewed by Cleland (1990).

Product cooling makes the greatest contribution to both the mean and peak heat loads. The total amount of heat to be removed from the product during cooling is well known for most products, being the difference in enthalpy between the starting and ending temperatures. One would expect more heat to be released during the early part of the process than at the end due to the higher temperature driving force but detailed information is difficult to obtain. In the absence of more

information on the heat load profile over the process, designers must make assumptions and work from those, to the possible detriment of their designs.

Food product cooling can be considered to consist of three stages. First, there is a "chilling", or pre-cooling, stage during which the product cools from its initial temperature, releasing sensible heat. This stage comes to an end and "phase change" starts when the surface temperature drops below the initial freezing temperature and ice formation commences, although there is no sharp transition between the stages. At the end of the chilling stage, the mass average temperature is still above the freezing temperature, and the remaining sensible heat is removed in the phase change period. The rate of cooling can be characterized by the Biot number, *Bi*:

$$Bi = \frac{h\,X}{k} \qquad (1)$$

At slow rates of cooling ($Bi \to 0$), the difference between the mass average and surface temperatures is small, so that little sensible heat remains at the end of chilling. In contrast, at high rates of cooling ($Bi \to \infty$), the surface is much colder than the mass average, with the result that the chilling stage is short, and ends with much of the material still significantly superheated.

Once the phase change period commences, the dominant influence on heat load is the latent heat of freezing. Initially, freezing commences at the surface, and a distinct (but not sharp) "freezing front" moves into the object. For this front to pass through a region, both the latent heat of freezing and the residual superheat must be removed, so it is common to treat the residual superheat and latent heat as a lump sum.

Even after the freezing front has passed, the material is not completely frozen and has not yet cooled to the ambient temperature. Further ice formation and sensible cooling will occur as the region progressively drops in temperature. Hence a third stage, that of "sub-cooling", can be distinguished. Sub-cooling commences once the

further cooling of the region through which the phase change front has passed dominates the heat release from the phase change front surrounding the shrinking unfrozen region. Again, the transition is not clear-cut. When $Bi \to 0$, the surface temperature remains higher than when $Bi \to \infty$, so in the former case, the transition occurs when the unfrozen region is very small, while for the latter there may be a significant fraction of the whole object still completely unfrozen when the transition occurs.

The three stages of the process have quite different heat load vs. time profiles, so to be able to predict heat load accurately for the whole process, a model needs to take the characteristics of each into account.
Of the various available methods that estimate the cooling heat load profile, finite difference (FD) and finite element (FE) methods have the strongest theoretical foundation, are known to predict freezing times within the margin of error for experimental measurements, and may be expected to predict heat loads with similar precision. Unfortunately, they have significant disadvantages which make them inappropriate for many heat load estimation tasks:

1. FD may be conveniently used for only a small range of regular shapes. FE can deal with a wider range of shapes, but is still limited to shapes that can be described using only a few parameters.
2. It can be time-consuming to prepare an FD calculation, and more so for an FE calculation.
3. FD calculation is computationally intensive and FE is even more so. In the authors' experience, both FD and FE require too many computer resources for these methods to be used as part of a complete refrigeration system model on a personal computer.

An alternative group of methods includes those which utilize a small number of ordinary differential equations (ODEs) to describe the problem. These may sometimes be integrated analytically if simplifying assumptions can be made, but are still much quicker to prepare and run than FE or FD methods when integrated

numerically. Empirical factors are easily included when required. ODE methods were therefore the focus of this study.

Early work in predicting the heat load profile during freezing was carried out by Sokulski (1972), who based a meat freezing model on the work of Plank (1941). An empirical shape factor was used to extend the model to irregular shapes, but the main emphasis of the model was upon freezing time and it was not tested for heat load prediction accuracy. The first detailed dynamic food refrigeration system simulations were carried out by Marshall and James (1975). They modelled a continuous vegetable freezing tunnel by using a lumped parameter system with an apparent product specific heat capacity which varied through the process to account for latent heat. The product heat load was described by:

$$\phi = V C_p \frac{dT_\sim}{dt} = hA(T_a - T_\sim) \qquad (2)$$

The apparent volumetric specific heat capacity of the product, C_p, at any given time, was estimated from the amount of time that a product item had been in the freezer rather than from the amount the product was frozen, or its temperature. The model was accurate as long as the refrigeration plant was working correctly, but was difficult to extend to other situations. Further, equation (2) implicitly requires that $T_s = T_\sim$ which is true only if $Bi = 0$. Models such as this which do not take the internal temperature profile into account may be expected to be accurate only under low Biot number conditions (typically less than 5% error below $Bi = 0.1$, according to Welty (1978)). This approach is therefore inaccurate for most food freezing situations (where the Biot number is more commonly between 1 and 10) unless h is adjusted somehow to reflect the internal temperature profile.

Reynoso and De Michelis (1988) used a similar approach, but more correctly made the apparent specific heat capacity a function of temperature and position within the product, with the consequent drawback that they had to use finite difference methods to solve the equation.

Seeking a model applicable to a wide range of *Bi*, but wishing to avoid the computational complexity and limited applicability of FD, Cleland (1986) proposed a modified form of Plank's (1941) ordinary differential equation (ODE) model. Cleland's method simplified the prediction problem by separating it into (a) a technique of predicting the depth of the freezing front, and (b) a technique of estimating the volume of product enclosed by that freezing front; that is:

$$\frac{dx_f}{dt} = \frac{(T_f - T_\infty)\,E}{\Delta H_{1a}\,E_{-4}\left[\frac{2P_a}{h}\left(\frac{x_f}{X}\right)^{\gamma_a - 1} + \frac{8R_a x_f}{k_s}\left(\frac{x_f}{x_{-a}}\right)^{\gamma_a - 1}\right]} \qquad (3)$$

$$\frac{dV}{dx} = K_a E x^{\xi - 1} \qquad (4)$$

(and hence: $V = K_a x^{\xi}$)

$$\phi = \frac{dV}{dx}\frac{dx_f}{dt}\left(\Delta H_{1a} + C_f(T_i - T_f)\right) \qquad (5)$$

The Equivalent Heat Transfer Dimensionality (E) shape factor was used to deal with irregular shapes. Empirically, E may be viewed as being defined by equation (6).

$$t_{f\,a-b} = \frac{t_{f\,slab}}{E} \qquad (6)$$

That is, a body with a given E value will freeze E times faster than an infinite slab. Until the work of McNabb *et al* (1990*a*,*b*), equation (6) and correlations derived by combining it with experimental data were the only available methods of determining E for complex shapes, but it may now be determined from the physical shape of the product as described by Hossain *et al* (1991*a*,*b*). While E is largely dependent upon the geometry of the freezing body, it is also a weak function of the Biot number.

Cleland's method was found to be accurate for some simple shapes during all but the early and latter stages of the freezing process with a Biot number between 0.1

and 10. This corresponded with the assumptions made when the model was developed — specifically, the lumping of sensible heat with latent heat. While this was a significant improvement over the earlier models, it had the disadvantage that it relied upon an appropriate choice of reference shape, to which the predicted heat load could be very sensitive. Further, it predicted only the phase change heat load. A more general method would predict chilling and sub-cooling heat loads as well.

Chilling heat load models have been developed by Cleland (1983), Cleland (1985), Wade (1984), and Szczechowiak and Rainczak (1987). All were of the same form as equation (2), except that the heat capacity was constant with product temperature. The model of Cleland (1985) was more sophisticated than the others in that it used the half-life method of Cleland and Earle (1982) to correct the h values for the implied $T_s = T_\infty$ assumption when $Bi > 0$. The method was found to be satisfactory for most food product chilling applications and a wide range of product shapes, but it was not directly compatible with any of the freezing models.

There have been no previous models specifically for sub-cooling of frozen product, although Cleland (1985) included a model identical to that for chilling which ignored the possibility of residual latent heat.

In the present work, a new method was sought to predict food product heat release over the whole cooling process. Its desired attributes were as follows:

1. The method had to be accurate over most (preferably all) of the process.
2. The method had to be accurate over a broad range of Biot numbers.
3. The method had to be extensible to all shapes of food product (including complex shapes such as lamb and beef carcasses).
4. The method had to be general enough to handle both chilling and freezing processes.

5 Ideally, the method should have a sound theoretical basis, and require a minimum of empirically-derived parameters.

6 The method had to be convenient to implement and practical to compute on a personal computer.

THEORETICAL DEVELOPMENT

Given the discussion above, an ODE model appeared to hold the most promise. For the chilling and sub-cooling periods, equation (2) was considered a reasonable beginning, but the phase change period required a different approach. The idea of tracing the freezing front movement first proposed by Plank (1941), and the relationship between that and the product heat release rate developed by Cleland (1986) provided a starting point. If the resulting ODEs were integrated numerically, then time-variable conditions (for instance, surface heat transfer coefficient and ambient temperature) could be taken into account after the fashion of Loeffen *et al* (1981).

A heat release profile calculation should commence with the chilling model, transitioning to the phase change, and thence to the sub-cooling model when appropriate. It will be seen that correct selection of the transition criteria is important.

While the method should be able to handle time-variable conditions, it would be advantageous if, in the special case of constant conditions, it could be integrated analytically. Many refrigeration systems are designed for mean environmental conditions, and in such cases, an analytical solution could predict the heat release profile without the need for numerical integration.

Chilling Model

The model used for the chilling stage heat load follows that of Pflug *et al* (1965), as extended by Cleland and Earle (1982) to deal with a greater range of shapes. The general equation for one-dimensional transient heat conduction is given by equation (7) (Welty, 1978):

$$\frac{1}{x^n}\frac{\partial}{\partial x}\left(x^n \frac{\partial T}{\partial x}\right) = \frac{1}{\alpha}\frac{\partial T}{\partial t} \qquad (7)$$

where $n = 0, 1, 2$, for an infinite slab, infinite cylinder and sphere respectively. For these three cases, there are exact analytical solutions and so the approach of Cleland and Earle (1982) was to calculate solutions for a simple reference shape, and use E to estimate solutions for more complex shapes from that reference. For reasons which will be discussed below, it is generally most useful to use the sphere as a reference shape when dealing with food products. For a sphere, the solution for the fractional unaccomplished mass average temperature change, $Y_{\sim}$ for cooling under constant conditions may be found to be:

$$Y_{\sim} = \frac{T_{\sim} - T_s}{T_i - T_s} = \sum_{i=1}^{\infty} \frac{6\,Bi^2}{\beta_i^2\left[\beta_i^2 + Bi\,(Bi-1)\right]} e^{-\beta_i^2 F_o} \qquad (8)$$

where β_i is ith root of:

$$\beta \cot\beta + (Bi - 1) = 0 \qquad (9)$$

During all but the initial period of the cooling process, the terms where $i > 1$ are insignificant. In that case, the slope of $\ln(Y_{\sim})$ as described by the abbreviated series solution ought to equal the slope of the integrated equation (2) when expressed in terms of $Y_{\sim}$. This implies that:

$$hA_s = \frac{E}{3}\frac{V\beta_1^2 k_f}{X^2} \qquad (10)$$

Substituted into equation (2), this gives:

$$\phi_{sat} = VC_t\frac{dT_\infty}{dt} = \frac{E}{3}\frac{V\beta_1^2 k_t}{X^2}(T_s - T_\infty) \qquad (11)$$

This is not an exact solution, although the importance of the implied assumption that $T_s = T_\infty$ is considerably reduced compared with equation (2). A corollary of equation (10) implies that:

$$\frac{6\,Bi^2}{\beta_1^2\left[\beta_1^2 + Bi\,(Bi-1)\right]} = 1 \qquad (12)$$

This is true with less than 10% error for $Bi < 3.5$, indicating that reasonable heat load estimates may be expected from equation (11) for Biot number values commonly found in food chilling and freezing.

The sphere is the most appropriate of the simple shapes to use as a reference due to the method which is used by the authors to evaluate E. Experience has shown that food products are generally best represented by equivalent ellipsoids when E is found by the method of McNabb *et al* (1990*a*,*b*), and that the chilling times estimated on this basis correspond best to finite difference results if a sphere reference is used. Details of how to apply these E-estimation methods are given by Hossain *et al* (1991*a*,*b*), but it must be noted that the McNabb method is strictly derived only for the phase change case. E values for chilling may be somewhat different from those for freezing, and this is an area of continuing research at the authors' laboratories.

As a result of the weaknesses described above, the heat load predicted by this method will underestimate the actual load at the start of the chilling process, and this error will increase as $Bi \to \infty$.

Freezing Model

The model used for the freezing stage was developed from the following assumptions, most of which hold much better for freezing than for chilling.

1. Sensible heat effects are negligible (or they can be bundled with the latent heat effects).
2. The body is homogeneous.
3. There is a sharp freezing front and all of the latent heat is released the freezing temperature T_f
4. It is possible to treat heat transfer at the surface of the body as varying linearly with the difference between the body surface and ambient temperatures (Newton's law of cooling, also known as the "third kind of boundary condition").
5. The cross-sectional area through which heat flows within the frozen region from the freezing front to the surface is related to the distanc from the product centre by equation (13).

$$A_x = x^n \qquad (13)$$

where n corresponds with that of equation (7). When $n = 0$, 1, or 2, equation (13) describes the cases of infinite slabs, infinite cylinders and spheres respectively. For other shapes, equation (13) approximates the three-dimensional conduction problem using one dimension, thereby allowing the conduction problem to be solved without the need for partial differential equations.

An energy balance at the freezing front requires that the rate at which heat is conducted away equals the rate at which that heat is released at the front, hence:

$$k_u \left(\frac{dT}{dx}\right)_{x=x_f} = L\left(\frac{dx_f}{dt}\right) \tag{14}$$

$$\Rightarrow \frac{dx_f}{dt} = \frac{k_u}{L}\left(\frac{dT}{dx}\right)_{x=x_f} \tag{15}$$

The heat release rate at the surface equals the rate at which heat flows through the frozen region:

$$hA_s(T_\infty - T_s) = k_u A_s \left(\frac{dT}{dx}\right) \tag{16}$$

Substituting the relationship in equation (13):

$$\Rightarrow \frac{dT}{dx} = \frac{h}{k_u} X^n x^{-n}(T_\infty - T_s) \tag{17}$$

$$\Rightarrow \int_{T_s}^{T_f} dT = \frac{h}{k_u} X^n (T_\infty - T_s) \int_x^y x^{-n} dx \tag{18}$$

Assuming that $n \neq 1$, integrating equation (18) gives equation (19):

$$T_s = \frac{(1-n)T_f - (h/k_u)X^n\left(x_f^{1-n} - X^{1-n}\right)T_\infty}{(1-n) - (h/k_u)X^n\left(x_f^{1-n} - X^{1-n}\right)} \tag{19}$$

Substituting equation (19) into equation (17) and thence into equation (15) gives equation (20). This equation can be regarded as a replacement for equation (3), with the advantages of greater simplicity and no need for a reference shape.

$$\Rightarrow \frac{dx_f}{dt} = \frac{T_\infty - T_f}{Lx_f^n\left[\frac{1}{hX^n} - \frac{\left(x_f^{1-n} - X^{1-n}\right)}{k_u(1-n)}\right]} \tag{20}$$

To estimate the freezing heat load from this, it is necessary to find an expression for the change in frozen volume with x_f. If it is assumed that the volumetric unfrozen fraction is related to the linear unfrozen fraction by equation (21),

$$\frac{V_f}{V} = \left(\frac{x_f}{X}\right)^N \tag{21}$$

then differentiating (21) obtains (22).

$$\frac{dV}{dx_f} = N\left(\frac{x_f}{X}\right)^{N-1}\frac{V}{X} \tag{22}$$

And so the freezing heat load at any given time, t, may be estimated by:

$$\phi = V\frac{dH}{dt} = L\frac{dx_f}{dt}\frac{dV}{dx_f} \tag{23}$$

N is a parameter which measures the dimensionality of the freezing front (eg. $N = 1$ implies a one-dimensional freezing front). The problem of evaluating N is considered below.

Product Enthalpy in Freezing and Sub-cooling

For food products, the freezing front is not sharp, and not all of the heat is released at T_f, so when equation (20) is integrated numerically, experience has shown that much better results are achieved by replacing T_f with T_{∞}. T_{∞} is re-calculated at each timestep from the commonly used product enthalpy function (Schwartzberg, 1976):

$$H = a + bT_{ss} + \frac{c}{T_{ss}} \tag{24}$$

which can be solved for T_{ss} to obtain:

$$T_{ss} = \frac{(H-a) - \sqrt{(H-a)^2 - 4bc}}{2b} \tag{25}$$

The value of H can be obtained at any time in the process from solving equation (23). The coefficients a, b and c are given for many food products by Pham (1987*a*), or they may be calculated by solving equation (24) at three points. For example, data for enthalpy H_f at the freezing temperature T_f, enthalpy H_{bse} at the enthalpy base temperature T_{bse} (usually -40°C), and the frozen heat capacity C_f at T_{bse} allow one to evaluate the following equations:

$$c = \frac{H_f - C_f(T_f - T_{bse})}{\frac{T_f - T_{bse}}{T_{bse}^2} + \left(\frac{1}{T_f} - \frac{1}{T_{bse}}\right)} \tag{26}$$

$$b = \frac{H_f - \frac{c}{T_f} + \frac{c}{T_{bse}}}{T_f - T_{bse}} \tag{27}$$

$$a = -bT_{bse} - \frac{c}{T_{bse}} \tag{28}$$

Sub-cooling Model

This model is the same as the chilling model, except that k_u is used instead of k_f and T_{ss} is calculated using the enthalpy function described above so that the remaining latent heat is included in the calculation.

The chilling model has a number of weaknesses which are apparent when it is used in the sub-cooling case. The presence of residual latent heat means that the notional heat capacity of the body changes during the subcooling period. This violates the assumption of constant heat capacity made in deriving the analytical

series solution for chilling under constant conditions. One would expect the subcooling heat load to be predicted poorly until the latent heat component becomes negligible.

The presence of the freezing front within the body during the first part of the sub-cooling period causes the internal geometry of the sub-cooling body to change during the process. This violates the assumption that the E geometry factor is constant during sub-cooling, a problem that does not occur during freezing as the freezing front is dealt with separately from the heat load. Again, one would expect this problem to decrease in importance during the latter part of the subcooling phase.

Criteria for Transition Between Models

There is no theoretical basis for deciding when to move from one model to the next during a numerical integration of the heat load estimation method. A number of criteria were considered:

1. Move from chill to freeze when the mass average temperature of the body drops to T_f.
2. Move from freeze to sub-cool when the mass average temperature drops to some point at which freezing is considered to be complete (e.g. -10°C).
3. Move from the current model to the next when the next model produces a higher heat load than the current one.
4. Move from the current model to the next when the rate of change of heat release in the current model is equal to the rate of change of heat release in the next model.

Of these, criteria 1 and 2 are physically reasonable only at low Biot numbers. At higher Biot numbers, they generate discontinuities in the predicted heat load profile. Criterion 4 provides a continuous first derivative for the heat load result,

but the equality condition is practically difficult to implement in a computer algorithm. Criterion 3 provides a heat load result without discontinuities (although transitions may not be smooth, ie. the first derivative is discontinuous), and realistically sites the transition points depending upon the Biot number for the process.

Criterion 3 was chosen as being the most appropriate for the chill–freeze transition. Criterion 3 was also chosen for the freeze–sub-cool transition, but some difficulties remained. Firstly, the transition from freeze to sub-cool is made before the freezing front has reached the centre of the product. This is desirable as it avoids the point where $x_f = 0$, under which circumstances the freezing front moves at a nominally infinite rate for all shapes but the slab. It does introduce the problem, however, that when the sub-cooling stage commences there is still some latent heat remaining in the body. Latent heat is dealt with by continuing to use the enthalpy-temperature function (equation (25)) to calculate $T_{\sim}$ during the sub-cooling stage, thereby implicitly re-estimating the specific heat capacity at each step of the process. The apparent specific heat of the material is quite high during the early part of the sub-cooling stage, as is true in reality. The same problem can occur at the transition from chill to freeze, but the remaining superheat can easily be bundled into the latent heat L, so there is less trouble in that case.

Secondly, criterion 3 alone is inadequate at high Biot numbers, and it causes a transition to sub-cooling when there is still a substantial latent heat load, despite that fact that the sub-cooling model is physically unrealistic when the latent heat load is still dominant. In practice, the overall method has been found to perform much better if transition to sub-cooling is not allowed until the freezing object is at least 80% frozen by volume, i.e.:

$$\left(\frac{x_f}{X}\right)^{n+1} < 0.2 \qquad (29)$$

The factor $\beta_1{}^2 k$ in equation (10) is re-calculated at the time of transition to make the initial sub-cooling stage heat load equal to the final freezing stage heat load. This is a rather crude correction, but high accuracy at this point in the process is not usually critical as the total heat load is quite small.

Evaluation of Freezing Equation Parameters

Now, n is not in fact a new parameter, but rather, $n = E-1$, so n may be evaluated by the same methods as are used for E (Hossain *et al*, 1991*a*,*b*).

At the start of freezing, the shape of the unfrozen region is the same as the shape of the outer surface, thus it can be seen from (21) that $N = 1, 2, 3$ for the slab, cylinder, and sphere cases respectively, and for all shapes, $N = E$ at $Bi = 0$. N initially follows the expression:

$$N_{i=0} = \frac{AX}{V} \qquad (30)$$

but the value of N for irregular shapes may be expected to change during the process when $Bi > 0$ as the shape of the unfrozen region becomes more rounded. An accurate estimate of N at any given time would depend strongly upon the shape of the body, and would consequently require a great deal of data for the more irregular shapes, as well as considerable computation. It could be postulated that N might change from the initial value given by equation (30) downwards, perhaps reaching E at the completion of freezing, but the mathematical function relating N to frozen volume may be very complex. To simplify the problem, it may be hoped that there is some average N for any given shape that is adequately close to the real (changing) N during the period of significant heat release. In the absence of other data, equation (30) was used as an approximation to this average N.

The latent heat of freezing, L, is not well defined for food products. For the purposes of this heat load prediction method, L may be conveniently defined by:

$$L = H_f - C_s (T_f - T_{\infty}) \tag{31}$$

If freezing starts when T_{∞} is still above T_f, the heat balance can be maintained if L is defined so that it includes the residual superheat:

$$L = H_f + C_f (T_{\infty} - T_f) - C_s (T_f - T_{\infty}) \tag{32}$$

PRACTICAL IMPLEMENTATION

Accurate estimation of the sub-cooling heat load is strongly dependent upon a correct estimate of the mass average temperature, T_{∞}. This can cause some difficulty as the predicted temperature is very sensitive to the accuracy of equation (25). Fortunately, the heat load is small during this stage, and accurate estimation is less important than it is earlier in the process.

When solving the ordinary differential equations numerically, care must be taken at the transitions between chilling and phase change, between phase change and sub-cooling, and when the freezing front is close to the centre of the object. Some of the less sophisticated numerical ODE solution methods may have difficulty at these points due to discontinuities and singularities.

There is a singularity in the freezing front position ODE for $n = 1$. It is not worth deriving the results for this special case, however, as one may simply use some value of n as close to 1 as the precision of one's computer allows, and expect that the resulting predicted heat loads will be negligibly different from the special case itself.

Summary of the Method as Implemented

The method was implemented in the following form:

1 Calculate the chilling heat load from

$$\phi_{sub} = V \frac{dH}{dt} = \frac{E}{3} \frac{V \beta_1^2 k_1}{R^2} (T_s - T_{\infty}) \tag{33}$$

2 while calculating T_{∞} from

$$T_{\infty} = \frac{H - H_f}{C_f} + T_f \tag{34}$$

3 Calculate the freezing heat load from

$$\frac{dx_f}{dt} = \frac{T_s - T_{\infty}}{L x_f^n \left[\frac{1}{hX^n} - \frac{\left(x_f^{1-n} - X^{1-n} \right)}{k_s (1-n)} \right]} \tag{35}$$

where L is calculated from equation (32)

$$\frac{dV}{dx_f} = N \left(\frac{x_f}{X} \right)^{N-1} \frac{V}{X} \tag{36}$$

and so

$$\phi_{freeze} = L \frac{dx_f}{dt} \frac{dV}{dx_f} \qquad (37)$$

while calculating T_{ss} from

$$T_{ss} = T_f \text{ if } H > H_f$$
$$\text{or } T_{ss} = \frac{(H-a) - \sqrt{(H-a)^2 - 4bc}}{2b} \text{ if } H \le H_f \qquad (38)$$

with a, b and c being found from equations (26) to (28) or some other source.

4 The predicted heat load is ϕ_{sNB} until $\phi_{freeze} > \phi_{sNB}$. After this is so, the predicted heat load is ϕ_{freeze} as calculated by step 2. Continue calculating ϕ_{sNB} as in step 1, using the T_{ss} of equation (38) and k_f instead of k_p as it is needed for the transition to sub-cooling.

5 When the frozen fraction is greater than 80% (i.e. the relationship in equation (29) is true) and $\phi_{freeze} < \phi_{sNB}$, recalculate the factor $\beta^2 k$ from equation (33) so that $\phi_{sNB} = \phi_{freeze}$ and go back to using ϕ_{sNB} from equation (33) as the predicted heat load.

6 Continue until T_{ss} is less than the required final temperature.

A computer program which implements this method in the Pascal programming language is available from the authors on request.

Heat Load Determination Under Constant Conditions

If h and T_a are assumed to be constant through the process, then equation (20) may be integrated with respect to time to give a recursive expression for the freezing front position, and hence the heat load at any given time.

$$\int dt = \int \frac{L}{T_a - T_f} \left[\frac{x_f^n}{(n+1)hX^n} - \frac{x_f^2}{2k(1-n)} + \frac{x_f X^{1-n}}{k(1-n)} \right] dx$$
$$\Rightarrow t = \frac{L}{T_a - T_f} \left[\frac{2x_f^{n+1}(1-n) - x_f^2 X^n \frac{h}{k}(n+1) + 2x_f^{n+1} Bi}{2hX^n(1-n^2)} + C \right] \qquad (39)$$

Now, when $t = 0$, $x_f = X$, the integration constant is:

$$C = \frac{-X\left(\frac{Bi}{2} + 1\right)}{h(1+n)} \qquad (40)$$

and thus:

$$\Rightarrow t = \frac{L}{T_a - T_f} \left[\frac{2x_f^{n+1}(1-n) - x_f^2 X^n \frac{h}{k}(n+1) + 2x_f^{n+1} Bi}{2hX^n(1-n^2)} - \frac{X\left(1 + \frac{Bi}{2}\right)}{h(1+n)} \right] \qquad (41)$$

The product is completely frozen when $x_f = 0$, and $t = t_f$, so:

$$t_f = \frac{LX\left(1 + \frac{Bi}{2}\right)}{(T_f - T_a)h(n+1)} \qquad (42)$$

This is just an expression of Plank's (1941) equation for the freezing time of a slab, with the addition of the $n+1$, factor which allows it to apply to other shapes, depending upon the value of $n+1$. It should be noted that this equation is derived only for the phase change process, and should not be used for general freezing time prediction — most of the methods described by Cleland (1990) are likely to produce much better overall freezing time predictions.

In cases where it is not necessary to obtain the whole heat load profile and if Plank's (1941) assumptions can be made, equation (43) may be derived from equation (41) which obviates the need to numerically evaluate the ODE from time zero. It is necessary to make an initial estimate for x_f ($x_f = 0$, or $x_f = X$ are satisfactory), and iterate equation (43) until x_f is sufficiently accurate (typically 8 iterations for less than 1% error).

$$x_f = \left[\frac{2hX^{n}(1-n^2)\left(\frac{t(T_a - T_f)}{L} + \frac{X(1+\frac{n}{2})}{h(1+n)} \right) + x_f^2 X^{n} \frac{1}{2}(n+1)}{2((1-n) + Bi)} \right]^{\frac{1}{n+1}} \qquad (43)$$

Having estimated x_f by this means, equations (20) and (22) may be evaluated, and the product heat load may be calculated from equation (23).

TESTING AGAINST FINITE DIFFERENCE RESULTS

Part 2 of this paper will describe experimental testing of the ODE method. Initial testing was carrried out against finite difference calculations. Comparisons of the ODE method against the finite difference Lees scheme described by Cleland (1990) are shown in Figures 1 to 3. The conditions of these tests were particularly demanding for any heat load prediction or freezing time estimation method:

Initial Temperature:	30°C	- high initial superheat
Final Temperature:	-12°C	- significant sub-cooling
Biot Number:	0.01 to 100	- wide range of *Bi*
Ambient Temperature:	-21°C	

The simulated product was the commonly used food product analogue Tylose (Cleland, 1990), for which accurate thermal property data are available. For each ODE run, N was calculated from equation (30).

Figure 1 shows results for a 200 mm thick slab ($E = 1.0$) with Biot numbers varying from 0.01 to 100. The time axis has been normalized by dividing by the freezing time, t_f. Heat loads are shown as ratios to the finite difference prediction, so that ideally, all of the ratios should equal 1.0 for the whole process. The plotted ratios for $Bi = 0.01$ and 0.1 are indeed very close to 1.0, but the ODE method predictions for even the highest values of Bi are well within 10% of the FD predictions for almost all of the cooling process.

Figure 2 shows results for a 200 mm thick cube cooled under the same conditions as the slab. This is a more difficult test than the slab as E for the cube varies from 3.0 at $Bi = 0.01$ to 2.23 at $Bi = 100$. Even so, the $Bi = 0.01$, 0.1, and 1.0 cases are still within 10% of the FD prediction throughout the process and the $Bi = 10$ and 100 cases are within 20% of the FD prediction until the process is 70% finished. Deviations late in the process are more acceptable due to the reduced total heat load at this time. Oscillations in the heat load ratio early in the process for the higher Bi cases were not due to the ODE method, but to discretization error in the finite difference calculation which could have been solved by increasing the number of integration nodes used (at the cost of more computer resources).

Figure 3 shows the ODE to FD heat load ratios for a selection of five shapes presented for $Bi = 1.0$. The sphere ($E = 3.0$) and infinite cylinder ($E = 2.0$, but used $E = 2.01$ to avoid the singularity in equation ?) had diameters of 200mm. The rectangular rod ($E = 1.899$) was 200 mm square and the finite cylinder ($E = 2.447$)

was 200 mm in diameter and 300 mm long. The meat carton (E = 1.326) was 154 mm by 335 mm by 510 mm. Again, all shapes were well within 10% of the FD prediction except very early and very late in the process.

Some ODE method calculations which assumed that N changed linearly with frozen volume from the value predicted by equation (30) to E were carried out in addition to the tests shown, but this produced no net improvement over using the N of equation (30) throughout the run.

Figure 4 puts the comparison in a practical perspective by demonstrating the scale of the heat loads involved for a typical New Zealand meat carton freezing situation. The meat carton run was the same as that shown in Figure 3. The result of the Cleland (1986) method is also shown for this case. It can be seen that the apparent error in the ODE heat load prediction at the end of the process is, in fact, trivial. Further, the current method is a significant improvement over the previous best ODE method of Cleland (1986).

DISCUSSION

The model testing demonstrated the validity of the assumptions made when deriving the freezing model. The assumptions that sensible heat effects were negligible, and that there was a sharp freezing front were ensured by the chilling→freezing transition condition as it allowed chilling to continue until the mass average temperature was only a little above T_f for the low Bi cases.

The assumption that the product was homogeneous was not tested, as only very complex finite element calculations are capable of providing accurate test data. This assumption is not troublesome, however, unless the properties of the materials within the product are substantially different.

The assumption that Newton's law of cooling applies at the product surface is applicable for the forced-convection boundary condition most often found in food

cooling. In practical situations, radiative heat transfer may have some influence, but this may be approximated by the use of a radiation heat transfer coefficient'. Natural convection may be dealt with in a similar manner.

All of the test cases had symmetrical cooling conditions. Where heat transfer coefficients or surface temperatures are asymmetric, a thermal centre may be defined which gives a depth equivalent to that of a symmetrical case with the same freezing time by using the method of Pham (1987*b*). The ODE heat load prediction method may then be used with the modified thermal centre depth.

The choice of $n+1$ as the exponent and of 0.2 as the maximum unfrozen fraction in equation (29) were not crucial — N would have been used as the exponent if it could be estimated accurately at such a high frozen fraction. If the subcooling model were improved, there would be no need for such a pragmatic approach to the transition criterion.

CONCLUSIONS

There has long been a need for a method of predicting the distribution of heat load through a freezing process. Until now, the available methods have either been very expensive computationally, or inaccurate. The method described here is comparable in accuracy to finite difference methods, while requiring some two orders of magnitude less computation for three-dimensional shapes. There is no need for a choice of reference shape, or for empirical constants. Additional functions may be derived from the heat load estimation method which allow heat load prediction at discrete points during the process and freezing time estimation for the phase change stage of the process.

With the availability of this method, refrigeration system designers and operators will be able to better characterize the freezing process and more closely match refrigeration supply to product heat load. For refrigeration system modellers, an accurate model of product heat load considerably reduces the uncertainty in

dynamic simulation of refrigeration systems for product-dominated applications, such as are often found in the food industry.

REFERENCES

Cleland, A. C. (1983). Simulation of industrial refrigeration plants under variable load conditions, *Int. J. Refrig.*, 6, 11-19

Cleland, A. C. (1985). Experimental verification of a mathematical model for simulation of industrial refrigeration plants, *Int. J. Refrig.*, 8, 275-282

Cleland, A. C. (1986). Use of dynamic simulation in the design of food freezing equipment, in *Food Engineering and Process Applications, Volume 2*, Le Maguer, M, Jelen, P (eds), Elsevier Applied Science Publishers, London, 55-65

Cleland, A. C. (1990). *Food Refrigeration Processes - Analysis, Design and Simulation*, Elsevier Science Publishers, London.

Cleland, A. C., Earle, R. L. (1982). A simple method for prediction of heating and cooling rates in solids of various shapes, *Int. J. Refrig.*, 5, 98-106

Fleming, A. K. (1976). Refrigeration demands for meat processing, *Refrigeration Science and Technology*, 1, 285-292

Hossain, Md. M., Cleland, D. J., Cleland, A. C. (1991*a*). Prediction of freezing and thawing times for foods of two-dimensional irregular shape using a semi-analytical geometric factor, *Int. J. Refrig.* (In press)

Hossain, Md. M., Cleland, D. J., Cleland, A. C. (1991*b*). Prediction of freezing and thawing times for foods of three-dimensional irregular shape using a semi-analytical geometric factor, *Int. J. Refrig.* (In press)

Loeffen, M. P. F., Earle, R. L., Cleland, A. C. (1981). Two simple methods for predicting food freezing times with time-variable boundary conditions, *J. Food Science*, 46, 1032-1034

Marshall, S. A., James, R. W. (1975). Dynamic analysis of an industrial refrigeration system to investigate capacity control, *Proc. Inst. Mech. Engrs.*, 189, 44/75, 437-445

McNabb, A., Wake, G. C., Hossain, Md. M. (1990*a*). Transition times between steady states for heat conduction. Part I - General theory and some exact results. *Occasional Papers in Mathematics and Statistics*, No. 20, Massey University, New Zealand

McNabb, A., Wake, G. C., Hossain, Md. M., Lambourne, R. D. (1990*b*). Transition times between steady states for heat conduction. Part II - Approximate solutions and examples. *Occasional Papers in Mathematics and Statistics*, No. 21, Massey University

Pflug, I. J., Blaisdel, J. L., Kopelman, J. (1965). Developing temperature-time curves for objects that can be approximated by a sphere, infinite plate, or infinite cylinder, *ASHRAE Trans.*, 71, pt 1, 238-248

Pham, Q. T. (1987*a*). A converging-front model for the asymmetric freezing of slab-shaped food, *J. Food Science* 52, 10, 795-800

Pham, Q. T. (1987*b*). Calculation of bound water in frozen food, *J. Food Science*, 52, 1, 210-212

Plank, R. (1941). Beitrage zur berechnung und bewertung der gefriergeschwindikeit von lebensmitteln, *Zeit. fur die gesamte Kalte Ind 10 Beih Reihe* 3: 1

Reynoso, R. O., De Michelis, A. (1988). Simulation of cryogenic batch freezers, *Int. J. Refrig.* 11, 6-10

Schwartzberg, H. G. (1976). Effective heat capacities for the freezing and thawing of food, *J. Food Science* 46, 153

Sokulski, M. B. (1972). Sharp freezer room calculation, *Refrigeration Science and Technology*, 2, 343-353

Szechowiak, E., Rainczak, J. (1987). The cooling chamber and air cooler operation in unsteady states, *Proc. 17th Int. Congr. Refrig.*, B, 864-869

Wade, N. L. (1984). Estimation of the refrigeration capacity required to cool horticultural produce, *Int. J. Refrig.*, 7, 358-366

Welty, J. R. (1978). *Engineering heat transfer — SI. Version*, Wiley, New York

A New Method of Predicting the Time-Variability of Product Heat Load During Food Cooling - Part 1: Theoretical Considerations

Figure Captions:

Figure 1: Ratio of ODE to FD heat load predictions — Slab, range of *Bi*

Figure 2: Ratio of ODE to FD heat load predictions — Cube, range of *Bi*

Figure 3: Ratio of ODE to FD heat load predictions — *Bi* = 1.0, range of shapes.

Figure 4: Predicted heat load of a meat carton shape.

FIG 1

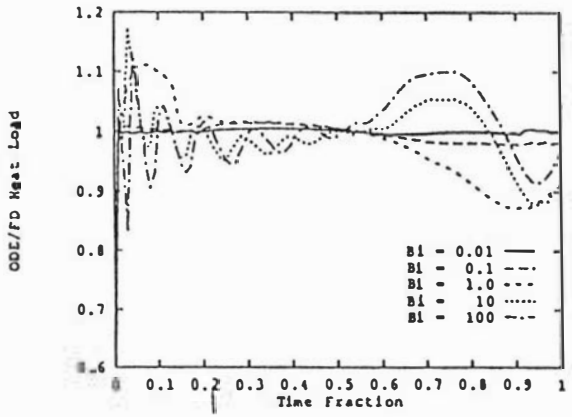
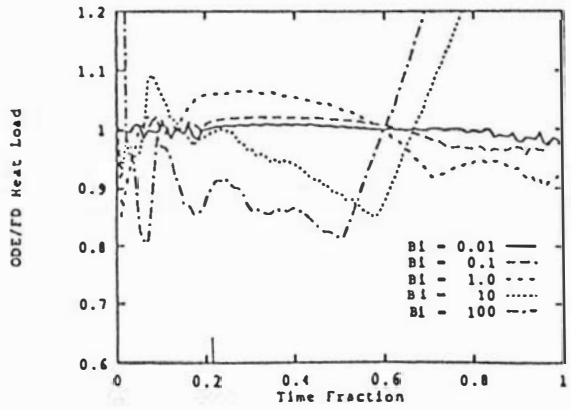


FIG 2



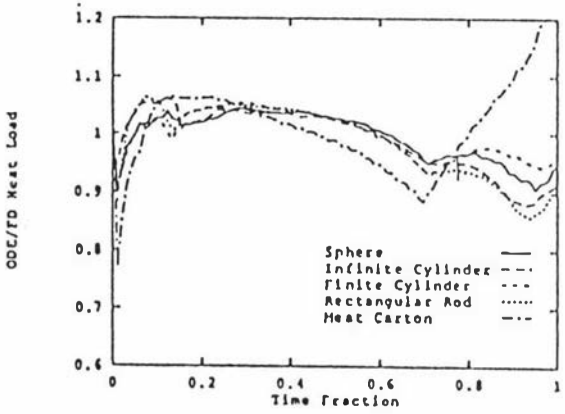
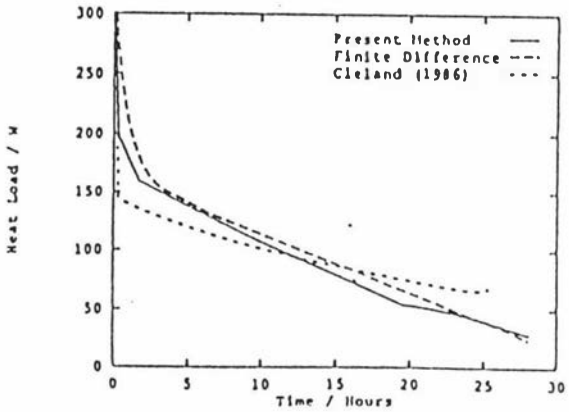

ODE/FD Heat Load
1.2
1.1
1
0.9
0.8
0.7
0.6
0
0.2
0.4
0.6
0.8
1
Time Fraction
Sphere
Infinite Cylinder
Finite Cylinder
Rectangular Rod
Heat Carton

FIG 4


Heat Load / W
300
250
200
150
100
50
0
0
5
10
15
20
25
30
Time / Hours
Present Method
Finite Difference
Cleland (1986)

A4.2 A New Method of Predicting the Time-Variability of Product Heat Load During Food Cooling — Part 2: Experimental testing

This manuscript is referenced in the thesis text as Lovatt *et al* (1992*b*). References in this manuscript to Hossain *et al* (1991*a*,*b*) correspond to the references Hossain *et al* (1992*a*,*b*) in the thesis text. References to Lovatt *et al* (1991) correspond to the reference Lovatt *et al* (1991*a*) in the thesis text — included as Appendix 4.1.

A New Method of Predicting the Time-Variability of Product Heat Load During Food Cooling - Part 2: Experimental Testing

S J Lovatt, Q T Pham, M P F Loeffen, A C Cleland
Massey University, Palmerston North, New Zealand, and Meat Industry Research Institute of New Zealand, Hamilton, New Zealand

ABSTRACT

An ordinary differential equation (ODE) method of predicting the heat load - time profile of cooling and freezing food product has been developed and tested against finite difference (FD) predictions for a selection of shapes. To test both the FD calculation and the ODE method against reality, a technique has been developed to measure freezing food product heat load in an experimental freezer using the difference in temperature between the air flowing onto and off the product. Heat balances for the experimental runs were generally within 10% of their expected values. When the FD and ODE heat load predictions for a meat carton shape were compared with measured data, the predictions matched the real values to within the experimental uncertainty. The ODE heat release predictions for lamb and sheep carcasses were also found to be within about 10% of the experimental data during most of the process, although it was necessary to modify the value of the one new parameter in the ODE method to deal with such highly irregular shapes.

NOTATION

A	Surface area of product (m^2)
E	Equivalent heat transfer dimensionality, calculated by the methods of Hossain *et al* (1991)

HTC	Heat transfer coefficient ($W/(m^2\ K)$)
N	Heat release geometry factor
M	Mass (kg)
Q	Volumetric flow rate (m^3/s)
T	Temperature (K)
V	Volume of product (m^3)
X	Critical depth (m)
ΔT	Temperature change (K)
ϕ	Heat flow (W)

Subscripts:

$_a$	ambient
$_{air}$	air
$_e$	electrical
$_{es}$	electrical steady state
$_p$	product
$_s$	surface

INTRODUCTION

Refrigeration is an important component of total costs in the New Zealand meat industry. Both capital and operating costs are affected by variations in the heat load on a plant's refrigeration system, so there is a strong motivation for developing ways of predicting load variations over time, and the response of the refrigeration system to these variations. In Part 1 (Lovatt *et al*, 1991), a new ordinary differential equation (ODE) method of predicting the heat load profile of cooling food products was proposed and tested against finite difference calculations. Part 2 reports the results of testing the model against experimental measurements.

Cleland (1985, 1986) has discussed the use of a full plant simulation to test a product heat load model, and reported some success. In that case, however, the

product heat load models comprised only about 10% of the dynamic models in the simulation, all of which were subject to error, and most of which interacted with the others to some greater or lesser degree. Product heat load was not measured explicitly, so the accuracy of the product heat load model could only be inferred from its effects upon secondary values such as the ambient air temperatures in each refrigerated room. While gross errors in the product heat load model may have been evident had they existed, some inaccuracies in Cleland's (1986) heat load model which have since been demonstrated (Lovatt *et al*, 1991) were not disclosed by the full plant simulation analysis.

It was apparent that the output of a product heat load model should be compared directly with the measured product heat load in order to provide adequate verification of the model, and this was the purpose of the work described here. Product types of most interest to the authors were two principal exports of the New Zealand meat industry: frozen whole lamb and sheep carcasses (up to 30kg) and frozen boned meat cartons (of about 27kg each).

DEVELOPMENT OF METHODS FOR MEASURING PRODUCT HEAT LOAD

The experimental measurement of product heat load is a difficult problem. Three techniques were attempted.

Method 1: Industrial freezer air cooler load measurement

This method was designed to measure the performance of one freezer in detail rather than attempting to measure overall plant performance in the fashion of Cleland (1985). This would allow a much more sensitive evaluation of the product heat load model than the full plant simulation had done. The minimum requirement was to measure the cooler air flow rate before or after the freezing process, and to monitor the air temperature change over the air cooler during the process using differential temperature probes. Heat infiltration could be estimated by

conventional means (Cleland, 1990), and the remaining heat load would be due to items within the room. There were a number of problems which restricted the generality of any results obtained:

1 There are other heat loads in a typical freezer room besides the product. In particular, the effect of structural loads will distort the measured load (the authors' experience suggests that an initially warm concrete floor may add 20% or more to the total load in a typical New Zealand batch freezer). Steady state loads such as the fans on forced convection evaporators may be accounted for by measuring electrical energy input. Time-variable loads such as those due to air interchange are difficult to estimate accurately.

2 Differential temperature measurements record sensible loads only so it is not practical to use this method for product where there is a significant latent load due to evaporation from the product.

3 Refrigeration conditions in the room may not remain consistent during the freezing process. Refrigerant evaporating temperatures may fluctuate considerably over the freezing period due to interactions with the rest of the plant, and in a batch freezer the air temperature will often fall during much of the process as the refrigeration load drops (Cleland, 1985).

4 If the refrigerant or air flow is varied to regulate air temperature, the measured instantaneous load may not bear any resemblance at all to the real load under constant conditions.

Preliminary trials of this method were undertaken as part of a related project, but it was not sufficiently precise when applied to batch freezers, and therefore it could not be used to measure the changing heat load as a product item froze. Errors in the energy balance were typically in the region of 50-100%, and there was no indication that this level of error could be substantially reduced without a radical change of approach.

Method 2: The Back-heat Approach

In principle, product heat load measurement required a form of calorimeter. Unfortunately, the product and process were not suitable for placing in a conventional calorimeter, so the principles of calorimetric operation were extended to a controlled-environment tunnel which was converted for the purpose.

The intention was to supply a constant refrigeration effect to the freezing tunnel throughout the process, and control an electrical heater in such a way that it held the air temperature constant. If this was done, then the heat load of the product at any given time would be equal to the difference between the electrical power input to the heater and the power required to maintain the same temperature in the absence of product. That is:

$$\phi_p = \phi_{\_} - \phi_h \qquad (1)$$

This followed the method of James and Bailey (1981) who applied a similar method to measure the heat load while chilling pork sides. When this technique was tried, there were found to be a number of problems with this method as well:

1 It was not practical to use alternating current to supply the heat required since it was difficult to measure AC current to sufficient accuracy (less than 5% error). This mandated the use of a direct current heater, which in turn limited the capacity of the heater to the capacity of the available DC source (about 650W).

2 It was very time-consuming to adjust the plant to the point were the heater was near, but below, maximum output and therefore had enough range to cope with the maximum load without dropping to zero output.

3 At the start of the freezing process, the peak load resulting from the warm product and air which had entered with the product exceeded the maximum capacity of the electrical heater, and so the air temperature became

temporarily uncontrolled with a deleterious effect on the estimated heat load.

4 It was very difficult to keep the refrigeration effect constant. In the experimental plant, refrigeration was supplied as chilled ethanol which circulated through the air cooler. The ethanol temperature was controlled to within 0.1°C of its setpoint, but even this small variation had a 2% effect on the supplied refrigeration effect due to changes to the logarithmic mean temperature difference. In turn, this resulted in larger percentage changes in the heater power (5% of maximum output) and even larger changes in the calculated heat load (10-15% of the mean freezing product heat load). This noise obscured much of the useful data.

5 The baseline, $\phi_{\_}$, was subject to significant shifts when the door was opened to load and unload product. Heat infiltration around the door was affected by the way in which the door seated, and this varied from closing to closing.

6 The large surface area of the environmental tunnel made heat infiltration an important factor. Runs had to be adjusted for changing heat infiltration due to the changing external temperature over the length of the run (12 to 60 hours depending on product and freezing conditions).

7 The heater power measured in this experiment was the output of the controller rather than the controlled variable. Whilst the air temperature was well controlled except immediately after starting a freezing run, the heater power output by the controller was subject to considerable noise. Detuning the controller helped reduce the noise level, but some noise remained due to this source

Some valid results were achieved through the use of this method, but data smoothing was required to make the results intelligible. The start of each run was

inaccurate due to starting effects, while the last half of each run had a heat load which was within the range of the data noise. Results could only be relied upon for the period after start-up and before the heat load dropped below about 40W. This was not acceptable. It was concluded that the method could only be made satisfactory if a purpose-designed experimental plant were have to have been constructed — a very expensive exercise.

Method 3: The Differential Air Temperature Approach

In an attempt to find a method which was less sensitive to external influences, the principle of the flow calorimeter was adapted to product heat load measurement. The environmental tunnel described for Method 2 was retained (see Figure 1), but a thermocouple array was added before and after the product in the air stream comprising 16 differential T-type thermocouples coupled together in series and spaced evenly across the cross-section of the tunnel. The change in air temperature as it passed over the product was measured using this array, and an anemometer traverse was carried out to measure the air flow rate in the working section of the tunnel, so that the sensible product heat load could be calculated from equation (2).

$$\phi_r = C_{ab}\, Q_{ab}\, \Delta T_{ab} \qquad (2)$$

A number of advantages ensued:

1 With the thermocouple arrays just each side of the product, the amount of heat infiltration variability which was included in the product heat load estimate was reduced by a factor of about 10 compared with having the whole tunnel surface infiltration included.

2 There was now no need for the tunnel air temperature to be held exactly constant.

3 With the improved air temperature tolerance, it was no longer necessary to include the refrigeration effect in the heat load calculations, and therefore a large source of noise was removed from the data.

There remained two principal disadvantages with this approach:

1 Only sensible heat could be measured. For this reason, a load cell was installed to monitor product weight during processes where latent heat was important. There was some difficulty in making that system function repeatably, so estimates of the latent heat contributions are not included in the results shown below.

2 There was some difficulty in accurately measuring the air flow rate. It was not possible to carry out a complete traverse across the tunnel because there were only three points in the tunnel ceiling where the anemometer could be inserted. Fortunately, from the data collected, there appeared to be little horizontal variation in air flow rate across the width of the tunnel. There was some variation in air flow rate with vertical position, but this was taken into account in the anemometer traverse.

Given the advantages of this method and the acceptability of its disadvantages, Method 3 was chosen as the best technique for producing repeatable product heat load profiles.

MATERIALS AND EXPERIMENTAL METHOD

Product Studied

Two product types and processes were chosen for study.

Carton Shape

The first process was the freezing of boned meat cartons in a moderate air velocity environment, as occurs in many New Zealand meat processing plants. To make the process more convenient to simulate with finite differences, the meat analogue Tylose MH1000 was used instead of boneless beef or lamb. The multi-layer cardboard boxes which are used in industry were also difficult to accurately quantify in a finite difference scheme, so open-topped light-gauge galvanised steel boxes were used to hold the Tylose. The boxes were 335mm wide by 510mm long, and were filled with Tylose to a depth of 154mm, making each of them essentially the same shape and volume as a typical export meat carton (see Figure 2). Two cartons were used in each run.

The Tylose was made up using the hot water method (described by Riedel (1960)) in which the Tylose powder was mixed with hot water prior to placing it in the boxes. The authors have found that Tylose made in this way is slower to set than Tylose made with cold water, and consequently results in easier mixing, a smoother surface, and a negligible amount of dry powder in the finished product. The only additive used was some 5g of copper sulphate per carton to retard the growth of moulds and bacteria. No salt was added.

Lamb and Sheep Carcasses

Another important product type in the industry is the frozen lamb or sheep carcass. Carcasses are typically frozen hanging by the hind legs in vertical air flow batch and continuous freezers. It was important to find the best way of representing such an irregularly-shaped product using the ODE heat load prediction method.

A variety of lamb and sheep carcasses were frozen at different air velocities and with different wrappings as found in the industry.

Experimental Procedure

The tunnel was cooled from room temperature to the working temperature, and then run until it equilibrated at the experimental conditions.

Meanwhile, the product to be frozen was allowed to equilibrate in a controlled-temperature room at 8°C. The Tylose cartons were left until their temperatures were negligibly different from the controlled room temperature, as monitored by a thermocouple probe in the centre of one of the two cartons. Two cartons were used in order to maintain sufficient heat load for accurate measurement during the experiment. Thermocouples were inserted in the centre and five surfaces of one carton, as shown in Figure 2.

Three replicates were conducted using the carton shapes and the heat loads measured during these runs are shown in Figure 4. Run 3 was halted early due to equipment failure.

The lamb and sheep carcasses were processed to ensure that they were hygienic, and met the New Zealand Accelerated Conditioning and Aging (AC&A) specification for high-quality frozen meat. The lamb and sheep were killed early in the morning, the carcasses were electro-stimulated within 30 minutes of sticking, and then placed in the 8°C controlled-temperature room for about 6.5 hours. The highest temperature in the carcasses was then typically about 18°C.

Before loading into the tunnel, each carcass was weighed and six thermocouples were inserted in various locations.

Each carcass was wired to a stainless steel rod in a configuration similar to the way in which it would hang naturally in a typical New Zealand vertical air flow blast freezer, and the rod was hung from a wire sling which would support it in the tunnel. This was necessary because the experimental tunnel was of the

horizontal air flow type, unlike the usual industrial situation, and it was desirable to keep the experiment directly comparable to industrial freezing practice.

Both product types were loaded into the tunnel in the same manner. The air cooler fan was stopped, the side loading door opened, and the product lifted inside. Carcasses were hung on their sling from a double hook in the ceiling of the working section, with hind legs towards the airflow. Cartons were placed on metal supports which allowed the air to pass along all surfaces of the cartons. The carton with the temperature probes inserted was placed on top of the other one, separated from it by similar supports. The six thermocouple probes were plugged into the data logging system, the side door closed, and the fan restarted. The whole operation usually took about 30 seconds, but never more than 60 seconds.

All of the runs were conducted with an ambient air temperature of -21°C (although the air temperature was not closely controlled, and therefore tended to drop from about -19.5°C to -21.5°C over the course of the run), and an average air velocity of 1.42m/s across the tunnel. Most runs were continued until the lowest measured temperature was within 1°C of the tunnel air temperature to make baseline heat load estimation easier. Some runs were terminated early due to mechanical problems.

Air Flow Rate Measurement

The major area of potential error in the heat load measured by the differential temperature method lay in the measurement of air flow rate in the tunnel. Measurement during an actual run was impractical due to the deleterious effect on heat load measurement accuracy of warm air infiltrating through the anemometer access holes, so the air flow rate was measured when there was no run taking place.

Initial air velocity measurements were conducted with the tunnel at room temperature, but early analysis of the runs indicated that the heat load was being

overestimated by a considerable fraction, and so the tunnel air velocity was re-measured at the operating temperature (-21°C). This was done even though the vane anemometer used for the purpose was not rated as accurate below 0°C (no anemometer available to the authors at the time was guaranteed at the temperature in question), but inaccuracy was minimized by only holding the anemometer in the cold air flow for a few seconds at a time — just long enough to obtain a consistent reading.

Air mass flow rate was calculated from the mean of the measured air velocities, the air density, and the cross-sectional area of the tunnel. While the mass flow rate would have been slightly different due to the increased pressure drop when the product was in the tunnel, the pressure drop caused by the product was expected to be negligible compared with the pressure drop within the air cooler and that caused by the direction changes as the air passed around the tunnel.

ANALYSIS OF EXPERIMENTAL RESULTS

Heat Load Baseline Estimation

The differential thermocouple heat load measurement technique could not measure the absolute sensible heat load of the product. Heat infiltration around the working area of the tunnel significantly affected the measured air temperature change and the extent of this infiltration load varied from run to run due to the quality of the loading door seal. It was necessary to estimate the base level of heat infiltration within the working area during each run. The baseline estimate accuracy could be assessed by checking the energy balance for the experiment — since the starting and finishing temperatures, and the mass of the product were all known, the theoretical change in enthalpy between the two temperatures could be estimated from product thermal properties.

The baseline estimation approach assumed that the heat load at the end of the run was a good estimate of the baseline. One early run showed that this was much

better than assuming the baseline to be estimated by the heat load measured immediately prior to loading the product because the heat load at the end of this run was much lower than the heat load before the start. For runs which were cut short for any reason, the baseline was assessed as being the lowest heat load measured from several hours before the start of the run (when logging was started) until the run finished. To reduce the influence of random error, that loads were averaged over 20 minute periods for baseline estimation purposes.

An example of the raw output from the differential air temperature thermocouple array (for carton run 1) is shown in Figure 3.

Carton Shapes

In Part 1 (Lovatt *et al*, 1991), the new ODE heat load prediction method had been tested against finite difference calculations, so the first objective of the regular shape experiments was to verify the accuracy of finite difference methods when predicting product heat load. To provide an meaningful comparison with the experimental results, a finite difference calculation was carried out for conditions similar to those encountered in the experiment. The simulated carton shape was started at the same uniform temperature as the experimental carton. The heat transfer coefficient used in the FD run was back-calculated to give the same freezing time (to -15°C centre temperature) as was measured in the experimental case.

The Tylose MH1000 used in this experiment had an initial freezing temperature of -1.2°C rather than the expected -0.68°C (Pham, 1987*a*). Density was measured by fluid displacement, thermal conductivity was obtained from Pham (1990), and enthalpy was obtained from unpublished experiments with an adiabatic calorimeter by one of the authors.

The comparison between the predicted FD and experimentally-measured heat loads shown in Figure 4 demonstrated that there was a problem with the overall

energy balance. The change in enthalpy over the process for runs 1 and 2 as estimated by numerically integrating the measured heat load was 25% and 12% greater than the theoretical result. The measured enthalpy change for run 3 was 4% less than the theoretical result, although it should be noted that this run was shorter than the others due to equipment failure.

The shapes of the FD and experimental heat load profiles were also quite different.

The failure to obtain an energy balance and the different shapes of the experimental and finite difference heat load profiles must have been due either to error in the experiment, error in the finite difference calculation, or some error in both. Both possible error sources were investigated.

Sources of Experimental Error

There was some initial heat load due to opening the tunnel door when loading the product. The size of this load was checked by carrying out the normal loading operation, but without actually loading any product into the tunnel. Loading effects were seen to be insignificant within five to ten minutes of commencing the test run.

The method used to estimate the heat load baseline was susceptible to some error. Since this error would have been different for each of the runs, it was checked by evaluating the difference between the three experimental profiles and a smoothed mean of the three. The standard deviation of the experimental profiles about this mean was found to be 6.5W, which was about 5% of the mean product heat load. Error in the heat load baseline estimate would have contributed only to the heat balance error.

The differential thermocouples may have been susceptible to radiation effects which could have caused the thermocouples to report higher temperatures than

actually existed in the air. This was indeed a problem while commissioning the experimental plant, but it appeared to be resolved by shielding each of the upstream thermocouples from the product with aluminium foil, while still allowing air to flow around the probes. There may have been some residual trouble with radiation effects, but this was not expected to be a major source of error.

As discussed above, the process of measuring the air mass flow rate in the tunnel was difficult and error-prone. A 10-15% error in the results of that procedure was quite conceivable, but it would affect the energy balance — not the shape of the heat load profile.

Sources of Finite Difference Error

The finite difference computer program used for these calculations has been well tested by several of the authors over 15 years, but its input data may not have accurately represented the reality of the experiment. One area of possible error was in the thermal properties used for the calculation. An error in the estimated enthalpy change may explain some of the energy balance error, but this was not expected to contribute more than about 2% error.

The radiation effects discussed with reference to the differential thermocouples were not considered in the finite difference calculation. The effective heat transfer coefficient included a radiation component which treated radiation as a pseudo-convective process. The error introduced by not treating radiation in the correct manner (proportional to the difference between the fourth powers of surface and ambient temperatures) was checked and found to be no more than about 4%.

The final assumption made for the finite difference run was that the heat transfer coefficient was the same on all parts of all surfaces. That this was not so was indicated by the plot of measured temperatures in the carton during the process. Figure 5 shows a plot of the six measured temperatures during the first carton

run. The thermocouple located under the upper (open) surface of the carton (T5 in Figure 5) remained much warmer than would have been expected if the heat transfer coefficient was as high on that face as it was on the other faces. Further, T1 and T4 (the leading and trailing surfaces) should have maintained similar temperatures (as the two surfaces were of the same area), but showed a difference of up to 5°C. T2 and T3 were as close as might be expected given slight variations in depth beneath their surfaces.

To test this hypothesis, the relative sizes of the heat transfer coefficients in the centre of each surface were evaluated separately using the method of Cleland and Earle (1976). The heat transfer coefficient found by back-calculating from the freezing time was scaled on each surface to these relative sizes, and the finite difference calculation repeated to produce the "Uneven HTC" line in Figure 4.

The temperatures predicted by this FD calculation for the centre of each surface are shown in Figure 6 for comparison with the experimental temperatures shown on Figure 5. T7 is the estimated temperature at the centre of the bottom surface of the carton. The two plots are similar, indicating that the estimated heat transfer coefficients were approximately correct.

The heat load estimated using the uneven heat transfer coefficients was a much better fit to that measured experimentally than that for uniform surface heat transfer. It should be noted, however, that to properly assess the effects of variable heat transfer coefficient, it would be necessary to measure heat transfer coefficients at more than five points on the carton. In fact, it is possible that heat transfer coefficient variation across different parts of the same surface may be at least as great as that found between faces. This was not explored experimentally.

Carton Results — Model Validation

The energy balance error was within the combined uncertainties of the air flow rate measurement method and the heat load baseline estimate. Improved precision may be possible, but air flow rate measurement precision may be expected to set a lower bound upon the overall accuracy of the experiment. Errors in enthalpy data may have been responsible for a small part of the heat balance error.

Under-estimation of the measured heat load by the FD method early in the run and over-estimation later probably resulted from uneven heat transfer effects. In practice, however, a typical carton freezer contains several thousand cartons and each carton moves through a variety of positions in the freezer, encountering a variety of air velocities and heat transfer coefficients (Pham and Willix, 1987) as it freezes. The total product heat load on an industrial freezer at any time represents the loads of all of these cartons together, but modelling each carton individually (even if only one heat transfer coefficient was assumed for each) would make a complete meat processing plant simulation very complex indeed. A more practical approach is only to model a typical carton with a typical even heat transfer coefficient and to rely upon the averaging effect of the large number of cartons to make the total predicted load similar to the total load on the real plant.

In Part 1 (Lovatt *et al*, 1991), it was suggested that the technique of Pham (1987*b*) for extending an ODE slab freezing time model to asymmetric heat transfer conditions could be applied to the ODE heat load prediction method, but no test results were reported. It would have been very complicated to apply this technique to three-dimensional asymmetry, but most of the carton heat content passes out through its top and bottom surfaces, so it was proposed that the technique only be used in this "primary" dimension and not in the other two. The critical depth of the carton was adjusted by Pham's (1987*b*) technique to compensate for the uneven heat transfer coefficients on the top and bottom surfaces and E was recalculated for the new critical depth. The shape of the freezing front in the

asymmetric case was not expected to be much different from the symmetric case so N was left unchanged and was still calculated with the original X value.

The heat load predicted by the asymmetric ODE calculation is shown in Figure 4 as the "ODE Method" line. It may be seen that the predicted heat load was very close to that of the "Uneven HTC" FD run except at the start of the process. While this does not constitute a comprehensive test of the ODE method under asymmetric conditions, it does provide some confidence that the method is capable of extension to complicated heat transfer situations.

Lamb Carcasses

Nine lamb carcass runs were conducted. Three which were conducted at an air velocity of 0.35 ms^{-1}, had very poor energy balances (ratios of measured to theoretical enthalpy change over the process of 1.86, 0.72 and 0.73), probably due to air flow stratification in the tunnel and the fact that lower product heat loads at low air velocities made measurement noise a larger fraction of the total load. These three runs were discarded. Summary results for the remaining six lamb carcass runs are shown in Table 1. The energy balances for these runs were much better, generally being within 10% of their expected values.

Given the air mass flow rate uncertainties discussed for the carton runs and the difficulty in estimating thermal property data for carcasses of varying composition, these runs had a tolerable level of error.

Variability in heat transfer coefficient over the product surface was expected to be less important with the carcasses than it was with the cartons. For most of the runs, the carcass surface was covered with wrappings which were expected to dominate the value of the heat transfer coefficient. Heat transfer into the cavity of the carcass was expected to be essentially zero and temperature measurements made by the probes inserted into various parts of the carcass confirmed this.

No finite difference or finite element calculations were conducted for the carcasses due to the difficulty of accurately describing their shape to the FD or FE method. The following analysis therefore focusses directly upon testing the ODE method. To apply the ODE model to a lamb carcass, it was necessary to obtain the values of the two shape parameters, E and N.

Evaluation of E for a lamb carcass

E has previously been evaluated for a lamb carcass by Cleland and Earle (1982), who chose an E which allowed freezing times calculated by freezing time prediction methods to best fit experimentally-measured freezing times. In the light of the work of McNabb *et al* (1990), the opportunity existed to evaluate E by a new technique.

The E-estimation method (Hossain *et al*, 1991) required that an irregular shape be approximated by an ellipsoid with the same volume, cross-section perpendicular to the major axis, and characteristic dimension. Three lamb carcasses of widely varying weights and grades were frozen and cut through in various areas to obtain this data. It was not possible to do this with the carcasses actually used the heat load experiment.

Firstly, each carcass was divided into three sections — leg (from the front of the pelvis backwards), loin (from the front of the pelvis to two ribs up from the base of the sternum, including most of the rack and all of the flap), and shoulder (the rest of the carcass, including the neck and forelegs). Each of these sections was expected to include a thermal centre (a unique location which would be the last part of the section to freeze). Each section was weighed and its volume calculated from its mass and density. The leg and shoulder sections were then cut in half vertically along the spine, and all sections were cut into slices perpendicular to the apparent major axis of the section (usually the central bone). The slice with the greatest characteristic dimension from each section was selected as the likely

location of the thermal centre for that section. For the shoulder and leg sections, both left and right halves were measured.

It has previously been assumed that the thermal centre of a whole lamb carcass lay at the "deep leg" location. Consideration of the freezing times shown in Table I indicates that low heat transfer within the carcass cavity for most of the experimental runs resulted in the last frozen point being within the shoulder, close to the inner cavity. For the purposes of the ODE heat load prediction method, the critical depth, X, was taken as being the full thickness of the shoulder, and E was calculated from the geometric data using the method of Hossain *et al* (1991) to be 1.48.

Evaluation of N for a lamb carcass

Part 1 (Lovatt *et al*, 1991) showed that for regular shapes, N was estimated by equation (3).

$$N = \frac{A\,X}{V} \qquad (3)$$

V, the carcass volume, was estimated from the known mass and estimated density. X was measured whilst evaluating E. Estimating the surface area of the carcass was more difficult, but if it was assumed that the carcass area was similar to the pelt area, then correlating the data of Fleming and Earle (1967) using a second order polynomial produced the following result:

$$A = 0.0653\,M - 0.000927\,M^2 \qquad (4)$$

where A was the pelt area of the carcass in square metres and M was the carcass mass in kilograms. The critical dimension (ie. $2X$) is shown in Table I. When calculated by this method, N was found to be about 3.7 for runs 1 and 2, and about 4.0 for runs 3 to 6.

Evaluation of Thermal Properties

The ODE heat load estimation method required the following thermal properties:

- Frozen thermal conductivity
- Unfrozen thermal conductivity
- Volumetric Enthalpy at the onset of freezing
- Frozen volumetric specific heat capacity
- Unfrozen volumetric specific heat capacity

It was impractical to render down entire carcasses in order to estimate a mean composition, so the enthalpies measured by Fleming (1969) were used together with thermal conductivities estimated from Fleming's measured compositions using the methods recommended by Miles *et al* (1983).

Evaluation of the Mean Heat Transfer Coefficient

The heat transfer coefficient in each run was evaluated indirectly. Given the measured time required for each carcass to freeze to -10°C, a freezing time prediction method of established accuracy (Pham, 1986) was used to back-calculate the mean heat transfer coefficient. The heat transfer coefficients thus derived are shown in Table I.

Comparison of Experimental Carcass Results with the ODE Method

When the N values found from equation (3) were used to predict the heat load profiles of the lamb carcasses studied in the experiment, the fit of the prediction to the experimental measurement was found to be poor. Figure 7 shows a typical example for carcass run 4.

In Part 1 (Lovatt *et al*, 1991), it was pointed out that equation (3) was only strictly correct at the start of the freezing process, and that N could be expected to vary after that time as the shape of the unfrozen region changed. In a lamb carcass, the

shape of the unfrozen region would quickly become quite different from the shape of the carcass surface, and therefore the true value of N would also change quickly. Further, the multiplicity of thermal centres (both shoulders, both legs, the loin, and possibly others) mean that later in the process there would be more than one unfrozen region, thus conflicting with one of the assumptions made in developing the model (Lovatt *et al*, 1991).

The range of values that may be assumed by N is an area of continuing research at the authors' laboratories. Hossain *et al* (1991) note that $E \leq A\ X\ /\ V$ under all conditions. Equation (3) was found to predict a single N value which was satisfactory for regular shapes (Lovatt *et al*, 1991). However, it was speculated that as N changes during the freezing process, it may be bounded by $A\ X\ /\ V$ and E. An attempt was made to model this change by assuming that it was linear with frozen volume, but this was not pursued as there was no net improvement compared with the results predicted by using a single N value.

An example of the predicted heat load for carcass run 4 using $N = E$ is shown in Figure 7. As a further bound, it should be noted that the regular shape with the highest surface area to volume ratio is the sphere, for which $A\ X\ /\ V = 3.0$. N values higher than 3.0 make the initial part of the heat load profile very steep, and the authors do not consider them likely in practical circumstances.

In the absence of better information, a trial and error approach was used and an N value of 2.5 was found to provide the best fit to the experimental data. Figures 8 to 13 show the experimental heat load profiles and the heat load profiles predicted by the ODE method for the six analyzed carcass runs. For each of these runs, the ODE heat load estimate was within about 10% of the experimental result during the first half of each run, and within 20% of the experimental result during the second half of each run, when the absolute heat load was smaller.

The sudden change in the direction of the predicted heat load curve about half way through each plot was due to a transition from the freezing model to the sub-

cooling model. The reasons for this lack of smoothness have been discussed in Part 1 (Lovatt *et al*, 1991).

DISCUSSION AND CONCLUSIONS

While the results of the product freezing heat load measurements were not as consistent as had been hoped, they were good enough to show that the differential air temperature measurement method was practical and useful.

Some of the problems encountered in the experimental technique were not fully resolved. Inaccuracies due to poor air distribution at low air velocities, and variability in heat transfer coefficient over the surface of the product were problems which reflected the environment in which the product was frozen. These problems may be lessened in practice by the averaging effect of the large numbers of carcasses or cartons found in an industrial freezer.

Direct measurement of heat transfer coefficients over the product surface is difficult. Even if good estimates of heat transfer coefficients were available for each part of the product, it would be difficult to include such detail in any relatively simple heat load prediction method. Further, it would be impractical to require that the user of such a method provide this data.

The ODE heat load estimation method described in Part 1 performed at least up to the standard of the measured data for both the carton and carcass runs. The expression $N = A X / V$ appears to be inadequate for very irregular shapes, although it was satisfactory for regular shapes. Where a shape is irregular and has more than one thermal centre, the authors suggest the following approximate guidelines:

1 Calculate $N = A X / V$.
2 Calculate E by the methods of Hossain *et al* (1991).
3 If N is greater than 3.0, set $N = 3.0$.

Note that these guidelines are based only upon the authors' experience, and should not be used once guidelines which have some theoretical basis are developed. The authors speculate that there may be some link between N and E, but while the nature of this relationship is uncertain at the time of writing, an additional guideline that seems to have some merit is to restrict N so that it may not be greater than about $E+1$.

Further work in this area would be interesting. The ODE heat load prediction method used in this paper could usefully be extended to other irregular shapes if better guidelines for estimating N and its variation with extent of freezing were developed. Experimental results for a wider range of air velocities, for chilling conditions with the air temperature above 0°C, and an improved level of accuracy would be desirable. The adiabatic calorimeter approach (in which the experimental tunnel would be located inside a cold room, as with the calorimeter of Fleming (1969)) could be useful. Improved air flow rate measurement techniques, and a more reliable heat load baseline estimate would be necessary to achieve a better energy balance accuracy than was found in this work.

REFERENCES

Cleland, A. C. (1986). Use of dynamic simulation in the design of food freezing equipment, in *Food Engineering and Process Applications, Volume 2*, Le Maguer, M., Jelen, P. (eds), Elsevier Applied Science Publishers, London, 55-65

Cleland, A. C. (1990). Experimental verification of a mathematical model for simulation of industrial refrigeration plants, *Int. J. Refrig.*, 8 (1985), 275-282

Cleland, A. C., *Food Refrigeration Processes — Analysis, Design and Simulation*, Elsevier Science Publishers, London

Cleland, A. C., Earle, R. L. (1976). A new method for prediction of surface heat transfer coefficients in freezing, *Refrig. Sci. Tech.*, 1, 361-368

Cleland, A. C., Earle, R. L. (1982). Freezing time prediction for foods — a simplified procedure, *Int. J. Refrig.*, 5, 134-140

Fleming, A. K. (1969). Calorimetric properties of lamb and other meats, *J. Food Technology*, 4, 199-215

Fleming, A. K., Earle, R. L. (1967). Cooling and freezing of lamb and mutton carcasses — 2. Weight loss during cooling, *Food Technology*, 22, 100-104

Hossain, Md. M., Cleland, D. J., Cleland, A. C. (1991). Prediction of freezing and thawing times for foods of three-dimensional irregular shape using a semi-analytical geometric factor, *Int. J. Refrig.*

James, S. J., Bailey, C. (1981) Measurement of product heat load during cooling, freezing and thawing of meat product, *Proc. Inst. Refrig.*, 78, 33-41

Lovatt, S. J., Pham, Q. T., Cleland, A. C., Loeffen, M. P. F. (1991). Prediction of product heat release as a function of time in food cooling - Part 1: Theoretical considerations, *J. Food Engineering*, (In press)

McNabb, A., Wake, G. C., Hossain, Md. M., Lambourne, R. D. (1990). Transition times between steady states for heat conduction. Part II - Approximate solutions and examples. *Occaisional Papers in Mathematics and Statistics*, No. 21, Massey University

Miles, C. A., van Beek, G., Veerkamp, C. H. (1983). Calculation of thermophysical properties of foods, in *Physical Properties of Food*, Jowitt, R. ed., Applied Science Publishers, London, 269-313

Pham, Q. T. (1986). Simplified equation for predicting the freezing time of foodstuffs, *J. Food Technology*, 21, 209-219

Pham, Q. T. (1987*a*). Calculation of bound water in frozen food, *J. Food Science.*, 51, 1, 210-212

Pham, Q. T. (1987*b*). A converging-front model for the asymmetric freezing of slab-shaped food, *J. Food Science*, 52, 3, 795-800

Pham, Q. T. (1990). Effect of Biot number and freezing rate on the accuracy of some food freezing time prediction methods, *J. Food Science*, 50, 1429-1434

Pham, Q. T., Willix, J. (1987). Heat transfer coefficients in the air blast-freezing of rows of cartons, *Proc. 17th Int. Congr. of Refrig.*, C, 350-357

Riedel, L. (1960). Eine Prufsubstanz Fur Gefrier-versuche, *Kaltetechnik* 12, 8, 222-225

Table I Lamb and Sheep Carcass Run Data

Run	Wrapping	Mass (kg)	Critical Dimension (m)	Exp Enth Chg (J)	Exp Enth Chg (J/kg)	Starting Temp (°C)	Ending Temp (°C)	Fre Time (hrs) Leg	Shoulder	HTC (W/m²K)	Th Enth Chg (J/kg)	Enth Chg Ratio
1	Shrink	11.67	0.131	14727009	252391	8.3	-18.0	10.3	15.1	14.4	247333	1.02
2	Shrink	10.48	0.124	16447569	313885	12.3	-19.7	12.8	12.7	17.0	264798	1.19
3	Poly+Stk	23.36	0.179	36361465	311314	17.4	-19.7	18.0	24.7	13.7	281934	1.10
4	Poly+Stk	26.65	0.190	35898854	269410	14.4	-19.8	14.7	21.4	18.5	272086	0.99
5	Bare	22.09	0.175	25803835	233623	17.7	-19.2	9.4	13.6	35.0	281773	0.83
6	Polybag	27.70	0.193	32263715	232965	14.8	-13.7	23.0	23.0	16.2	258074	0.90

Notes:

Shrink - Shrink wrapping
Poly+Stk - Polythene bag with stockinet covering
Polybag - Polythene bag

- Starting and Ending temperatures are means of all valid temperatures measured, and therefore only approximate the actual mean average temperatures at those times.
- Conditions for all runs were -21°C air temperature, and 1.42ms⁻¹ air velocity.
- Freezing times are to -10°C.
- Theoretical Enthalpy change is calculated from the Starting and Ending temperatures.
- The ratio of enthalpy changes is calculated by dividing the experimental measurement by the theoretical estimate. Note that for run 5, evaporative cooling was important, but was not measured by the differential air temperature method.

A New Method of Predicting the Time-Variability of Product Heat Load During Food Cooling - Part 2: Experimental Testing

Figure Captions:

Figure 1: Diagram of the MIRINZ environmental tunnel as set up for product heat load measurement.

Figure 2: Diagram of tylose-filled meat carton shapes as arranged for product heat load measurement. Thermocouple positions are numbered.

Figure 3: Sample output from the differential air temperature arrays. Spikes at 1200 and 4000 minutes indicate loading and unloading respectively.

Figure 4: Single meat carton shape heat load as measured and as predicted by both FD and ODE methods.

Figure 5: Product temperatures around a meat carton shape during freezing — Measured.

Figure 6: Product temperatures around a meat carton shape during freezing — Predicted by FD method with uneven heat transfer coefficient.

Figure 7: Measured heat load for a lamb carcass run, and loads predicted by the ODE method which show the effects of using extreme *N* values.

Figure 8: Measurement and ODE prediction of lamb carcass heat loads — Run 1.

Figure 9: Measurement and ODE prediction of lamb carcass heat loads — Run 2.

Figure 10: Measurement and ODE prediction of lamb carcass heat loads — Run 3.

Figure 11: Measurement and ODE prediction of lamb carcass heat loads — Run 4.

Figure 12: Measurement and ODE prediction of lamb carcass heat loads — Run 5.

Figure 13: Measurement and ODE prediction of lamb carcass heat loads — Run 6.

FIG 1

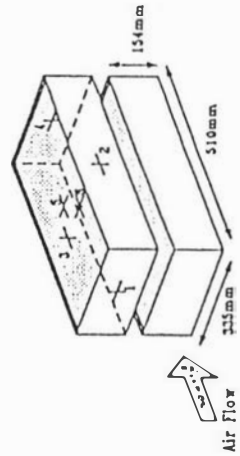


FIG 2

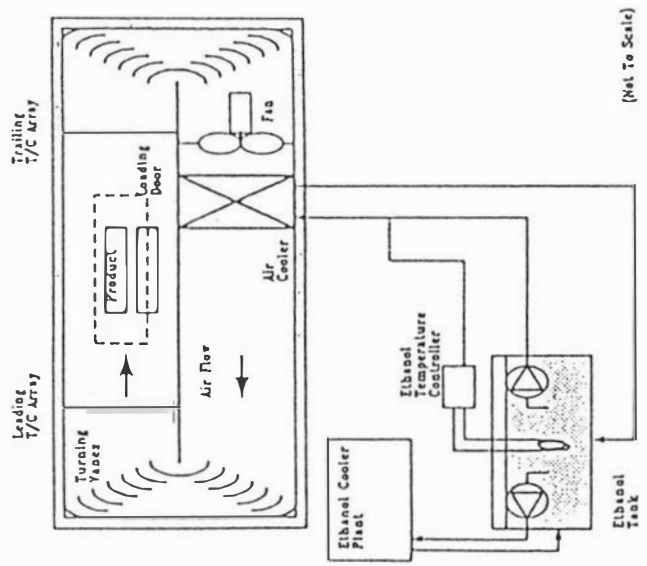


FIG 3

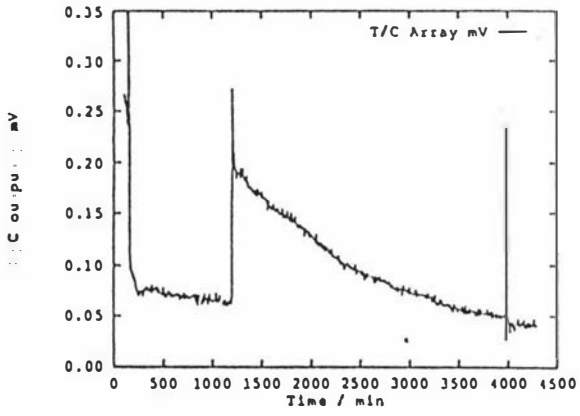


FIG 4

FIG 5

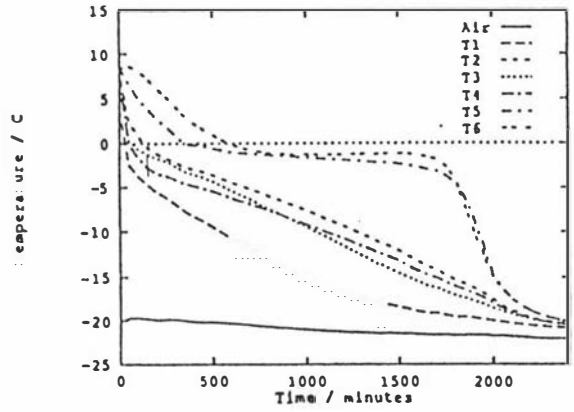
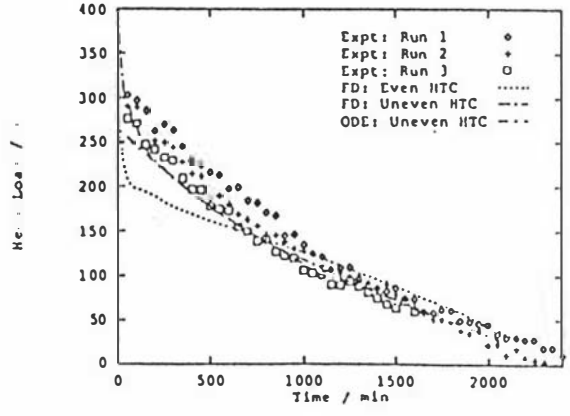


FIG 6



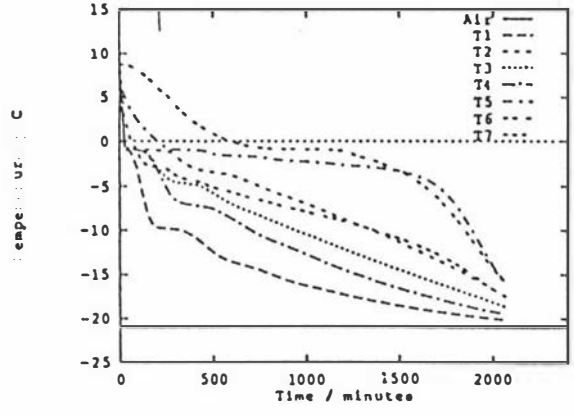


Fig 7

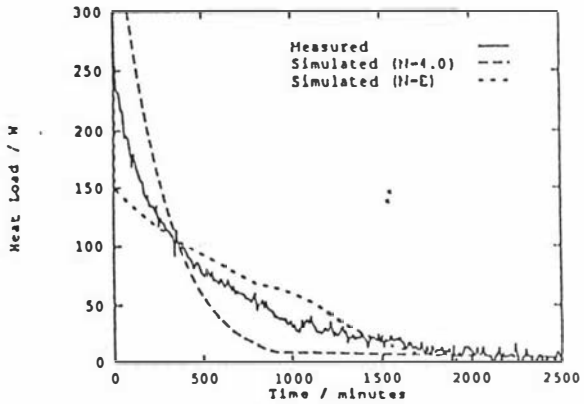


Fig 8

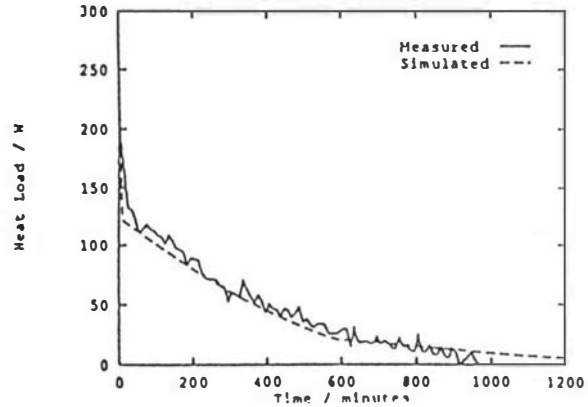


Fig 9

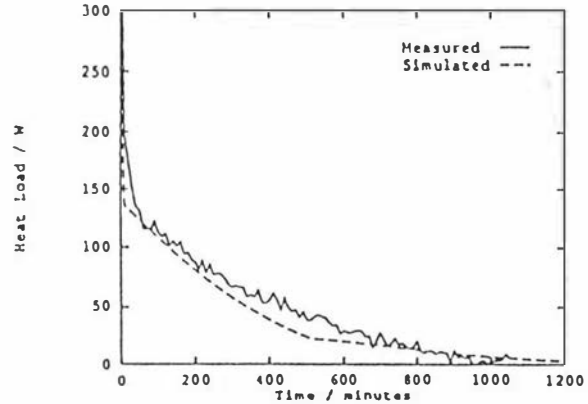


Fig 10

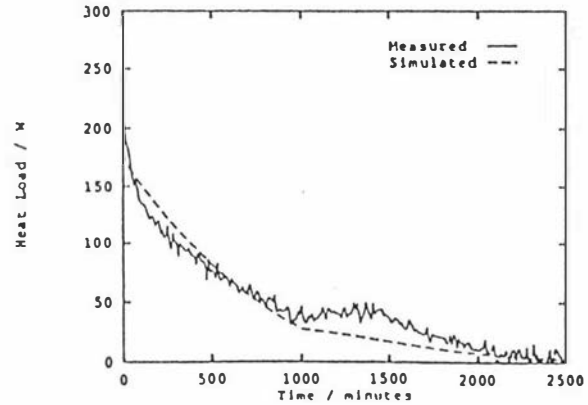


Fig 11

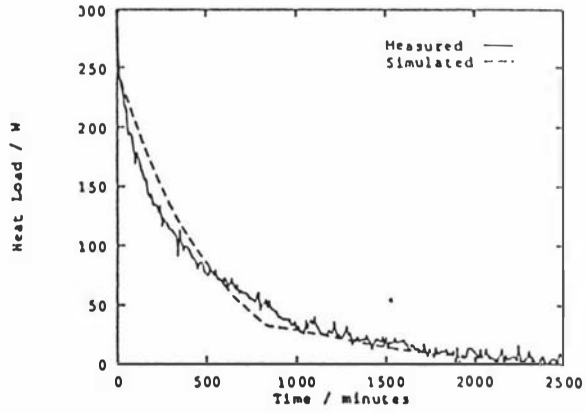


Fig 12

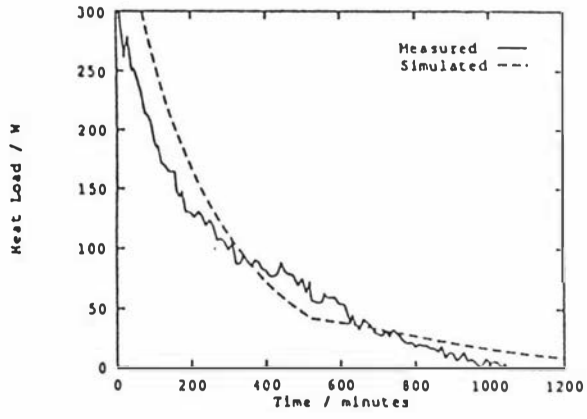
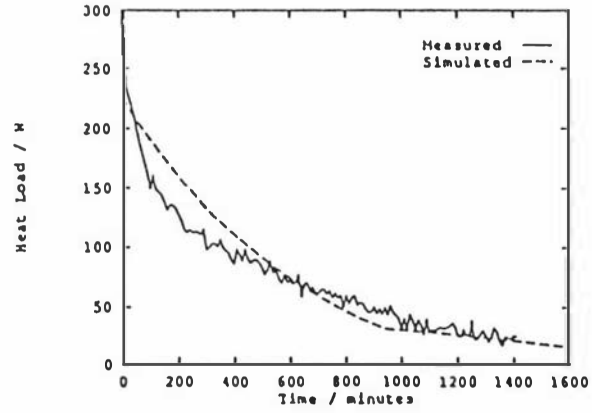


Fig 13



A4.3 Assessment of a Simple Mathematical Model for Predicting the Transient Behaviour of a Refrigeration System

This manuscript is referenced in the thesis text as Darrow *et al* (1991).

Appendix 5: Detailed descriptions of RefSim models

This Appendix contains detailed descriptions of all the models implemented within RefSim. The descriptions were generated by a computer program from the RefSim source code and are therefore an exact representation of the models as they are implemented in the simulation environment. The source code itself may be found in Appendix 8 (on diskette).

Models are listed in alphabetical order. The models included in this Appendix include all of the models derived from the base object *Model*. The Appendix therefore includes abstract models which are not useful of themselves and only serve to collect common properties together for their descendant models. There are also some variants on the main models which were developed to test ideas but were not subsequently used in simulations.

Model name

CCManualSched

Inputs

Suction controlled variable (of type InputType)

Outputs

ControlVar	Returns a setpoint
InputType	Returns the value of the controlled variable.

Comments

This compressor controller controls one compressor with scheduled times for suction and discharge connections and variable setpoint depending on the suction connection.

Properties

event	Priority queue of CCManualSched events which parallels the main event queue.
ControlledVar	Current value of the controlled variable.
SetPoint	Set point of controlled variable.
Suction	Pointer to the current compressor suction model.
Discharge	Pointer to the current compressor discharge model.
InputType	Variable type of controlled variable (e.g. Temperature, Pressure).

Methods

GetValue	Return ControlVar or InputType.
GetSuction	Return pointer to the current compressor suction model.
GetDischarge	Return pointer to the current compressor discharge model.
Event	Change from one set of Suction, Discharge, SetPoint to the next.
Evaluate	Evaluate the current value of ControlledVar.
Initialise	Initialise the CCManualSched model.

Parameters

Connections (Table)	Default for element i = 0.0
Connections (Table)	Default for element i+1 = ''
Connections (Table)	Default for element i+2 = ''
SetPoint (Table)	Default for element i = ''
SetPoint (Table)	Default for element i+1 = 0.0

Implementation

The Connections table contains three items per entry: Time /hrs, Suction model name, Discharge model name. The SetPoint table contains two items per entry:

Suction model name, set point for that suction model. The Initialise method queues the Connections table entries, along with the appropriate set points in its own priority queue and schedules discrete events to call the Event method of the current instance at the times specified in the Connections table. The Evaluate method does no more than obtain the current value of ControlledVar from the current suction model.

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
IF Suction <> NIL THEN
        ControlledVar := Suction^^.GetValue (SELF,InputType);
END;
```

Model name

CompController

Inputs

None	abstract model type

Outputs

None	abstract model type

Comments

This is the base class for refrigeration compressor controllers. These are different from SISO controllers in that they may change compressor suction and discharge connections. All three of the Getxxxxx methods should be re-implemented by child classes as the implementations in this class are dummies.

Methods

GetValue	Return normal data as per requests
GetSuction	Return a pointer to the model which is at the compressor suction.
GetDischarge	Return a pointer to the model which is at the compressor discharge.

Model name

ConvSignal

Inputs

InputType	as specified by the user.

Outputs

OutputV	the same value, no matter what variable is requested.

Comments

This models a linear + offset signal convertor. It accepts a specified input variable time from a specified Source model and produces an output to all requested variables which is defined by: Output = Factor * Input + Offset. This is especially useful in control loops and for converting the standard output of models like TimedOutput into other variables.

Properties

Source	Pointer to the source model.
Offset	Offset added to (Factor * Input).
Factor	Factor by which the input is multiplied.
OutputV	Value of the calculated output variable.
InputType	Type of input data to be read from Source.

Methods

GetValue	Return OutputV, regardless of the requested data type.
Evaluate	Calculate OutputV.
Initialise	Initialise the ConvSignal model.

Parameters

Factor	Default = 1.0
Offset	Default = 0.0
Units	Default = 'not_defined'
Variable	Default = 'ControlVar'
SourceModel	Default = ''

Implementation

```
IF Evaluated THEN
    RETURN;
ELSE
    Model.Evaluate;
END;
OutputV := Factor * Source^^.GetValue (SELF,InputType) + Offset;
```

Model name

DayTimeEnviron

Inputs

None

Outputs

as for the Environment model.

Comments

This is an Environment model which has set values for its variables between the Start and Stop times, and 0 values outside those times. It is used for intermittant loads and flows, such as those caused by machinery which is turned on and off regularly each day.

Properties

as for the Environment model, with the addition of:

SwitchedOn	BOOLEAN variable which is TRUE if the model is in the On state.
OnTime	Time at which the model will next turn on / s
OffTime	Time at which the model will next turn off / s
Onheatflow	Heat flow when the model is in the On state / W
Onmassflow	Mass flow when the model is in the On state / kg/s

Methods

Event	Changes the values of heatflow and massflow on schedule.
Initialise	Initialise the DayTimeEnviron model.

Parameters

as for the Environment model, with the addition of:

Start	Default = 0.0 / hours
Stop	Default = 24.0 / hours

Implementation

No calculations are carried out for the DayTimeEnviron model because the provided parameters remain constant throughout the simulation. The values of heatflow and massflow are changed by scheduled time events.

Model name

Door

Inputs

Temperature	from each side of the door
AbsHumidity	from each side of the door

Outputs

as for Room_Room_IFace

Comments

This model resembles that of RADS (Cornelius, 1991), but does not include a buffering factor used in the RADS door model and obtains opening times from an exponential distribution. Lengths of time open at each opening are obtained from a normal distribution given a mean opening time and a standard deviation.

Properties

Area	Area of door / m^2
Height	Height of doorway / m
OHTC	Overall heat transfer coefficient for the door when closed / W/m^2K
ProtFactor	Ratio of air movement through door to that predicted by Tamm's equation.
FirstOpen	Time at which the door will first open for the day / s
LastClose	Time at which the door will last close for the day / s
OpenLength	Length of time that the door is open at the current opening / s
AvgOpen	Mean length of time that the door stays open at each opening / s
StdDevOpen	Standard deviation of time that the door stays open at each opening / s
FractionOpen	Fraction of time between FirstOpen and LastClose that the door is open.
Open	TRUE if the door is currently open.

Methods

Evaluate	Calculate water vapour and heat flow through the door.
Event	Open or close the door.
Initialise	Initialise this instance of the Door model.

Parameters

OHTC	Default = 0.0 / W/m^2K
Width	Default = 3.0 / m
ProtFactor	Default = 1.0

Height	Default = 3.0 / m
FirstOpen	Default = 8.0 / hrs
LastClose	Default = 17.0 / hrs
FractionOpen	Default = 0.05
AvgOpen	Default = 2.0 / minutes
StdDevOpen	Default = 1.0 / minutes

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
temp1 := LinkedObjects^.model^.GetValue (SELF,Variable.Temperature);
temp2 :=
        LinkedObjects^.next^.model^.GetValue (SELF,Variable.Temperature);
IF Open THEN
use Tamm's equation and protection factor as per RADS 3.1
        hum1 := LinkedObjects^.model^.GetValue (SELF,Variable.AbsHumidity);
        hum2 :=
                LinkedObjects^.next^.model^.GetValue (SELF,Variable.AbsHumidity);
        dens1 := Air.Density (temp1,hum1);
        dens2 := Air.Density (temp2,hum2);
        IF dens1 < dens2 THEN
                densratio := dens1/dens2;
        ELSE
                densratio := dens2/dens1;
        END;
        vel := 5.91 * Sqrt (Height*(1.0 - densratio) /
                                Pow (1.0 + Pow (densratio, 1.0/3.0), 3.0));
        volflow := ProtFactor * 0.5 * Area * vel;
        enth1 := Air.Enthalpy (temp1,hum1);
        enth2 := Air.Enthalpy (temp2,hum2);
        WaterFlow := volflow * (dens1 * hum1 - dens2 * hum2);
        HeatFlow := volflow * (dens1 + dens2)/2.0 * (enth1 - enth2);
        HeatFlow only includes sensible heat. Latent heat is dealt with using WaterFlow.
        Note that the HeatFlow calculated here is set to maintain a mass balance.
        Normally, there should be a leakage factor, but this is accounted for at this point.
ELSE
        WaterFlow := 0.0;
        HeatFlow := OHTC * Area * (temp1 - temp2);
END;
```

Model name

DummyLoad

Inputs

Temperature	of the refrigeration equipment model to which it is connected.

Outputs

as for the Environment model.

Comments

This is a DayTimeEnviron model which represents a fixed heat load per degree temperature difference between its temperature and the connected model temperature. DummyLoad requires some input data, which makes it inconsistent with the rest of the Environment hierarchy, but it shares all of the other properties of DayTimeEnviron, so this is a convenient place for DummyLoad in the overall hierarchy.

Properties

as for the DayTimeEnviron model, with the addition of:

HeatFactor	Output heat load per degree temperature difference / W/°C

Methods

Evaluate	Calculate the heat flow from the DummyLoad.
Initialise	Initialise the DummyLoad model

Parameters

Temp	Default = 10.0
Start	Default = 0.0
Stop	Default = 24.0
HeatFactor	Default = 10000.0

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
IF SwitchedOn THEN
        othertemp := LinkedObjects^.model^.GetValue (SELF,Variable.Temperature);
```

```
        IF othertemp < temperature THEN
                heatflow := (temperature - othertemp) * HeatFactor;
        END;
ELSE
        heatflow := 0.0;
END;
```

Model name

Environment

Inputs

None

Outputs

Temperature	as set
Pressure	as set
Heatflow	as set
Massflow	as set
RelHumidity	as set
AbsHumidity	as set
State	as set

Comments

This is a utility model which can respond to requests for any sort of information but does not request information itself. This is useful for modelling things like the air outside a building, the ground under it, a refrigerant supply source etc.

Properties

temperature	°C
pressure	Pa
heatflow	W
massflow	kg/s
relhumidity	fractional 0 -> 1
abshumidity	wt/wt
state	fraction liquid

Parameters

Temperature	Default = 10.0
Pressure	Default = 1.0E5
Heatflow	Default = 0.0
Massflow	Default = 0.0
RelHumidity	Default = 0.0
AbsHumidity	Default = 0.0

State	Default = 1.0

Implementation

No calculations are carried out for the Environment model because the provided parameters remain constant throughout the simulation.

Model name

FluidTank

Inputs

Heatflow	into the FluidTank
Massflow	of liquid water or water vapour into the FluidTank

Outputs

Temperature	of air in the room
AbsHumidity	of air in the room
RelHumidity	of air in the room

Comments

This model defines a room filled with humid air.

Properties

Hum	Dynamic variable for absolute humidity
Temp	Dynamic variable for air temperature / °C
WaterMass	Dynamic variable for mass of condensed water in the room / kg
RelHum	Relative humidity
Volume	Volume of air in the room / m^3
Heat_Load	Net positive heat load on room / W
Water_Load	Net positive water flow into room / kg/s
TimeConstant	Ratio of volume to flow in the room / s

Methods

GetValue	Valid requests are Temperature, AbsHumidity, RelHumidity
Evaluate	Evaluate the FluidTank model
Initialise	Initialise the FluidTank model

Parameters

Mass	Default = 100.0
Temperature	Default = 20.0
HeatCapacity	Default = 4180.0

Implementation

This forms a simple model of a well-mixed tank of fluid.

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
obj := LinkedObjects;
HeatFlowIn := 0.0;
the net heat flow into the tank
Heat_Load := 0.0;
total heat flow into the tank
WHILE obj <> NIL DO
        Load := obj^.model^.GetValue (SELF,Variable.Heatflow);
        HeatFlowIn := HeatFlowIn + Load;
        IF Load > 0.0 THEN
                Heat_Load := Heat_Load + Load;
        END;
        obj := obj^.next;
END;
temp := RefSys.Bounded (Temp.val,-50.0,50.0);

Now calculate the temperature derivative

Temp.der := HeatFlowIn / (Mass * HeatCapacity);
Temperature changes due to heat flow into the tank
```

Model name

GenCondenser

Inputs

Temperature	of the coolant source
Heatflow	of refrigerant into the condenser

Comments

This model follows the general condenser model of Darrow et al (1991).

Properties

Temp	Dynamic variable for evaporation temperature / °C
RefrigCorr	Correction for non-standard refrigerant or operating mode.
UA	Heat transfer coefficient * Area product / W m^{-2}
SubCooling	Fixed discharge subcooling / °C

MassHeatCap	Heat capacity of the evaporator mass / J/°C
RefrHeatFlow	Refrigerant heat flow out of the condenser / W
RefSrc	Pointer to the refrigerant source.
RefDst	Pointer to the refrigerant destination.
Coolant	Pointer to the model which provides coolant.
CoolantMassFlow	Mass flow rate of coolant through condenser / kg s^{-1}
CoolHeatFlow	Heat flow rate into coolant / W
RefrigFlow	Mass flow rate of refrigerant through condenser / kg s^{-1}
Pressure	Refrigerant pressure in the condenser / Pa

Methods

Event	Carry out scheduled events to turn the evaporator on, off, or to start or stop defrosting.
Evaluate	Evaluate model.
Initialise	Set up model.

Parameters

Coolant	Default = ''
UA	Default = 1000
SubCooling	Default = 0.0 °C
MassHeatCap	Default = 100000.0 J/°C
RefrigCorr	Default = 1.0
Refrigerant	Default = 'R717'
Source	Default = ''
Dest	Default = ''

Implementation

The following implementation follows the model of Darrow et al (1991).

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
give the variables silly values so they get caught quickly
Ta          := -200.0;
Td          := -200.0;
obj         := LinkedObjects;
Ta := RefSys.Bounded (
        Coolant^^.GetValue (SELF,Variable.Temperature),-50.0,50.0);
RefrHeatFlow := RefSrc^^.GetValue (SELF,Variable.Heatflow);
Tout := Temp.val + (Ta-Temp.val) *
        MATHLIB.Exp (-UA/CoolantMassFlow/Water.HeatCapacity (Temp.val));
```

```
LMTD := -((Ta-Temp.val) - (Tout-Temp.val))
      / MATHLIB.Log ((Ta-Temp.val) / (Tout - Temp.val));

suctEnth := Refrig.VapourEnthalpy (Ref,Temp.val);
RefrigFlow := RefSrc^^.GetValue (SELF,Variable.Massflow);
dischEnth := Refrig.LiquidEnthalpy (Ref,Temp.val - SubCooling);
CoolHeatFlow := UA * LMTD;
netRefrHeatFlow := RefrHeatFlow - RefrigFlow*dischEnth;
Temp.der := (netRefrHeatFlow - CoolHeatFlow) / MassHeatCap;
Pressure := Refrig.EvapPress (Ref,Temp.val);
RefrHeatFlow := -RefrigFlow*dischEnth;
```

Model name

GenEvaporator

Inputs

Temperature	of the application
Heatflow	of refrigerant into and out of the evaporator

Comments

This model follows the general evaporator model of Darrow et al (1991).

Properties

Temp	Dynamic variable for evaporation temperature / °C
RefrigCorr	Correction for non-standard refrigerant or operating mode.
UA	Heat transfer coefficient * Area product / W m^{-2}
SwitchTime	Time of next On/Off switch event / s
OnTime	Length of time spent switched on / s
OffTime	Length of time spent switched off /s
SuperHeat	Fixed discharge superheat / °C
MassHeatCap	Heat capacity of the evaporator mass / J/°C
RefSrc	Pointer to the refrigerant source.
RefDst	Pointer to the refrigerant destination.
Application	Pointer to the application model.
SwitchedOn	TRUE if the evaporator is currently switched on.
EvapTempCtrlr	Pointer to the evaporation temperature controller.

Methods

Evaluate	Evaluate model.
Initialise	Set up model.

Parameters

Application	Default = ''

UA	Default = 1000
SuperHeat	Default = 0.0 °C
MassHeatCap	Default = 100000.0 J/°C
Start	Default = 0.0
Stop	Default = 24.0
Multiple	Default = 1.0
RefrigCorr	Default = 1.0
Refrigerant	Default = 'R717'
Source	Default = ''
Dest	Default = ''

Implementation

The following implementation follows the model of Darrow et al (1991).

```
IF Evaluated THEN
    RETURN;
ELSE
    Model.Evaluate;
END;
IF SwitchedOn THEN
give the variables silly values so they get caught quickly
    Ta          := -200.0;
    Temperature := -200.0;
    Td          := -200.0;
    obj         := LinkedObjects;
    Ta := RefSys.Bounded (
        Application^^.GetValue (SELF,Variable.Temperature),-50.0,50.0);
    RefrHeatFlow := RefSrc^^.GetValue (SELF,Variable.Heatflow);
    tmp := RefDst^^.GetValue (SELF,Variable.Heatflow);
    IF tmp <> MAX(LONGREAL) THEN
    active destination
        netRefrHeatFlow := tmp + RefrHeatFlow;
        suctTemp := RefSrc^^.GetValue (SELF,Variable.Temperature);
        IF EvapTempCtrlr <> NIL THEN
            tmp := EvapTempCtrlr^^.GetValue (SELF,Variable.ControlVar);
            IF suctTemp < tmp THEN suctTemp := tmp END;
        END;
        suctEnth := Refrig.LiquidEnthalpy (Ref,suctTemp);
        RefrigFlow := RefrHeatFlow/suctEnth;
        Temperature := Temp.val + SuperHeat;
        Td := RefSys.Bounded (Ta - Temp.val,0.0,50.0);
        RoomHeatFlow := UA * Td;
        Temp.der := (RoomHeatFlow + netRefrHeatFlow) / MassHeatCap;
        RefrHeatFlow := RefrHeatFlow + RoomHeatFlow;
```

```
        Pressure := Refrig.EvapPress (Ref,Temp.val);
    ELSE
    passive destination, so behave like a RADSEvaporator
        Pressure := RefDst^^.GetValue (SELF,Variable.Pressure);
        IF Pressure < 1.0 THEN Pressure := 1.0 END;
        keep safe from startup problems
        evaptemp := Refrig.EvapTemp (Ref,Pressure);
        IF EvapTempCtrlr <> NIL THEN
            tmp := EvapTempCtrlr^^.GetValue (SELF,Variable.ControlVar);
            IF evaptemp < tmp THEN evaptemp := tmp END;
        END;
        Td := RefSys.Bounded (Ta - Temp.val,0.0,50.0);
        RoomHeatFlow := UA * Td;
        RefrHeatFlow := RefrHeatFlow + RoomHeatFlow;
        RefrigFlow := RefrHeatFlow / dischEnth;
        Temp.der := (evaptemp - Temp.val) / 100.0;
        the time constant here (100.0s) is just a guess and not important -- the
        RADSEvaporator models assume that the evaporator has no time constant
        at all, and when working to a passive refrigerant destination this model
        works pretty much in the same way as the RADSEvaporator models
    END;
ELSE
    RoomHeatFlow := 0.0;
    RefrHeatFlow := 0.0;
    RefrigFlow := 0.0;
END;
```

Model name

Header

Inputs

Massflow	of refrigerant into the header from each linked model.
Heatflow	into the header from each linked model.
Pressure	of refrigerant at the Header outlet.
Temperature	of refrigerant at the Header outlet.

Outputs

Massflow	of refrigerant into the requesting model
Heatflow	into the requesting model
Pressure	of refrigerant in the Header
Temperature	of refrigerant in the Header

Comments

The Header model solves the problem of connecting a common pipeline to a number of refrigeration system components in the same way as is often done on a real refrigeration plant. A pipeline may either run into the header or out of it. If the pipeline runs into the header then an arbitrary number of models may comprise the outlets. If the pipeline runs out of the header then an arbitrary number of models may comprise the inlets.

Properties

PressDrop	Pressure drop through the Header / Pa
Pressure	Mean pressure in the Header / Pa
NumConnect	Number of components connnected to the Header.
RefrigFlow	Total mass flow of refrigerant through the Header / kg/s
Temp	Mean temperature in the Header / °C
HeatFlow	Total heat flow through the header / W
Pipe	Pointer to the pipeline model which is connected to the Header.
OutPipe	TRUE if the pipeline is the outlet to the Header.

Methods

GetValue	Return Heatflow, Massflow, Temperature and Pressure to requesting models.
Evaluate	Calculate the conditions in the Header and the flows through it.
Initialise	Initialise the Header model.

Parameters

Refrigerant	Default = 'R717'
PressDrop	Default = 0.0 / Pa
Source	Default = ''
Dest	Default = ''

Implementation

Flow through the header is from the defined inlet(s) to the defined outlet(s) only as this is a thermal model and full mass flow (and hence pressure) calculations are not carried out.

```
IF Evaluated THEN
      RETURN;
ELSE
      Model.Evaluate;
END;
heatFlow := 0.0;
refrigFlow := 0.0;
```

```
press := 0.0;
obj := LinkedObjects;
IF OutPipe THEN
the pipe goes out of the header and a number of devices feed into it
        minpress := Pipe^^.GetValue (SELF,Variable.Pressure);
        mintemp := Pipe^^.GetValue (SELF,Variable.Temperature);
        WHILE obj <> NIL DO
                IF obj^.model^ <> Pipe^^ THEN
                        refrigFlow := obj^.model^.GetValue (SELF,Variable.Massflow)
                                                + refrigFlow;
                        heatFlow := obj^.model^.GetValue (SELF,Variable.Heatflow)
                                                + heatFlow;
                END;
                obj := obj^.next;
        END;
        Pressure := minpress + PressDrop;
        Temp := mintemp;
ELSE
the pipe goes into the header and a number of devices take refrigerant out of it
        refrigFlow := Pipe^^.GetValue (SELF,Variable.Massflow);
        heatFlow := Pipe^^.GetValue (SELF,Variable.Heatflow);
        Temp := Pipe^^.GetValue (SELF,Variable.Temperature);
        minpress := MAX(LONGREAL);
        WHILE obj <> NIL DO
                IF obj^.model^ <> Pipe^^ THEN
                        press := obj^.model^.GetValue (SELF,Variable.Pressure);
                        IF press < minpress THEN
                                minpress := press;
                        END;
                END;
                obj := obj^.next;
        END;
        Pressure := minpress + PressDrop;
END;
RefrigFlow := refrigFlow;
HeatFlow := heatFlow;
```

Model name

HotWaterSource

Inputs

Temperature	of the Room
AbsHumidity	of the Room

Outputs

as for the Environment model.

Comments

This is a DayTimeEnviron model which represents an area of hot water inside an air-filled room. HotWaterSource requires some input data, which makes it inconsistent with the rest of the Environment hierarchy, but it shares all of the other properties of DayTimeEnviron, so this is a convenient place for HotWaterSource in the overall hierarchy.

Properties

as for the DayTimeEnviron model, with the addition of:

HTC	Heat transfer coefficient at the water surface / W/m^2K
Area	Surface area of the water / m^2

Methods

Evaluate	Calculate the heat and water vapour flow from the hot water surface
Initialise	Initialise the HotWaterSource model

Parameters

Temp	Default = 85.0
h	Default = 6.0
A	Default = 1.0
Start	Default = 0.0
Stop	Default = 24.0

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
IF SwitchedOn THEN
        airtemp := LinkedObjects^.model^.GetValue (SELF,Variable.Temperature);
        airhum  := LinkedObjects^.model^.GetValue (SELF,Variable.AbsHumidity);
        cp := Air.HeatCapacity (airtemp,airhum);
        sathum := Air.SatHumidity (temperature);
        ky := HTC/cp;
        the Lewis relationship
        massflow := ky * Area * (sathum-airhum);
```

```
        evaporation mass flow
        heatflow := massflow * (Water.VapourEnthalpy (temperature) -
                Water.VapourEnthalpy (airtemp));
                heat flow due to evaporated water cooling to air temperature
ELSE
        massflow := 0.0;
        heatflow := 0.0;
END;
```

Model name

Instrument

Inputs

None	abstact model type

Outputs

None	abstract model type

Comments

This is the base model for instruments and controls. Most of the controller models available in RefSim are capable of operating as either component of a cascade control system, so they are capable of reading their set points from other models as well as having a constant set point provided by the user.

Methods

None

Model name

LagRoom

Comments

This model defines a room filled with humid air. Identical to Room except for one additional property.

Properties

Enthalpy	Enthalpy of humid air in the room / J/kg

Implementation

The difference between this and the Room model is that the LagRoom TimeConstant affects the temperature in the room as well as the WaterMass. In particular, there is an additional ODE: Enthalpy; which maintains an energy

balance in the room. All heat passing into or out of the room affects Enthalpy immediately, but this effect is transferred to Temp more slowly. Thus Temp in a LagRoom is not the mass average air temperature as in a Room, but instead lags behind the heat flow. This may be an appropriate model for a poorly-mixed room.

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
obj := LinkedObjects;
HeatFlowIn := 0.0;
the net heat flow into the room
WaterFlowIn := 0.0;
the net water vapour flow into the room
LiqFlow := 0.0;
the net water liquid flow into the room
Heat_Load := 0.0;
total heat flow into the room
Water_Load := 0.0;
total water vapour flow into the room
WHILE obj <> NIL DO
        Load := obj^.model^.GetValue (SELF,Variable.Heatflow);
        HeatFlowIn := HeatFlowIn + Load;
        IF Load > 0.0 THEN
                Heat_Load := Heat_Load + Load;
        END;
        IF obj^.model^.GetValue (SELF,Variable.State) > 0.99 THEN
        the water flow is of liquid
                LiqFlow := LiqFlow
                        + obj^.model^.GetValue (SELF,Variable.Massflow);
        ELSE
        the water flow is of vapour
                WLoad := obj^.model^.GetValue (SELF,Variable.Massflow);
                WaterFlowIn := WaterFlowIn + WLoad;
                IF WLoad > 0.0 THEN
                        Water_Load := Water_Load + WLoad;
                END;
        END;
        obj := obj^.next;
END;
temp := RefSys.Bounded (Temp^.val,-50.0,50.0);
AirMass := Volume * Air.Density (temp,Hum^.val);
```

```
DryAirMass := Volume * Air.Density (temp,0.0);
SatHum := Air.SatHumidity (temp);
RelHum := Hum^.val / SatHum;

Now calculate the derivatives of the dynamic variables

WaterMassFlow := (RelHum - 1.0) * SatHum * DryAirMass/TimeConstant;
WaterMass is the mass of water in the room as frost, drip, or fog. TimeConstant will
depend on air movement rates in the room.
IF (WaterMass^.val < 0.0) AND (WaterMassFlow < 0.0) THEN
If the WaterMass^.val ever becomes negative, don't let it get worse.
        WaterMassFlow := 0.0;
END;

Load := WaterMassFlow * Water.VaporisationHeat (temp);
IF Load > 0.0 THEN
        Heat_Load := Heat_Load + Load;
END;

HeatFlowIn := HeatFlowIn + Load;
HeatFlowIn is sensible heat flow into the room. Can come from outside, objects inside the
room, or conversion from latent heat of water vapour

Enthalpy^.der := HeatFlowIn/AirMass;

Hum^.der := (WaterFlowIn - WaterMassFlow) / DryAirMass;
Humidity changes depending upon the water flowing into the room and the amount of
water in liquid or solid form in the room.

WaterMass^.der := LiqFlow + WaterMassFlow;
IF (WaterMass^.val < 0.0) AND (WaterMass^.der < 0.0) THEN
        WaterMass^.der := 0.0;
END;

Temp^.der := (Enthalpy^.val - Air.Enthalpy (Temp^.val,Hum^.val))
        * (AirMass / TimeConstant) / (AirMass * Air.HeatCapacity (temp,Hum^.val)
        + WaterMass^.val * Water.HeatCapacity (temp));
        Temperature changes due to heat flow into the room. Any WaterMass in the room
        provides a buffering effect
```

Model name

OUTPUT

Inputs

None	values are read from those variables registered with OUTPUT.

Outputs

None	OUTPUT does not communicate with other models.

Comments

OUTPUT is unlike other models in that it represents neither a real component, nor an abstract component class. It is part of the model hierarchy so that it may schedule and receive discrete events as models do.

Properties

OutputInterval	Interval between output events / s
OutputFile	Name of the file to which output is written.
VList	Pointer to a list of LONGREAL variables which are registered for output.
TList	Pointer to a list of titles to be placed at the heads of the columns in the output file.
RealOutputTime	Real time of last output (obtained from the system clock).

Methods

Register	Register a variable for output.
DisposeTitles	Dispose of the dynamic memory used to store titles.
DisposeVariables	Dispose of the dynamic memory used to store variable pointers.
Interval	Set the output interval.
Destination	Set the output file name.
Event	Write the current values of registered variables to the output file.
Headings	Write column headings to the output file
ReadReport	Write a report to the screen after reading in the data for a model.
Initialise	Initialise the output system.

Implementation

At each output Event, OUTPUT writes a report to the screen indicating the current simulation time, and the simulation rate (i.e. the ratio of simulated to real time). It then opens the output file, runs through the list of variables which are registered for output and appends their value to the output file. The output file is then closed

so that it is not lost in the event of an unexpected program halt (such as a user break, or electric power failure).

Model name

FINAL

Inputs

None — values are read from those variables registered with OUTPUT.

Outputs

as for OUTPUT

Comments

FINAL calls OUTPUT.Event to write the final values of all variables to the output file, and then disposes of all memory dynamically allocated by the program and HALTs the program.

Properties

as for OUTPUT

Methods

Event — Call OUTPUT.Event, tidy up and HALT the program.

Model name

PassiveEnv

Inputs

None

Outputs

All as for Environment, except for -

Heatflow — returns MAX(LONGREAL) to indicate that PassiveEnv is passive.

Comments

This is a utility model which can respond to requests for any sort of information but does not request information itself. This is useful for modelling things like the air outside a building, the ground under it, a refrigerant supply source etc. Unlike an ordinary Environment, PassiveEnv signals to other models that it is passive (i.e. it does not supply any heat load) by returning MAX(LONGREAL) to Heatflow requests.

Properties

As for Environment.

Parameters

As for Environment.

Implementation

As for Environment.

Model name

Pipeline

Inputs

None	abstract model

Outputs

Heatflow	Heat flow from the pipeline into the requesting model.
Massflow	Refrigerant mass flow from the pipeline into the requesting model.
Pressure	Refrigerant pressure at the requested end of the pipeline.
Temperature	Refrigerant temperature at the requested end of the pipeline.
State	Refrigerant fraction liquid at the requested end of the pipeline.

Comments

A Pipeline is a RefrigComponent with two ends, each of which may connect to another RefrigComponent, and one middle, which may be connected to an Environment (through which the pipeline passes). Each end may have different refrigerant properties (temperature, pressure, state) and the middle has mean properties.

Properties

Press	Array of two elements, each representing the pressure at one end.
Temp	Array of two elements, each representing the temperature at one end.
State	Array of two elements, each representing the liquid fraction at one end.
End	Array of two elements, each a pointer to the component model at one end.
PipeEnv	Pointer to the environment around the pipeline.

RefrigFlow	Refrigerant mass flow throught the pipeline.
HeatFlow	Heat flow through the pipeline.

Methods

GetValue	Return Heatflow, Massflow, Temperature, Pressure, State to the requesting model.

Model name

RADSACondenser

Inputs

Temperature	from refrigerant source and coolant
AbsHumidity	from coolant

Outputs

as for RADSCondenser

Comments

This model represents an air-cooled condenser.

Properties

as for RADSCondenser

Methods

Evaluate	Evaluate the air-cooled condenser model.

Parameters

as for RADSCondenser

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
IF MyController = NIL THEN
        loadfrac := 1.0;
        always runs full on
ELSE
        loadfrac := MyController^^.GetValue (SELF,Variable.ControlVar);
```

```
END;
IF loadfrac > 0.0 THEN
        condtemp := Source^^.GetValue (SELF,Variable.Temperature);
        airtemp := Coolant^^.GetValue (SELF,Variable.Temperature);
        airhum := Coolant^^.GetValue (SELF,Variable.AbsHumidity);
        ActualCoolFlow := CoolantFlow*loadfrac;
        IF condtemp < airtemp+0.5 THEN condtemp := airtemp+0.5 END;
        IF TempMode = RADSCondenser.Mean THEN
                w := RefSys.Bounded (condtemp-airtemp-TempChg,0.001,50.0);
                tempdiff := ((condtemp - airtemp) - w) /
                                        MATHLIB.Log ((condtemp - airtemp)/w);
        ELSE
                tempdiff := RefSys.Bounded (condtemp-airtemp,0.5,50.0);
        END;
        corr1 := (Corr1[1] + condtemp*(Corr1[2] + condtemp*(Corr1[3] +
                condtemp*Corr1[4])) + tempdiff*(Corr1[5] +
                condtemp*Corr1[6])) * tempdiff/Corr1[7];
        IF VelMode <> RADSCondenser.Correct THEN
                corr2 := 1.0
        ELSE
                corr2 := Corr1[1] + ActualCoolFlow*(Corr1[2]
                        + ActualCoolFlow*(Corr1[3] + ActualCoolFlow*Corr1[4]))
        END;
        Heat_Load := StdDuty*corr1*corr2*FoulingFactor;
        TempChg := Heat_Load /
                (ActualCoolFlow*Air.HeatCapacity (airtemp,airhum));
        TempChg := RefSys.Bounded (TempChg,0.1,condtemp-airtemp-0.1);
        vapenth := Refrig.VapourEnthalpy (Ref,condtemp);
        liqenth := Refrig.LiquidEnthalpy (Ref,condtemp);
        RefMassFlow := Heat_Load / (vapenth - liqenth);
        HeatFlowIn := RefMassFlow * vapenth;
        HeatFlowOut := RefMassFlow * liqenth;
ELSE
        Heat_Load := 0.0;
        RefMassFlow := 0.0;
        TempChg := 0.0;
        HeatFlowIn := 0.0;
        HeatFlowOut := 0.0;
END;
```

Model name

RADSBrCooler

Inputs

Temperature	of the application
AbsHumidity	in the application.
Temperature	of secondary refrigerant at the air cooler inlet
Heatflow	of secondary refrigerant into the air cooler

Comments

This is the base model for the evaporator models implemented in the RADS environment as described by Cornelius (1991). While there are three sorts of RADS evaporator, they have much in common, so the common evaluation is done here (initial evaluation in Eval1 and final evaluation in Eval2). The special parts of the evaluation are done by the children of this model.

Properties

Htmode	Temperature differences in the fitted polynomials are expressed as Mean or as FluidOn.
Hvmode	Correct or NoCorrect for cooled-fluid flow rate in the duty calculation.
Hhmode	Humidities are expressed either as TSRatio or RHumidity in the fitted polynomials.
TSRat	Actual total to sensible heat ratio of our duty
TempChange	Air temperature change across evaporator / °C
FanPower	Fan power / W
FaceVel	Face velocity / m/s
VolAirFlow	Volumetric air flow rate / m^3/s
Xfactor	Fin resistance factor
DefInterval	Time between defrosts / s
DefLength	Length of each defrost / s
DefPower	Power released during defrost / W
DefrostTime	Time of next defrost (on or off) event / s
SwitchTime	Time of next On/Off switch event / s
OnTime	Length of time spent switched on / s
OffTime	Length of time spent switched off /s
FanspeedCtrlr	Pointer to the fan speed controller
LiquidCtrlr	Pointer to the liquid level controller
RefSrc	Pointer to the refrigerant source.
RefDst	Pointer to the refrigerant destination.
Application	Pointer to the application model.
DefrostOn	TRUE if the evaporator is currently defrosting.
SwitchedOn	TRUE if the evaporator is currently switched on.

Methods

Event	Carry out scheduled events to turn the evaporator on, off, or to start or stop defrosting.
Evaluate	Evaluate model
Initialise	Set up model

Implementation

The calculations specific to RADSBrCooler are as follows:

```
IF Evaluated THEN
    RETURN;
ELSE
    Model.Evaluate;
END;
IF SwitchedOn THEN
    IF FanspeedCtrlr <> NIL THEN
        airFlow := VolAirFlow *
                        FanspeedCtrlr^^.GetValue
                        (SELF,Variable.ControlVar);
    ELSE
        airFlow := VolAirFlow;
    END;
    IF LiquidCtrlr <> NIL THEN
        liqFlow := SecRefFlow
            * LiquidCtrlr^^.GetValue (SELF,Variable.ControlVar);
    ELSE
        liqFlow := SecRefFlow;
    END;

    IF liqFlow > 1.0E-3 THEN
        Ta := Application^^.GetValue (SELF,Variable.Temperature);
        Hum := Application^^.GetValue (SELF,Variable.AbsHumidity);
        Tliq := RefSrc^^.GetValue (SELF,Variable.Temperature);
        tsurf := Ta - Xfactor*(Ta - Tliq);
        sathum := Air.SatHumidity (tsurf);
        htcap := Air.HeatCapacity (tsurf,0.0);
        IF ABS (Ta - tsurf) > 0.01 THEN
            TSRat := 1.0 + (Hum - sathum) * Water.VaporisationHeat (tsurf)
                        / (htcap * (Ta - tsurf));
            IF TSRat < 1.0 THEN TSRat := 1.0 END;
        ELSE
            TSRat := 1.0;
        END;
        RefrHeatFlow := RefSrc^^.GetValue (SELF,Variable.Heatflow);
```

```
            tmp := RefSys.Bounded (
                  (HeatLoad - FanPower) / TSRat + FanPower,0.1,HeatLoad);
            ratio := RefSys.Bounded (HeatLoad / tmp,1.0,MAX(LONGREAL));
            airFlowHtCap := airFlow * Air.Density (Ta,Hum)
                  * Air.HeatCapacity (Ta,Hum) * ratio;
            liqFlowHtCap := liqFlow * BrineSpHeat;
            htCapRatio := airFlowHtCap/liqFlowHtCap;
            XTU := UA / liqFlowHtCap;
            tmp := MATHLIB.Exp (-XTU*(1.0-htCapRatio));
            IF ABS (1.0 - htCapRatio*tmp) > 0.000001 THEN
                  eff := (1.0 - tmp) / (1.0-htCapRatio*tmp);
                  Tout := Ta - eff*(Ta - Tliq);
            ELSE
                  Tout := (liqFlow*BrineSpHeat*Ta + UA*Tliq)
                        / (UA + liqFlow*BrineSpHeat);
            END;
            HeatLoad := airFlowHtCap*(Ta - Tout);
            RoomHeatFlow := HeatLoad-FanPower;
            RefrHeatFlow := RefrHeatFlow + HeatLoad*TSRat;
            RefrigFlow := liqFlow;
            Temperature := Tliq + HeatLoad*TSRat / liqFlowHtCap;
            WaterFlow := HeatLoad*(TSRat-1.0)/Water.VaporisationHeat (tsurf)
                                          /TSRat;
      ELSE
            RoomHeatFlow := 0.0;
            RefrHeatFlow := 0.0;
            RefrigFlow := 0.0;
            Temperature := 0.0;
      END;
END;
```

Model name

RADSCompressor

Inputs

Suction	Pointer to suction model from CompController.
Discharge	Pointer to discharge model from CompController.
ControlVar	Required loading fraction from controller.
Pressure	From both suction and discharge models.

Outputs

Massflow	Refrigerant mass flow into and out of the compressor.
Heatflow	Refrigerant enthalpy flow into and out of the compressor.

Comments

This model follows the RADS compressor model described by Comelius (1991).

Properties

LiquidSource	Pointer to the model supplying liquid refrigerant for cooling.
Suction	Pointer to the compressor suction model.
Discharge	Pointer to the compressor discharge model.
MyController	Pointer to the compressor controller model.
InMassFlow	Refrigerant vapour mass flow into the compressor suction / kg/s
OutMassFlow	Refrigerant vapour mass flow out of the compressor discharge / kg/s
InHeatFlow	Refrigerant vapour enthalpy flow into the compressor / W
OutHeatFlow	Refrigerant vapour enthalpy flow into the compressor / W
LiqMassFlow	Refrigerant liquid enthalpy flow into the compressor / kg/s
LiqHeatFlow	Refrigerant liquid enthalpy flow into the compressor / W
SweptVol	Total compressor swept volume at standard speed / m^3/s
SuctPDrop	Refrigerant pressure drop in suction line / Pa
OpSpeed	Compressor operating speed / rpm
HeatRetention	Fraction of heat retained in compressor outlet vapour.
MaxDischTemp	Maximum allowable compressor discharge temperature / °C
MaxSuperHeat	Maximum allowable compressor discharge superheat / °C
Power	Compressor shaft power input / W
DischTemp	Discharge temperature / °C
SuctTemp	Suction temperature / °C
Capacity	Approximate refrigeration capacity generated by the compressor / W
MinLoadFrac	Loading fraction below which the compressor turns off completely.
VolEffy	Fractional volumetric efficiency
IsEffy	Fractional isentropic efficiency
PR	Compression ratio
FirstStage	Fraction of swept volume in the first stage of the compressor
StdSpeed	Operating speed at which the characteristic curve-fits were obtained / rpm
VCorr	Array of coefficients for the fitted volumetric efficiency polynomial.
ICorr	Array of coefficients for the fitted isentropic efficiency polynomial.
SCorr	Array of coefficients for the fitted speed vs. capacity polynomial.
Load	Array of allowable loaded fractions (if ContinuousUnload = FALSE).

PCorr	Array of coefficients for the fitted speed vs. power polynomial.
LCorr	Array of coefficients for the fitted part-load efficiency polynomial.
FixedSuction	TRUE if the model connected to the compressor suction does not change.
FixedDischarge	TRUE if the model connected to the compressor discharge does not change.
ContinuousUnload	TRUE if the compressor is capable of continuous unloading.
Cooling	Either None, Liquid, or External.

Methods

GetValue	Return Massflow or Heatflow to the requesting model.
Evaluate	Calculated the current compressor operating conditions.
Initialise	Initialise this instance of the RADSCompressor model.

Parameters

Cooling	Default = 'None'
Suction	Default = ''
Discharge	Default = ''
Controller	Default = ''
IsEffCorr (Table)	Default for element i = 0.0
VolEffCorr (Table)	Default for element i = 0.0
SpeedCapCorr (Table)	Default for element i = 0.0
SpeedPowCorr (Table)	Default for element i = 0.0
UnloadPowCorr (Table)	Default for element i = 0.0
LoadingSteps (Table)	Default for element i = 0.0
IsEffCorr (Table)	Default for element 1 = 1.0
VolEffCorr (Table)	Default for element 1 = 1.0
UnloadPowCorr (Table)	Default for element 1 = 1.0
Speed	Default = 300.0
StdSpeed	Default = -1.0
SweptVol	Default = 1.0
FirstStage	Default = 1.0
HeatRetention	Default = 0.9
MaxDischTemp	Default = -100.0
MaxSuperHeat	Default = -100.0
Refrigerant	Default = 'R717'
MinLoadFrac	Default = 0.0

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
Establish suction and discharge conditions
IF NOT FixedSuction THEN
        (MyController^^::mCompController).GetSuction (SELF,Suction);
END;
IF NOT FixedDischarge THEN
        (MyController^^::mCompController).GetDischarge (SELF,Discharge);
END;

Establish load fraction
IF MyController = NIL THEN
        loadfrac   := 1.0;
ELSE
        loadfrac   := MyController^^.GetValue (SELF,Variable.ControlVar);
END;
IF loadfrac < MinLoadFrac THEN loadfrac := 0.0 END;
IF (loadfrac > 0.0) AND (NOT ContinuousUnload) THEN
set the load fraction to one of the fractions
        i := 1;
        LOOP
                IF (loadfrac <= Load[i]) AND (loadfrac >= Load[i+1]) THEN
                loadfrac is between two load steps
                        IF (Load[i] - loadfrac) > (loadfrac - Load[i+1]) THEN
                        closer to Load[i+1]
                                loadfrac := Load[i+1]
                        ELSE
                        closer to Load[i]
                                loadfrac := Load[i]
                        END;
                        EXIT;
                ELSE
                        INC (i);
                        IF i = MAXCOEFF THEN
                                loadfrac := Load[i];
                                EXIT;
                        END;
                END;
```

```
        END;
END;

Read data from linked models, taking care to keep values within reasonable bounds to
deal with startup transients etc.
suctpress  := RefSys.Bounded (
        Suction^^.GetValue (SELF,Variable.Pressure) - SuctPDrop,
        40980.0,1.0E8);
satSuctTemp := Refrig.EvapTemp (Ref,suctpress);
dischpress := RefSys.Bounded (
        Discharge^^.GetValue (SELF,Variable.Pressure),40980.0,1.0E8);

        Initial calculations
PR         := dischpress / suctpress;
IF PR < 1.0 THEN PR := 1.0 END;
in case of startup problems
SuctTemp   := Suction^^.GetValue (SELF,Variable.Temperature);
deltemp    := Refrig.EvapTemp (Ref,dischpress);
saturation temperature at discharge as compared with DischTemp which may include
superheat
gamma      := Refrig.SupGamma (Ref,SuctTemp,SuctTemp,deltemp);
specvol    := Refrig.SupSpecificVolume (Ref,satSuctTemp,SuctTemp);
isenthchg  := gamma/(gamma-1.0) * suctpress * specvol
                                    * (Pow (PR,(gamma-1.0)/gamma) - 1.0);

VolEffy    := VCorr[1] + PR*(VCorr[2] + PR*(VCorr[3]
        + PR*(VCorr[4] + PR*VCorr[5]))) + deltemp*(VCorr[6]
        + PR*(VCorr[7] + deltemp*VCorr[8]));

IsEffy     := ICorr[1] + PR*(ICorr[2] + PR*(ICorr[3]
        + PR*(ICorr[4] + PR*ICorr[5]))) + deltemp*(ICorr[6]
        + PR*(ICorr[7] + deltemp*ICorr[8]));

        efficiency := V1 + V2*PR + V3*PR^2 + V4*PR^3 + V5*PR^4 + V6*deltemp +
        V7*PR*deltemp + V8*PR*deltemp^2

VolEffy := RefSys.Bounded (VolEffy,0.05,1.0);
IsEffy  := RefSys.Bounded (IsEffy, 0.05,1.0);

qtheor  := SweptVol * VolEffy * loadfrac * FirstStage;
standard theoretical volumetric capacity (in cu m /s) depends on swept volume at std
speed, volumetric efficiency, loading fraction, and the fraction of swept volume in the first
stage (1.0 for single stage machines, <1.0 for compound machines)
```

```
Corrections for speeds different from those at which the standard condition curve fits were
taken
capacitycorr := SCorr[1] + SCorr[2]*OpSpeed;
powercorr    := PCorr[1] + OpSpeed*(PCorr[2] + OpSpeed*PCorr[3]);

capacitycorr := RefSys.Bounded (capacitycorr,0.01,10.0);
powercorr    := RefSys.Bounded (powercorr,0.01,10.0);

qtheor := qtheor * capacitycorr;

InMassFlow := qtheor / specvol;
Power  := InMassFlow * isenthchg/IsEffy * powercorr;
full load power in W

Part-load efficiency correction

IF loadfrac > 0.999 THEN
        partloadeffy := 1.0;
ELSE
        partloadeffy := LCorr[1] + loadfrac*(LCorr[2] + loadfrac*LCorr[3]);
END;
IF partloadeffy > 0.01 THEN
        Power := Power/partloadeffy;
ELSE
        Power := Power/0.01;
END;

Compressor cooling calculation
suctenth := Refrig.SupEnthalpy (Ref,satSuctTemp,SuctTemp);
maxenth  := suctenth + isenthchg/IsEffy/partloadeffy;
superheat := (maxenth - Refrig.VapourEnthalpy (Ref,deltemp))
        / Refrig.VapourSpecHt (Ref,deltemp);
        estimate the uncooled superheat
IF Cooling = None THEN
        dischenth := suctenth + HeatRetention*isenthchg/IsEffy/partloadeffy;
        the fraction HeatRetention of the enthalpy change is retained in the heat flowing
        out of the compressor
        DischTemp := deltemp + HeatRetention*superheat;
ELSE
        IF MaxDischTemp > deltemp + MaxSuperHeat THEN
                DischTemp := MaxDischTemp
                the MaxDischTemp limit is higher
        ELSE
```

```
            DischTemp := deltemp + MaxSuperHeat
            the MaxSuperHeat limit is higher
        END;
        IF DischTemp > (deltemp+superheat) THEN
            DischTemp := deltemp+superheat;
        END;
        dischenth := Refrig.SupEnthalpy (Ref,deltemp,DischTemp);
END;
heatloss := RefSys.Bounded (InMassFlow*(maxenth-dischenth),0.0,
                            MAX(LONGREAL));
IF heatloss <= 0.0 THEN dischenth := maxenth END;
this is the heat lost from the refrigeration system in some way
IF (Cooling = Liquid) AND (LiquidSource <> NIL) THEN
        liqenth := Refrig.LiquidEnthalpy (Ref,
                    LiquidSource^^.GetValue (SELF,Variable.Temperature));
        LiqMassFlow := heatloss/(dischenth - liqenth);
        LiqHeatFlow := RefSys.Bounded (LiqMassFlow*liqenth,0.0,MAX(LONGREAL));
        heatloss := 0.0;
        no heat is actually lost, it all ends up in the discharge
ELSE
        LiqMassFlow := 0.0;
        LiqHeatFlow := 0.0;
END;

Now calculate output data
OutMassFlow := InMassFlow + LiqMassFlow;
InHeatFlow := InMassFlow * suctenth;
OutHeatFlow := OutMassFlow * dischenth;

Include any liquid refrigerant heat flows in the flows for the suction or discharge vessels
if the liquid is sourced from one of those (so that it is only necessary for the vessel to ask
for one heat flow from the compressor)
IF Suction = LiquidSource THEN
        InHeatFlow := InHeatFlow + LiqHeatFlow;
ELSIF Discharge = LiquidSource THEN
        OutHeatFlow := OutHeatFlow - LiqHeatFlow;
END;

Estimate compressor capacity for output purposes
Capacity := InMassFlow
        * (suctenth - Refrig.LiquidEnthalpy (Ref,deltemp));

heatbal := InHeatFlow - OutHeatFlow - heatloss + LiqHeatFlow + Power;
```

Model name

RADSCondenser

Inputs

None	no evaluation carried out in this model

Outputs

Heatflow	into and out of the condenser
Massflow	through the condenser

Comments

This is the condenser part of the RADS condenser model as described by Cornelius (1991). The original RADS model was, in fact, a combination of condenser and refrigerant vessel. In the RefSim implementation, these functions have been separated.

Properties

Heat_Load	Net heat flow out of the condenser into the coolant fluid / W
HeatFlowIn	Heat flow into the condenser / W
HeatFlowOut	Heat flow out of the condenser / W
TempChg	Coolant temperature change across the condenser / °C
CoolantFlow	Maximum coolant flow rate through the condenser / kg/s
StdDuty	Net heat flow out of the condenser into the coolant fluid under standard conditions / W
ActualCoolFlow	Actual coolant flow rate through the condenser at reduced loading / kg/s
RefMassFlow	Refrigerant mass flow through the condenser / kg/s
FoulingFactor	Heat transfer resistance fouling factor.
Corr1	Array of correction coefficients for the temperature and temperature difference duty correction polynomial.
Corr2	Array of correction coefficients for the coolant flow rate duty correction polynomial.
TempMode	Temperature differences expressed as either FluidOn or Mean.
VelMode	Either Correct or NoCorrect. Indicates whether to correct duty for coolant flow rate with Corr2.
MyController	Pointer to the condenser controller.
Source	Pointer to the condenser refrigerant source.
Dest	Pointer to the condenser refrigerant destination.
Coolant	Pointer to the coolant model.

Methods

GetValue	Return Heatflow and Massflow to requesting models.

Initialise	Initialise this instance of the RADSCondenser model.

Parameters

Corr1 (Table)	Default for element i = 0.0
Corr2 (Table)	Default for element i = 0.0
TempMode	Default = 'Mean'
VelMode	Default = 'NoCorrect'
Source	Default = ''
Dest	Default = ''
Coolant	Default = ''
Refrigerant	Default = 'R717'
FoulingFactor	Default = 1.0
CoolantMassFlow	Default = 10.0
StdDuty	Default = 100000.0

Implementation

No implementation exists for the RADSCondenser base model. Evaluation is carried out by the children of this model.

Model name

RADSECondenser

Inputs

Temperature	from refrigerant source and coolant
AbsHumidity	from coolant

Outputs

as for RADSCondenser

Comments

This model represents a evaporative condenser.

Properties

as for RADSCondenser

Methods

Evaluate	Evaluate the evaporative condenser model.

Parameters

as for RADSCondenser

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
IF MyController = NIL THEN
        loadfrac := 1.0;
        always runs full on
ELSE
        loadfrac := MyController^^.GetValue (SELF,Variable.ControlVar);
END;
IF loadfrac > 0.0 THEN
        condtemp := Source^^.GetValue (SELF,Variable.Temperature);
        airtemp := Coolant^^.GetValue (SELF,Variable.Temperature);
        airhum := Coolant^^.GetValue (SELF,Variable.AbsHumidity);
        wetbulbT := airtemp - (Water.VaporisationHeat (airtemp) *
                                        (Air.SatHumidity (airtemp) - airhum))/950.0;
                                        refer Treybal 3rd edn. pp240-241
        IF condtemp < airtemp+0.5 THEN condtemp := airtemp+0.5 END;
        tempdiff := RefSys.Bounded (condtemp-airtemp,0.5,50.0);
        corr1 := (Corr1[1] + condtemp*(Corr1[2] + condtemp*(Corr1[3] +
                condtemp*Corr1[4])) + tempdiff*(Corr1[5] +
                condtemp*Corr1[6])) * tempdiff/Corr1[7];
        IF VelMode <> RADSCondenser.Correct THEN
                corr2 := 1.0
        ELSE
                corr2 := Corr1[1] + ActualCoolFlow*(Corr1[2]
                        + ActualCoolFlow*(Corr1[3] + ActualCoolFlow*Corr1[4]))
        END;
        Heat_Load := StdDuty*corr1*corr2*FoulingFactor;
        vapenth := Refrig.VapourEnthalpy (Ref,condtemp);
        liqenth := Refrig.LiquidEnthalpy (Ref,condtemp);
        RefMassFlow := Heat_Load / (vapenth - liqenth);
        HeatFlowIn := RefMassFlow * vapenth;
        HeatFlowOut := RefMassFlow * liqenth;
ELSE
        Heat_Load := 0.0;
        HeatFlowIn := 0.0;
        HeatFlowOut := 0.0;
        RefMassFlow := 0.0;
END;
```

Model name

RADSEvaporator

Inputs

Temperature	of the application
AbsHumidity	in the application
Pressure	of refrigerant at the evaporator outlet
Heatflow	of refrigerant into the evaporator

Comments

This is the base model for the evaporator models implemented in the RADS environment as described by Comelius (1991). While there are three sorts of RADS evaporator, they have much in common, so the common evaluation is done here (initial evaluation in Eval1 and final evaluation in Eval2). The special parts of the evaluation are done by the children of this model.

Properties

Htmode	Temperature differences in the fitted polynomials are expressed as Mean or as FluidOn.
Hvmode	Correct or NoCorrect for cooled-fluid flow rate in the duty calculation.
Hhmode	Humidities are expressed either as TSRatio or RHumidity in the fitted polynomials.
Corr1	Fitted polynomial to correct the standard duty for evaporation temperature and temperature difference.
Corr2	Fitted polynomial to correct duty for total to sensible heat ratio.
Corr3	Fitted polynomial to correct the standard duty for the cooled-fluid flow rate.
Corr4	Fitted polynomial to correct fan power for cooled-fluid flow rate.
MinFaceV	Minimum face air velocity / m/s
RefrigCorr	Correction for non-standard refrigerant or operating mode.
TSRat	Actual total to sensible heat ratio of our duty
TempChange	Air temperature change across evaporator / °C
FanPower	Fan power / W
FaceVel	Face velocity / m/s
VolAirFlow	Volumetric air flow rate / m^3/s
StdDuty	Standard gross refrigeration duty / W
FaceArea	Face area / m^2
Xfactor	Fin resistance factor
PressDrop	Pressure drop through evaporator / Pa

DefInterval	Time between defrosts / s
DefLength	Length of each defrost / s
DefPower	Power released during defrost / W
Ndx	Number of modules wide for a modular evaporator.
Ndy	Number of modules deep for a modular evaporator.
Ndz	Number of modules high for a modular evaporator.
DefrostTime	Time of next defrost (on or off) event / s
SwitchTime	Time of next On/Off switch event / s
OnTime	Length of time spent switched on / s
OffTime	Length of time spent switched off /s
FanspeedCtrlr	Pointer to the fan speed controller
LiquidCtrlr	Pointer to the liquid level controller
EvapTempCtrlr	Pointer to the evaporation temperature controller
RefSrc	Pointer to the refrigerant source.
RefDst	Pointer to the refrigerant destination.
Application	Pointer to the application model.
DefrostOn	TRUE if the evaporator is currently defrosting.
SwitchedOn	TRUE if the evaporator is currently switched on.

Methods

Event	Carry out scheduled events to turn the evaporator on, off, or to start or stop defrosting.
Evaluate	Carry out the common evaluation.
EvapEval	Carry out the specialised evaluation.

Parameters

TempMode	Default = 'Mean'
VelMode	Default = 'NoCorrect'
HumMode	Default = 'RHumidity'
LiquidCtrlr	Default = ''
FanspeedCtrlr	Default = ''
EvapTempCtrlr	Default = ''
Application	Default = ''
Corr1 (Table)	Default for element i = 0.0
Corr2 (Table)	Default for element i = 0.0
Corr3 (Table)	Default for element i = 0.0
Corr4 (Table)	Default for element i = 0.0
Start	Default = 0.0
Stop	Default = 24.0
Multiple	Default = 1.0
MinFaceV	Default = 0.0
VolAirFlow	Default = 6.0
RefrigCorr	Default = 1.0
Xfactor	Default = 0.85

Refrigerant	Default = 'R717'
StdDuty	Default = 10000.0
FaceArea	Default = 10.0
FanPower	Default = 3.0
PressDrop	Default = 1.0
DefrostPower	Default = 0.0
DefrostLength	Default = 1.0
DefrostInterval	Default = 24.0
FirstDefrost	Default = MAX(LONGREAL)
NumX	Default = 1.0
NumY	Default = 1.0
NumZ	Default = 1.0
Source	Default = ''
Dest	Default = ''

Implementation

The following implementation includes the common parts of the RADS evaporator models.

```
IF Evaluated OR NOT SwitchedOn THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
```

The initial part of the evaluation

```
give the variables silly values so they get caught quickly
Pressure    := -1.0;
Ta          := -200.0;
Temperature := -200.0;
Td          := -200.0;
obj         := LinkedObjects;
Ta := RefSys.Bounded (
        Application^^.GetValue (SELF,Variable.Temperature),-50.0,50.0);
        Pressure is determined by the outlet pressure. Heat flow into the evaporator is
        determined by the heat flow from the inlet
Pressure := RefDst^^.GetValue (SELF,Variable.Pressure);
Qx := RefrHeatFlow;
RefrHeatFlow := RefSrc^^.GetValue (SELF,Variable.Heatflow);
IF Pressure < 1.0 THEN Pressure := 1.0 END;
keep safe from startup problems
Temperature := Refrig.EvapTemp (Ref,Pressure);
IF Temperature < -49.5 THEN Temperature := -49.5 END;
```

```
keep safe from startup problems
IF EvapTempCtrlr <> NIL THEN
        tmp := EvapTempCtrlr^^.GetValue (SELF,Variable.ControlVar);
        IF Temperature < tmp THEN Temperature := tmp END;
END;
Pressure := Refrig.EvapPress (Ref,Temperature);
IF FanspeedCtrlr <> NIL THEN
        Speed := FaceVel *
                        FanspeedCtrlr^^.GetValue (SELF,Variable.ControlVar);
        IF Speed < MinFaceV THEN
                Speed := MinFaceV;
        END;
ELSE
        Speed := FaceVel;
END;

count := 0;
REPEAT
        OldQx := Qx;
        Td := RefSys.Bounded (Ta - Temperature,0.0,50.0);
        EvapEval (Ta,Td,Speed,Qx,Tsrh,Powx);
        Do the calculations which are specific to the child model

        Now for the final part of the evaluation

        IF Qx < 0.0 THEN Qx := 0.0 END;
        IF LiquidCtrlr <> NIL THEN
                LiquidLevel := LiquidCtrlr^^.GetValue (SELF,Variable.ControlVar);
        ELSE
                LiquidLevel := 1.0;
        END;
        Tsurf := Temperature + (Ta - Temperature)* Xfactor;
        Hevap := Air.SatHumidity (Tsurf);
        Hum := Application^^.GetValue (SELF,Variable.AbsHumidity);
        tdiff := Ta - Tsurf;
        htcap := Air.HeatCapacity (Tsurf,0.0);
        IF tdiff > 1.0E-2 THEN
                TSRat := RefSys.Bounded (
                                1.0+(Hum-Hevap)*Water.VaporisationHeat (Tsurf) /
                                htcap / tdiff, 1.0, MAX(REAL));
        ELSE
                TSRat := 10.0;
        END;
        IF Htmode = RADSEvaporator.FluidOn THEN
```

```
        IF Hhmode = RADSEvaporator.TSRatio THEN
                Qsens := Qx/Tsrh;
                Qx := TSRat*Qsens;
                Ratio := TSRat;
                IF Ratio < 1.0 THEN Ratio := 1.0 END;
        ELSIF Hhmode = RADSEvaporator.RHumidity THEN
                Hum := Tsrh * Air.SatHumidity (Ta)/100.0;
                Tsurf := Temperature + (Ta - Temperature)*Xfactor;
                Hevap := Air.SatHumidity (Tsurf);
                tdiff := Ta - Tsurf;
                IF tdiff > 1.0E-2 THEN
                        Ratio := 1.0 +
                                (Hum - Hevap)*Water.VaporisationHeat (Tsurf)
                                / htcap / tdiff;
                        IF Ratio < 1.0 THEN Ratio := 1.0 END;
                ELSE
                        Ratio := 10.0;
                END;
                Qsens := Qx/Ratio;
                Qx := TSRat * Qsens;
        END;
ELSIF Htmode = RADSEvaporator.Mean THEN
        IF Hhmode = RADSEvaporator.RHumidity THEN
                Hum   :=
                        Tsrh*Air.SatHumidity (Temperature + Td +
                        0.5*TempChange)/100.0;
                Tsurf := Temperature + (Td + 0.5*TempChange)*Xfactor;
                Hevap := Air.SatHumidity (Tsurf);
                tdiff := Temperature + Td + 0.5*TempChange - Tsurf;
                htcap := Air.HeatCapacity (Tsurf,Hum);
                IF tdiff > 1.0E-2 THEN
                        Ratio := 1.0 + (Hum - Hevap) *
                        Water.VaporisationHeat(Tsurf)
                                        / htcap / tdiff;
                ELSE
                        Ratio := 10.0;
                END;
        ELSE
                Ratio := Tsrh;
        END;
        IF Ratio < 1.0 THEN Ratio := 1.0 END;
        Qsens := Qx/Ratio;
        Qx := TSRat*Qsens;
END;
```

```
        Qx := Qx*LiquidLevel;
        Qsens := Qsens*LiquidLevel;
        IF VolAirFlow > 0.0 THEN
                TempChange := Qsens / (VolAirFlow * Air.Density (Ta,Hum)
                        * Air.HeatCapacity (Ta,Hum));
        ELSE
                TempChange := 0.1;
        END;
        INC (count);
UNTIL (count >= 3) OR ((ABS(Qx) > 0.0)
        AND (ABS(OldQx/Qx - 1.0) < 0.0025));

WaterFlow := Qx*(TSRat-1.0)/Water.VaporisationHeat (Tsurf)/TSRat;
RoomHeatFlow := Qsens - Powx;
RefrHeatFlow := RefrHeatFlow + Qx;
RefrigFlow := RefrHeatFlow / (Refrig.VapourEnthalpy (Ref,Temperature)
                        - Refrig.LiquidEnthalpy (Ref,Temperature));
IF DefrostOn THEN
        RefrHeatFlow := RefrHeatFlow + DefPower;
        RoomHeatFlow := RoomHeatFlow - DefPower;
END;
```

Model name

RADSModEvap

Inputs

as for RADSEvaporator

Outputs

as for Refr_Room_IFace

Comments

This is an implementation of the RADS standard evaporator model.

Methods

Evaluate	Carry out calculations specific to RADSModEvap.

Implementation

The calculations specific to RADSModEvap are as follows:

```
IF Htmode = RADSEvaporator.Mean THEN
        w := Temperature;
        IF (Td <= TempChange) THEN TempChange := Td - 0.1 END;
```

```
        tmp := (Ta-w)/(Ta-TempChange-w);
        IF (tmp > 0.0) AND (ABS (tmp - 1.0) >0.01) THEN
                Td := ((Ta - w) - (Ta - TempChange - w)) / MATHLIB.Log (tmp);
        END;
END;
corr1 := (Corr1[1] + Corr1[2]*Temperature
        + Corr1[3]*Temperature*Temperature
        + Corr1[4]*Temperature*Temperature*Temperature
        + Corr1[5]*Td + Corr1[6]*Td*Temperature)*Td/Corr1[7];
Tsrh := Corr2[1] + Corr2[2]*Td + Corr2[3]*Temperature
        + Corr2[4]*Temperature*Temperature
        + Corr2[5]*Td*Temperature
        + Corr2[6]*Td*Temperature*Temperature;
IF Hvmode <> RADSEvaporator.Correct THEN
        corr2 := Ndz;
        corr3 := Ndz;
ELSE
        corr2 := (Corr3[1] + Corr3[2]*Speed + Corr3[3]*Speed*Speed
                + Corr3[4]*Ndz + Corr3[5]*Ndz*Speed
                + Corr3[6]*Ndz*Ndz)*Ndz;
        corr3 := (Corr4[1] + Corr4[2]*Speed + Corr4[3]*Speed*Speed
                + Corr4[4]*Ndz + Corr4[5]*Ndz*Speed
                + Corr4[6]*Ndz*Ndz)*Ndz *Speed*Speed*Speed;
END;
Qx := StdDuty*corr1*corr2*RefrigCorr*Ndx*Ndy;
Powx := FanPower*corr3*Ndx*Ndy;
VolAirFlow := FaceArea*Speed*Ndx*Ndy;
```

Model name

RADSNatEvap

Inputs

as for RADSEvaporator

Outputs

as for Refr_Room_IFace

Comments

This is an implementation of the RADS standard evaporator model.

Methods

Evaluate	Carry out calculations specific to RADSNatEvap.

Implementation

The calculations specific to RADSStdEvap are as follows:

```
IF Td < 0.0 THEN Td := 0.0 END;
corr1 := (Corr1[1] + Corr1[2]*Temperature
      + Corr1[3]*Temperature*Temperature
      + Corr1[4]*Temperature*Temperature*Temperature
      + Corr1[5]*Td + Corr1[6]*Td*Temperature)*Td/Corr1[7];
Tsth := Corr2[1] + Corr2[2]*Td + Corr2[3]*Temperature
      + Corr2[4]*Temperature*Temperature
      + Corr2[5]*Td*Temperature
      + Corr2[6]*Td*Temperature*Temperature;
Qx := StdDuty*corr1*RefrigCorr;
Powx := 0.0;
VolAirFlow := FaceArea*Speed;
```

Model name

RADSPipe

Inputs

Heatflow	from the active end of the line.
Temperature	from the source end of the line.
Pressure	from the destination end of the line.

Outputs

as for Pipeline

Comments

This model implements a refrigerant pipeline model as per RADS (Cornelius, 1991). The model in RADS is implicit within the simulation environment, however, while this model is explicit.

Properties

as for Pipeline with the addition of:

UA	Product of pipeline surface area and surface heat transfer coefficient / W/°C
PressDrop	Fixed pressure drop along the pipeline / Pa

Parameters

Refrigerant	Default = 'R717'
Length	Default = 10.0 / m
InsThick	Default = 0.0 / mm
Diameter	Default = 100.0 / mm

InsTCond	Default = 0.03 / W/m²K
InsByPass	Default = 1.0
PressDrop	Default = 0.0 / Pa
Source	Default = ''
Dest	Default = ''

Implementation

Refrigerant flows from one end (Source) to the other (Dest). RADSPipe is a thermal model and does not carry out detailed mass and pressure calculations so it is not capable of calculating flow directions itself. In essence, RADSPipe operates as if it contains a one-way flow valve.

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
flow can be determined at either end. Each component returns the flow that it is provides
to (+) or accepts from (-) the pipe. Vessels etc. return 0.0 and leave it up to the other end
of the pipe to decide.
flow0 := End[0]^^.GetValue (SELF,Variable.Heatflow);
flow1 := -1.0 * End[1]^^.GetValue (SELF,Variable.Heatflow);
IF flow0 = MAX(LONGREAL) THEN
the fluid source is passive
        HeatFlow := 0.0;
        IF PipeEnv <> NIL THEN
                HeatFlow := HeatFlow - UA * ((Temp[0]+Temp[1])/2.0
                                        - PipeEnv^^.GetValue
                                        (SELF,Variable.Temperature));
        END;
ELSIF flow1 = -MAX(LONGREAL) THEN
the fluid sink is passive
        HeatFlow := flow0;
        IF PipeEnv <> NIL THEN
                HeatFlow := HeatFlow - UA * ((Temp[0]+Temp[1])/2.0
                                        - PipeEnv^^.GetValue
                                        (SELF,Variable.Temperature));
        END;
ELSE
both are active
        IF flow0 > -flow1 THEN
                HeatFlow := -flow1;
        ELSE
                HeatFlow := flow0;
```

```
        END;
        IF PipeEnv <> NIL THEN
                HeatFlow := HeatFlow - UA * ((Temp[0]+Temp[1])/2.0
                                        - PipeEnv^^.GetValue
                                        (SELF,Variable.Temperature));
        END;
END;
Press[1] := End[1]^^.GetValue (SELF,Variable.Pressure) + PressEPS;
Press[0] := Press[1] + PressDrop;
Temp[0]  := End[0]^^.GetValue (SELF,Variable.Temperature);
assume saturated if carrying a valid refrigerant.  Don't adjust temperature of
non-refrigerants as the pipeline probably does not know the mass flow rate very
accurately anyway
IF Ref <> Refrig.SecondaryRefrig THEN
        Temp[1] := Refrig.EvapTemp (Ref,Press[1]);
ELSE
        Temp[1] := Temp[0];
END;
```

Model name

RADSRoom

Inputs

as for Room

Outputs

as for Room

Comments

RADSRoom is a room model of the type described by Cornelius (1991).

Methods

Evaluate	Calculate room conditions.

Implementation

RADSRoom differs from Room in that RADSRoom does not allow water vapour to condense within the room. It is therefore possible for the room relative humidity to exceed 1.0 for extended periods of time.

```
IF Evaluated THEN
        RETURN
ELSE
        Model.Evaluate;
```

```
END;
obj := LinkedObjects;
HeatFlowIn := 0.0;
the net heat flow into the room
WaterFlowIn := 0.0;
the net water vapour flow into the room
LiqFlow := 0.0;
the net water liquid flow into the room
Heat_Load := 0.0;
total heat flow into the room
Water_Load := 0.0;
total water vapour flow into the room
WHILE obj <> NIL DO
        Load := obj^.model^.GetValue (SELF,Variable.Heatflow);
        HeatFlowIn := HeatFlowIn + Load;
        IF Load > 0.0 THEN
                Heat_Load := Heat_Load + Load;
        END;
        IF obj^.model^.GetValue (SELF,Variable.State) > 0.99 THEN
        the water flow is of liquid
                LiqFlow := LiqFlow
                        + obj^.model^.GetValue (SELF,Variable.Massflow);
        ELSE
        the water flow is of vapour
                WLoad := obj^.model^.GetValue (SELF,Variable.Massflow);
                WaterFlowIn := WaterFlowIn + WLoad;
                IF WLoad > 0.0 THEN
                        Water_Load := Water_Load + WLoad;
                END;
        END;
        obj := obj^.next;
END;
temp := RefSys.Bounded (Temp.val,-50.0,50.0);
AirMass := Volume * Air.Density (temp,Hum.val);
DryAirMass := Volume * Air.Density (temp,0.0);
SatHum := Air.SatHumidity (temp);
RelHum := Hum.val / SatHum;

Now calculate the derivatives of the dynamic variables

Hum.der := WaterFlowIn / DryAirMass;
Humidity changes depending upon the water flowing into the room
Temp.der := HeatFlowIn / (AirMass * Air.HeatCapacity (temp,Hum.val));
Temperature changes due to heat flow into the room.
```

Model name

RADSStdEvap

Inputs

as for RADSEvaporator

Outputs

as for Refr_Room_IFace

Comments

This is an implementation of the RADS standard evaporator model.

Methods

Evaluate	Carry out calculations specific to RADSStdEvap.

Implementation

The calculations specific to RADSStdEvap are as follows:

```
IF Htmode = RADSEvaporator.Mean THEN
        w := Temperature;
        IF (Td <= TempChange) THEN TempChange := Td - 0.1 END;
        tmp := (Ta-w)/(Ta-TempChange-w);
        IF (tmp > 0.0) AND (ABS (tmp - 1.0) > 0.01) THEN
                Td := ((Ta - w) - (Ta - TempChange - w)) / MATHLIB.Log (tmp);
        END;
END;
corr1 := (Corr1[1] + Corr1[2]*Temperature
        + Corr1[3]*Temperature*Temperature
        + Corr1[4]*Temperature*Temperature*Temperature
        + Corr1[5]*Td + Corr1[6]*Td*Temperature)*Td/Corr1[7];
Tsrh := Corr2[1] + Corr2[2]*Td + Corr2[3]*Temperature
        + Corr2[4]*Temperature*Temperature
        + Corr2[5]*Td*Temperature
        + Corr2[6]*Td*Temperature*Temperature;
IF Hvmode <> RADSEvaporator.Correct THEN
        corr2 := 1.0;
        corr3 := 1.0;
ELSE
        corr2 := Corr3[1] + Corr3[2]*Speed + Corr3[3]*Speed*Speed
                + Corr3[4]*Speed*Speed*Speed;
        corr3 := Corr4[1] + Corr4[2]*Speed + Corr4[3]*Speed*Speed
                + Corr4[4]*Speed*Speed*Speed;
```

```
END;
Qx := StdDuty*corr1*corr2*RefrigCorr;
Powx := FanPower*corr3;
VolAirFlow := FaceArea*Speed;
```

Model name

RADSTyp4Control

Inputs

InputType	Variable type to be controlled.
ControlVar	Set point from a superior controller.

Outputs

ControlVar	Desired loading fraction

Comments

This model implements the RADS type 4 compressor control strategy as described by Cornelius (1991). This controller controls a number of compressors which have a set order of operation. One compressor at a time is partly loaded. When that compressor is fully loaded, the next compressor is started on minimum loading. When the partly loaded compressor reaches its minimum loading, it is turned off, and the previous compressor in order commences unloading.

Properties

Comp	An array containing pointers to up to 20 compressors and their control parameters.
NumCompressors	The number of compressors to be controlled.
CurrentPartLoad	The number of the compressor which is partly-loaded.
Source	Pointer to the model whose variable is controlled.
SetPtCtrlr	Pointer to the controller which provides a set point for this one.
InputType	Type of variable to be controlled.
SetPoint	Current value of the controller set point.
MinOutput	Minimum value of the controller output variable.
MaxOutput	Maximum value of the controller output variable.
PrevErr	Value of input - set point at last time step.
NumComp	The number of the compressor which is partly-loaded (as a LONGREAL).
LoadFrac	The control output for the partly-loaded compressor.

Methods

GetValue	Return the control output for the requesting compressor.
Evaluate	Calculate the outputs for all compressors.

Initialise	Initialise the RADSTyp4Control model.

Parameters

Parameters (Table)	Default for element 3*i-2 = ''
Parameters (Table)	Default for element 3*i-1 = 0.0
Parameters (Table)	Default for element 3*i = 0.0
InitialNoOn	Initial number of compressors in operation. Default = 1.0
MaxOutput	Default = 1.0
MinOutput	Default = 0.0
SetPoint	Default = -10.0
Control	Type of variable to control. Default = 'Temperature'
SetPtCtrlr	Default = ''
ControlledModel	Default = ''

Implementation

The table Parameters contains three data items per entry: Compressor name, Proportional control factor and Integral time. If Integral time = 0.0 then the control action for that compressor is differential instead of velocity-form Proportional-Integral and the second data item in the entry then represents a differential around the set point. If the name of a set point controller is provided, then the set point is obtained from that model at each evaluation, otherwise the fixed SetPoint is used. A model name must be provided for ControlledModel.

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
input := Source^^.GetValue (SELF,InputType);
IF SetPtCtrlr <> NIL THEN
        setpoint := SetPtCtrlr^^.GetValue (SELF,Variable.ControlVar);
ELSE
        setpoint := SetPoint;
END;
lasttime := Time.System.timeval.valstart;
curtime := Time.System.timeval.val;
timediff := curtime - lasttime;
IF Comp[CurrentPartLoad].Diff THEN
it is a differential controller. Note the signs because turning a compressor on reduces the
temperature
        IF curtime > (ChangeTime+Delay) THEN
                IF timediff < 1.0E-4 THEN
                start of a time-step, so we can change configuration
                        IF input < setpoint - 0.5*Comp[CurrentPartLoad].Differential
```

```
                        THEN
                    Comp[CurrentPartLoad].Output := MinOutput;
                    DEC (CurrentPartLoad);
                    ChangeTime := curtime;
                ELSIF input > setpoint
                        + 0.5*Comp[CurrentPartLoad].Differential THEN
                    Comp[CurrentPartLoad].Output := MaxOutput;
                    INC (CurrentPartLoad);
                    ChangeTime := curtime;
                ELSE
                happy with current configuration
                END;
            END;
        END;
ELSE
it is a PI controller
        err := input - setpoint;
        IF timediff < 1.0E-4 THEN
        take no control action here, but may change compressor configuration
            PrevErr := err;
            Comp[CurrentPartLoad].OutputVar.der := 0.0;
            Comp[CurrentPartLoad].Output :=
                Comp[CurrentPartLoad].OutputVar.val;
                Check if the compressor configuration needs changing
            IF Comp[CurrentPartLoad].Output <= MinOutput THEN
                Comp[CurrentPartLoad].Output := MinOutput;
                DEC (CurrentPartLoad);
            ELSIF Comp[CurrentPartLoad].Output >= MaxOutput THEN
                Comp[CurrentPartLoad].Output := MaxOutput;
                INC (CurrentPartLoad);
            END;
        ELSE
            Comp[CurrentPartLoad].OutputVar.der :=
                (Comp[CurrentPartLoad].P * (1.0 + timediff/
                Comp[CurrentPartLoad].I)*err
                - Comp[CurrentPartLoad].P*PrevErr) / timediff;
            Comp[CurrentPartLoad].Output :=
                Comp[CurrentPartLoad].OutputVar.val;
            IF Comp[CurrentPartLoad].Output < MinOutput THEN
                Comp[CurrentPartLoad].Output := MinOutput;
                IF Comp[CurrentPartLoad].OutputVar.der < 0.0 THEN
                    Comp[CurrentPartLoad].OutputVar.der := 0.0;
                END;
            ELSIF Comp[CurrentPartLoad].Output > MaxOutput THEN
```

```
                    Comp[CurrentPartLoad].Output := MaxOutput;
                    IF Comp[CurrentPartLoad].OutputVar.der > 0.0 THEN
                        Comp[CurrentPartLoad].OutputVar.der := 0.0;
                    END;
                END;
        END;
END;
IF CurrentPartLoad < 1 THEN
there must always be one compressor on part-load duty
        CurrentPartLoad := 1
ELSIF CurrentPartLoad > NumCompressors THEN
do not run more than the available number of compressors
        CurrentPartLoad := NumCompressors
END;
FOR i := 1 TO CurrentPartLoad-1 DO
make sure the working compressors are set properly
        IF NOT Comp[i].Diff THEN
            Comp[i].OutputVar.der := 0.0;
        END;
        Comp[i].Output := 1.0;
END;
NumComp := LONGREAL (CurrentPartLoad);
LoadFrac := Comp[CurrentPartLoad].Output;
```

Model name

RADSVessel

Inputs

Heatflow	into the vessel from each linked model.
State	of each stream flowing into the vessel.

Outputs

Heatflow	of liquid refrigerant into the vessel to the liquid source.
Massflow	of liquid refrigerant into the vessel to the liquid source.
Temperature	of refrigerant in the vessel.
Pressure	of refrigerant in the vessel.

Comments

This model corresponds to the RADS vessel model described by Cornelius (1991).

Properties

LiquidSource	Pointer to the source of make-up liquid refrigerant.
Heat_Load	The total positive heat load on the vessel / W

ReqdLiquid	The mass flow rate of liquid required to balance the vapour flow to the compressors / kg/s
LiquidHeatFlow	Enthalpy flow of make-up liquid / W
Pressure	Refrigerant pressure / Pa
RefrigMass	Mass of refrigerant in the vessel / Pa
Temp	Dynamic variable representing the refrigerant temperature / °C

Methods

GetValue	Return Heatflow, Massflow, Temperature or Pressure.
Evaluate	Calculate the conditions in the vessel.
Initialise	Initialise this instance of RADSVessel.

Parameters

Temp	Default = -20.0
RefrigMass	Default = 0.0
Refrigerant	Default = 'R717'
LiquidSource	Default = ''

Implementation

This model is thermal in nature, and it is assumed that there is a constant mass of refrigerant in the vessel at all times.

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
obj := LinkedObjects;
HeatFlowIn := 0.0;
net of all loads (positive and negative)
ReqdLiquid := 0.0;
make-up required for mass balance
Heat_Load := 0.0;
sum of positive loads
temp := RefSys.Bounded (Temp.val,-49.9,49.9);
do not allow the vessel temperature to go outside the range of refrigerant property
routines
WHILE obj <> NIL DO
        IF obj^.model <> LiquidSource^ THEN
                Load := obj^.model^.GetValue (SELF,Variable.Heatflow);
                IF Load < 0.0 THEN
                        ReqdLiquid := ReqdLiquid -
                                obj^.model^.GetValue (SELF,Variable.Massflow);
```

```
            ELSE
                Heat_Load := Heat_Load + Load;
            END;
            HeatFlowIn := HeatFlowIn + Load;
        END;
        obj := obj^.next;
END;
IF ReqdLiquid < 0.0 THEN
        ReqdLiquid := 0.0;
END;
LiquidHeatFlow := ReqdLiquid
        * Refrig.LiquidEnthalpy (Ref,
        LiquidSource^^.GetValue (SELF,Variable.Temperature));
HeatFlowIn := HeatFlowIn + LiquidHeatFlow;
Temp.der := HeatFlowIn
        / (RefrigMass * Refrig.LiquidSpecHt (Ref,temp));
Pressure := Refrig.EvapPress (Ref,temp);
```

Model name

RADSWCondenser

Inputs

Temperature	from refrigerant source and coolant

Outputs

as for RADSCondenser

Comments

This model represents a water-cooled condenser.

Properties

as for RADSCondenser

Methods

Evaluate	Evaluate the water-cooled condenser model.

Parameters

as for RADSCondenser

Implementation

```
IF Evaluated THEN
```

```
        RETURN;
ELSE
        Model.Evaluate;
END;
IF MyController = NIL THEN
        loadfrac := 1.0;
        always runs full on
ELSE
        loadfrac := MyController^^.GetValue (SELF,Variable.ControlVar);
END;
IF loadfrac > 0.0 THEN
        condtemp := Source^^.GetValue (SELF,Variable.Temperature);
        watertemp := Coolant^^.GetValue (SELF,Variable.Temperature);
        ActualCoolFlow := CoolantFlow*loadfrac;
        IF condtemp < watertemp THEN
                condtemp := watertemp;
                tempdiff := 0.0;
        ELSE
                IF condtemp < watertemp+0.5 THEN condtemp := watertemp+0.5 END;
                tempdiff := condtemp-watertemp;
                IF TempMode = RADSCondenser.Mean THEN
                        w := RefSys.Bounded (condtemp-watertemp-TempChg,0.001,50.0);
                        tmp := MATHLIB.Log ((condtemp - watertemp)/w);
                        IF ABS(tmp) > 0.01 THEN
                                tempdiff := ((condtemp - watertemp) - w) / tmp;
                        END;
                END;
                tempdiff := RefSys.Bounded (tempdiff,0.5,50.0);
        END;
        corr1 := (Corr1[1] + condtemp*(Corr1[2] + condtemp*(Corr1[3] +
                condtemp*Corr1[4])) + tempdiff*(Corr1[5] +
                condtemp*Corr1[6])) * tempdiff/Corr1[7];
        IF VelMode <> RADSCondenser.Correct THEN
                corr2 := 1.0
        ELSE
                corr2 := Corr1[1] + ActualCoolFlow*(Corr1[2]
                        + ActualCoolFlow*(Corr1[3] + ActualCoolFlow*Corr1[4]))
        END;
        Heat_Load := StdDuty*corr1*corr2*FoulingFactor;
        TempChg := Heat_Load /
                (ActualCoolFlow*Water.HeatCapacity (watertemp));
        TempChg := RefSys.Bounded (TempChg,0.1,condtemp-watertemp-0.1);
        vapenth := Refrig.VapourEnthalpy (Ref,condtemp);
        liqenth := Refrig.LiquidEnthalpy (Ref,condtemp);
```

```
        RefMassFlow := Heat_Load / (vapenth - liqenth);
        HeatFlowIn := RefMassFlow * vapenth;
        HeatFlowOut := RefMassFlow * liqenth;
ELSE
        Heat_Load := 0.0;
        RefMassFlow := 0.0;
        TempChg := 0.1;
        HeatFlowIn := 0.0;
        HeatFlowOut := 0.0;
END;
```

Model name

Refr_Room_IFace

Inputs

None	abstract model type

Outputs

Heatflow	to refrigeration system or room
Massflow	to refrigeration system or room
State	of Massflow
Pressure	of refrigerant
Temperature	of refrigerant

Comments

This is the base model type for interfaces between the refrigeration system and the refrigeration application. GetValue is implemented by Refr_Room_IFace and need not be re-defined by descendent models.

Properties

RoomHeatFlow	Heat flow from the room / W
WaterFlow	Water mass flow from the room / kg/s
RefrHeatFlow	Heat flow into the refrigeration system / W
RefrigFlow	Refrigerant mass flow through the model / kg/s
Pressure	Refrigerant pressure / Pa
Temperature	Refrigerant temperature / °C
State	Refrigerant liquid fraction in the model.
Multiple	Number of items represented by this model.

Methods

GetValue	Return Heatflow, Massflow, State, Pressure or Temperature.

Model name
RefrigComp

Inputs

None	abstract model type

Outputs

None	abstract model type

Comments
This model is the base class for all refrigeration system components.

Properties

Ref	The number of the refrigerant contained by this component.

Methods
None

Model name
Model

Inputs
None (abstract model type)

Outputs
None (abstract model type)

Comments
Model serves as a base class for all other models in the hierarchy and so its properties are inherited by all other models. Model provides a template for the minimum level of functionality required of any model (that is, the methods GetValue, Event, Evaluate and Initialise).

Properties

name	Character array to store the model instance name. Up to 15 characters.
LinkedObjects	List of models to which this model is linked.

Methods

GetValue	Calling this method causes a runtime error.
Event	Calling this method causes a runtime error.

NewEvaluation	Sets Evaluated = FALSE ready for a new evaluation.
Evaluate	Provides the default evaluation action, i.e. sets Evaluated = TRUE
Initialise	Calling this method causes a runtime error.

Parameters

None (abstract model type).

Implementation

The implementation of Model provides the default null action.

Model name

Room

Inputs

Heatflow	into the Room
Massflow	of liquid water or water vapour into the Room

Outputs

Temperature	of air in the room
AbsHumidity	of air in the room
RelHumidity	of air in the room

Comments

This model defines a room filled with humid air.

Properties

Hum	Dynamic variable for absolute humidity
Temp	Dynamic variable for air temperature / °C
WaterMass	Dynamic variable for mass of condensed water in the room / kg
RelHum	Relative humidity
Volume	Volume of air in the room / m^3
Heat_Load	Net positive heat load on room / W
Water_Load	Net positive water flow into room / kg/s
TimeConstant	Ratio of volume to flow in the room / s

Methods

GetValue	Valid requests are Temperature, AbsHumidity, RelHumidity
Evaluate	Evaluate the Room model
Initialise	Initialise the Room model

Parameters

RelHum	Default = 0.8
Volume	Default = 100.0
Temp	Default = -20.0
WaterMass	Default = 0.0
TimeConstant	Default = 300.0

Implementation

This follows the description given in Chapter 7.

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
obj := LinkedObjects;
HeatFlowIn := 0.0;
the net heat flow into the room
WaterFlowIn := 0.0;
the net water vapour flow into the room
LiqFlow := 0.0;
the net water liquid flow into the room
Heat_Load := 0.0;
total heat flow into the room
Water_Load := 0.0;
total water vapour flow into the room
WHILE obj <> NIL DO
        Load := obj^.model^.GetValue (SELF,Variable.Heatflow);
        HeatFlowIn := HeatFlowIn + Load;
        IF Load > 0.0 THEN
                Heat_Load := Heat_Load + Load;
        END;
        IF obj^.model^.GetValue (SELF,Variable.State) > 0.99 THEN
        the water flow is of liquid
                LiqFlow := LiqFlow
                        + obj^.model^.GetValue (SELF,Variable.Massflow);
        ELSE
        the water flow is of vapour
                WLoad := obj^.model^.GetValue (SELF,Variable.Massflow);
                WaterFlowIn := WaterFlowIn + WLoad;
                IF WLoad > 0.0 THEN
                        Water_Load := Water_Load + WLoad;
                END;
        END;
```

```
        obj := obj^.next;
END;
temp := RefSys.Bounded (Temp.val,-50.0,50.0);
AirMass := Volume * Air.Density (temp,Hum.val);
DryAirMass := Volume * Air.Density (temp,0.0);
SatHum := Air.SatHumidity (temp);
RelHum := Hum.val / SatHum;
IF RelHum < 0.0 THEN RelHum := 0.0 END;
Protect against startup transients

Now calculate the derivatives of the dynamic variables

WaterMassFlow := (RelHum - 1.0) * SatHum * DryAirMass/TimeConstant;
WaterMass is the mass of water in the room as frost, drip, or fog. TimeConstant will
depend on air movement rates in the room.
IF (WaterMass.val < 0.0) AND (WaterMassFlow < 0.0) THEN
if the WaterMass.val ever becomes negative, don't let it get worse.
        WaterMassFlow := 0.0;
END;

Load := WaterMassFlow * Water.VaporisationHeat (temp);
IF Load > 0.0 THEN
        Heat_Load := Heat_Load + Load;
END;

HeatFlowIn := HeatFlowIn + Load;
HeatFlowIn is sensible heat flow into the room. Can come from outside, objects inside the
room, or conversion from latent heat of water vapour

Hum.der := (WaterFlowIn - WaterMassFlow) / DryAirMass;
Humidity changes depending upon the water flowing into the room and the amount of
water in liquid or solid form in the room.

WaterMass.der := LiqFlow + WaterMassFlow;
IF (WaterMass.val <= 0.0) AND (WaterMass.der <= 0.0) THEN
                WaterMass.der := 0.0;
END;

Temp.der := HeatFlowIn / (AirMass * Air.HeatCapacity (temp,Hum.val)
        + WaterMass.val * Water.HeatCapacity (temp));
        Temperature changes due to heat flow into the room. Any WaterMass in the room
        provides a buffering effect
```

Model name

Room_Room_IFace

Inputs

None	abstract model type

Outputs

Heatflow	through the interface
Massflow	through the interface

Comments

Room_Room_IFace is the base model for all interfaces between room models. The convention followed is that flows from the first linked model to the second linked model are positive. This doesn't matter for users of the GetValue method as GetValue sets signs appropriately depending on the requesting model, but it does matter for child versions of Evaluate.

Properties

HeatFlow	Heat flow through the interface / W
WaterFlow	Water vapour flow through the interface / kg/s

Methods

GetValue	Return Heatflow or Massflow to requesting models.

Model name

SISOController

Inputs

InputType	Variable to be controlled.
ControlVar	Set point from a superior controller.

Outputs

ControlVar	Output to manipulated model.

Comments

This is the base model for Single Input, Single Output controllers. It provides the Input method by which all of its descendents obtain both of their input variables.

Properties

Source	Pointer to the model from which the controlled variable is obtained.

SetPtCtrlr	Pointer to the superior controller which provides the set point.
ControlOutput	Value of output calculated for the controller.
SetPoint	Current value of the controller set point.
MinOutput	Minimum controller output value.
MaxOutput	Maximum controller output value.
InputType	Type of the input variable.

Methods

GetValue	Return ControlVar to requesting models.
Input	Obtain the current input value (and setpoint if there is a superior controller).

Parameters

None	abstract model type.

Implementation

The only evaluation performed by this model is reading input in the Input method.

```
input := Source^^.GetValue (SELF,InputType);
IF SetPtCtrlr <> NIL THEN
        setpoint := SetPtCtrlr^^.GetValue (SELF,Variable.ControlVar);
ELSE
        setpoint := SetPoint;
END;
```

Model name

SCDiffControl

Inputs

as for SISOController

Outputs

as for SISOController

Comments

This is a basic on/off controller. The controller output is defined by: Output = MinOutput if Input > SetPoint + 0.5*Differential; Output = MaxOutput if Input < SetPoint - 0.5*Differential; no change to current value if SetPoint + 0.5*Differential > Input > SetPoint - 0.5*Differential.

Properties

as for SISOController with the addition of:

Differential	Dead band around the setpoint where no output change occurs.

Methods

Evaluate	Calculate the current controller output.
Initialise	Initialise the SODiffControl model.

Parameters

SetPoint	Default = -10.0
MaxOutput	Default = 1.0
MinOutput	Default = 0.0
Differential	Default = 2.0
SetPtCtrlr	Default = ''
ControlledModel	Default = ''
Control	Default = 'Temperature'

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
IF Status = Waiting THEN
waiting for the end of the Delay period so stop now
        RETURN;
END;
Input (in,setpoint);
curtime := Time.System.GetValue (SELF,Variable.Time);
IF (in < setpoint - 0.5*Differential) AND (Status <> Maximum) THEN
        Scheduler.AddStateEvent (SELF);
        EventScheduled := TRUE;
ELSIF (in > setpoint + 0.5*Differential) AND (Status <> Minimum) THEN
        Scheduler.AddStateEvent (SELF);
        EventScheduled := TRUE;
END;
```

Model name

SOPIDController

Inputs

as for SISOController

Outputs

as for SISOController

Comments

This class models a position form Single Input Single Output PID controller. The output is dimensionless and in the range MinOutput -> MaxOutput. This is important to know when setting the P parameter. The output of the controller is described by the equation: Output = P (Error + ErrorIntegral/I + D d(Input)/dt) + Bias; where Error = Input - Setpoint; P, I, D, Bias and Setpoint are input parameters.

Properties

Integral	Dynamic variable which describes the accumulated integral action.
P	Proportional control factor (units of OutputVariableTime/InputType)
I	Integral time / s
D	Derivative time / s
Bias	Control action when there is no error (units the same as the output variable).
Derivative	Input derivative value during the last time step (units of InputType/s).
StoredInput	Input value during the last time step (units of InputType).

Methods

Evaluate	Calculate control action.
Initialise	Initialise the SOPIDControl model.

Parameters

P	Default = 0.1
I	Default = 300.0
D	Default = 0.0
MaxOutput	Default = 1.0
MinOutput	Default = 0.0
SetPoint	Default = -10.0
Bias	Default = 0.0
Control	Default = 'Temperature'
SetPtCtrlr	Default = ''
ControlledModel	Default = ''

Implementation

Proper implementation of derivative action would require the evaluation of an integral equation in parallel with the differential equations, with allowances for the

possibility of backtracking and discrete events. This has not been done, so the derivative action is not of high quality.

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
Input (in,setpoint);
err := in - setpoint;
IF (ABS(Integral.val) > 20.0*ABS(err)) AND (Integral.val*err > 0.0)
        THEN
        do not allow the Integral component of the action to get very big relative to the
        proportional component if the error has the same sign as the Integral
        Integral.der := 0.0;
ELSE
        Integral.der := err/I;
END;
lasttime := Time.System.timeval.valstart;
curtime := Time.System.timeval.val;
IF lasttime = curtime THEN
        deriv := Derivative;
        StoredInput := in;
ELSE
        deriv := (in - StoredInput) / (curtime - lasttime);
        Derivative := deriv;
END;
IF ABS(D*deriv) > 5.0*ABS(err) THEN
don't let the derivative component get too big.
        IF deriv*err < 0.0 THEN
                deriv := -5.0*err/D;
        ELSE
                deriv := 5.0*err/D;
        END;
END;
ControlOutput := P * (err + Integral.val + D*deriv) + Bias;
IF ControlOutput < MinOutput THEN
        ControlOutput := MinOutput;
ELSIF ControlOutput > MaxOutput THEN
        ControlOutput := MaxOutput;
END;
```

Model name

ThermalObject

Inputs

Temperature	of Room
AbsHumidity	of Room

Outputs

Temperature	mass average
Heatflow	from "Multiple" ThermalObjects
Massflow	of water vapour

Comments

ThermalObject models any solid in the refrigeration application which may change temperature, including food product and structural material. A ThermalObject may be moved from room to room according to a set schedule. The heat load model is that described in Chapter 3. Two important cases which this implementation does not handle well are thawing of any substance (it treats thawing as if there was no phase change, although a correct enthalpy balance is maintained) and freezing of liquids (convection within the liquid is not considered).

Properties

x	Dynamic variable for unfrozen depth / m
H	Dynamic variable for the total enthalpy of a single ThermalObject / J
Room	Pointer to the room model in which the ThermalObject rests
NextRoom	Pointer to the room model to which the ThermalObject will move next
mode	material mode: soft, freeze_thaw, or hard
N	Heat load shape factor
E	Equivalent heat transfer dimensionality shape factor
h	Surface heat transfer coefficient / W/m^2K
Tf	Initial freezing temperature of the material / °C
X	Critical thickness / m
A	Surface area / m^2
V	Volume / m^3
Cl	Unfrozen specific heat capacity / J/kg K
Cs	Frozen specific heat capacity / J/kg K at temperature Tbase
kl	Unfrozen thermal conductivity / W/m K
ks	Frozen thermal conductivity / W/m K
L	Latent heat of freezing / J/kg
Tstart	Initial mass average temperature of the ThermalObject / °C
Tma	Mass average temperature of the ThermalObject / °C
Hstart	Initial total enthalpy of the ThermalObject / J

Hf	Enthalpy of the ThermalObject / J/kg at temperature Tf
Hfreeze	Sum of L and any residual sensible heat / J/kg
a	Parameter in the enthalpy function
b	Parameter in the enthalpy function
c	Parameter in the enthalpy function
B2k	Product of Beta squared and the thermal conductivity at Tma
Bi	Biot number
Tff	Freezing temperature of the underlying material / °C
Tbase	Temperature / °C at which enthalpy is equal to Hbase
Hbase	Base enthalpy / J/kg (e.g. 0 J/kg at -40°C)
Multiple	Number of individual items represented by this ThermalObject
RespA	Intercept of the respiration equation / W
RespB	Slope of the respiration equation / W/°C
VapDiffuRes	Vapour diffusion resistance for weight loss
MassTrCoef	Mass transfer coefficient for weight loss
WtLoss	Rate of weight loss / kg/s
NextRepeatTime	Next time at which conditions will be reset to Initial / s
RepeatInterval	Interval between resets to Intial conditions / s

Methods

GetValue	Valid requests are for Temperature, Heatflow, Massflow.
Event	Move the ThermalObject to its next room, or re-initialise.
Evaluate	Evaluate the ThermalObject model.
Initialise	Initialise the ThermalObject model.

Parameters

E	Default = 1.0
N	Default = 1.0
h	Default = 10.0
kl	Default = 0.5
ks	Default = 1.5
X	Default = 0.1
A	Default = 2.8
V	Default = 0.2
Cl	Default = 4.0E6
Cs	Default = 2.0E6
Tf	Default = -0.9
Tstart	Default = 10.0
Hf	Default = 2.8E8
Tff	Default = 0.0
Hbase	Default = 0.0
Tbase	Default = -40.0
Multiple	Default = 1.0

RepeatInterval	Default = 1.0E10
VapDiffuRes	Default = 0.0
MassTrCoef	Default = 0.0
Load (Table)	Default for element 1 = 0.0
Load (Table)	Default for element i = MAX(REAL)
Respiration (Table)	Default for element 1 = 0.0
Respiration (Table)	Default for element 2 = 0.0
Respiration (Table)	Default for element 3 = 100.0
Respiration (Table)	Default for element 4 = 0.0

Implementation

The ThermalObject implementation follows that described in Chapter 4, with additions for respiration and weight loss

```
IF Evaluated THEN
        RETURN
ELSE
        Model.Evaluate;
END;
IF Room = NIL THEN
        H.der := 0.0;
        x.der := 0.0;
        RETURN
END;
Get the room temperature and humidity
Tamb := RefSys.Bounded (
        Room^.model^.GetValue (SELF,Variable.Temperature),-50.0,50.0);
Hum  := Room^.model^.GetValue (SELF,Variable.AbsHumidity);
IF H.val > Hf THEN
above the freezing point enthalpy
        Tma := (H.val - Hf) / (Cl*V) + Tf;
        linear function right for pre-cooling phase
ELSE
        Tma := ((H.val-a) - Sqrt((H.val-a)*(H.val-a) - 4.0*b*c))
                        / (2.0*b) + Tff;
                        hyperbolic function for freezing and sub-cooling
END;
IF mode = soft THEN
recalculate the current value of Hfreeze, the enthalpy released during freezing
        IF (Tma > Tf) THEN
                Hfreeze := L + (Tma - Tf)*Cl
        ELSE
                Hfreeze := L
        END;
```

```
END;
IF x.val > 0.0 THEN
the freezing front is not yet at the centre
        IF Tma < Tf THEN
        at temperatures under Tf, allow the temperature to drift downwards as the process
        runs
                Temp := Tma;
        ELSE
        at temperatures over Tf, it is not sensible to run the freezing calculation at all, so
        use Tf
                Temp := Tf;
        END;
        IF Tamb < Tf THEN
        air temperature below freezing temperature -- possibility of freezing front
        movement
                dxfreeze := -(Temp - Tamb) /
                                (Hfreeze * (Pow(x.val,E-1.0)) * (1.0/(h*(Pow(X,E-1.0)))
                                + ((Pow(x.val,2.0-E)) - (Pow(X,2.0-E))) / (ks * (E-2.0))));
                                freezing front movement rate

                dV := N * (Pow(x.val/X,N - 1.0)) * (V/X);
                change in volume with radius

                Qfreeze := dV * dxfreeze * Hfreeze;
                heat release rate
        ELSE
                dxfreeze := 0.0;
                dV := 0.0;
                Qfreeze := 0.0;
        END;
ELSE
avoid the undefined result when x =< 0.0
        dxfreeze := 0.0;
        Qfreeze := 0.0;
END;
Qchill := -(E/3.0) * (V*B2k/(X*X)) * (Tma - Tamb);

IF dxfreeze < 0.0 THEN
chilling and freezing case
        CASE mode OF
        soft:
                H.der := Qchill;
                x.der := 0.0;
                IF (Qchill/Qfreeze < 0.99) OR (Tma < Tf) THEN
```

```
            transition if Qfreeze is significantly less than Qchill, or the mass average
            temp is less than the freezing temp (catches pathological cases)
                    Bi := h*X/ks;
                    beta := CalcBeta (Bi);
                    B2k := beta*beta*ks;
                    mode := freeze_thaw;
                    H.der := Qfreeze;
                    x.der := dxfreeze;
            ENDI
        freeze_thaw:
            H.der := Qfreeze;
            x.der := dxfreeze;
            IF (x.val > 0.0) THEN
                    IF (Qfreeze/Qchill < 0.99) AND (Pow(x.val/X,E) < trans_frac)
                            THEN
                            Transition if Qchill is significantly less than Qfreeze, and
                            ensure that enough freezing has been done
                            Correct B2k to give a smooth transition.
                            B2k := (Qfreeze/Qchill)*B2k;
                    END;
                    mode := hard;
                    H.der := Qfreeze;
            ELSE
                    mode := hard;
                    H.der := Qfreeze;
            ENDI
        hard:
            H.der := Qchill;
            IF x.val > 0.0 THEN
            arrange things so that the freezing front finishes off in a roughly correct
            manner
                    x.der := dxfreeze * Qchill/Qfreeze;
            ELSE
                    x.der := 0.0;
            END;
        END;
ELSE
warming and thawing case - shouldn't get here, but it doesn't fall over if we do.
        H.der := Qchill;
        x.der := 0.0;
END;
now add any respiration heat load
RespHt := RespA + RespB*Tma;
IF (RespHt > 0.0) AND (Tma > Tf) THEN
```

```
        H.der := H.der + RespHt*V;
END;
and calculate weight loss, using the model of Pham (1984), J Food Sci Vol 49,
p1275-1281,1294
WtLoss := MassTrCoef*A*(Air.SatHumidity (Tamb) - Hum) /
                        (1.0 + Exp (0.085*Tamb) + MassTrCoef*VapDiffuRes);
IF WtLoss < 0.0 THEN WtLoss := 0.0 END;
```

Model name

Time

Inputs

None

Outputs

Time	as simulated

Comments

This model provides a clock which simulates the passage of time during the simulation. Time only has one instance, and it is the only model which must exist in any simulation. If a user wishes an additional time model, then the UserTime class should be used.

Properties

timeval	Dynamic variable representing the current time / s

Methods

GetValue	Return the current value of the system Time.
Initialise	Initialise this instance of the time model.

Implementation

The derivative of timeval is the constant value 1.0, so no evaluation is necessary.

Model name

TimedOutput

Inputs

None

Outputs

as for SISOController.

Comments

This model provides output which varies linearly with time between specified values.

Properties

as for SISOController, with the addition of:

OutputValue	Array containing time values at element i and output values at element i+1.
Last	Index to the previous specified output (time,value) pair.
Cur	Index to the next specified output (time,value) pair.
Intercept	Intercept of the line fitted between Last and Cur at the most recent Event.
Slope	Slope of the line fitted between Last and Cur at the most recent Event.
RepeatLength	Length of time between repeats of the output / s .

Methods

Evaluate	Calculate the current output value.
Event	Advance Last and Cur, and fit a line between the new values.
Initialise	Initialise the TimedOutput model instance.

Parameters

Output (Table)	Default for element i = 0.0
RepeatLength	Default = 24.0 / hrs
Units	Default = 'not_defined'

Implementation

The table Output may contain up to 16 pairs of (time,value) elements.

```
IF Evaluated THEN
    RETURN;
ELSE
    Model.Evaluate;
END;
ControlOutput := Intercept
                    + Slope*Time.System.GetValue
                    (SELF,Variable.Time);
```

Model name

UserTime

Inputs

None

Outputs

Time	user defined time variable.

Comments

UserTime differs from Time only in that the derivative of timeval can be chosen by the user.

Properties

as for Time, with the addition of:

Ratio	Ratio of the UserTime time derivative to the system time derivative.

Methods

Evaluate	Calculate the time value.
Initialise	Initialise this instance of the UserTime model.

Parameters

Start	Initial value of the UserTime variable. Default = 0.0
Ratio	Default = 1.0
Units	Default = 'not_defined'

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
timeval.der := Ratio;
```

Model name

VelPIControl

Inputs

as for SISOControiler

Outputs

as for SISOController

Comments

This class models a velocity form Single Input Single Output PI controller.

Properties

P	Proportional factor (units OutputVariableType/InputType).
I	Integral time / s
PrevErr	Error value at the previous time step (units as for the input variable type).
OutputVar	Dynamic variable describing the current output value of the controller.

Methods

GetValue	GetValue - Return ControlVar to requesting models.
Evaluate	Calculate the controller output derivative.
Initialise	Initialise the VelPIControl model.

Parameters

P	Default = 0.1
I	Default = 300.0
MaxOutput	Default = 1.0
MinOutput	Default = 0.0
InitialOutput	Default = 0.0
SetPoint	Default = -10.0
Control	Default = 'Temperature'
SetPtCtrlr	Default = ''
ControlledModel	Default = ''

Implementation

This model suffers from problems similar to SOPIDControl. Accurate representation of this model would require the solution of an integral equation in parallel with the ODE solver, and a VelPIDControl model (should one be written) would require the solution of a double integral equation. Since an integral equation solver is not available within RefSim, the proportional action of this model is represented approximately.

```
IF Evaluated THEN
    RETURN;
ELSE
    Model.Evaluate;
END;
Input (in,setpoint);
```

```
err := in - setpoint;
lasttime := Time.System.timeval.valstart;
curtime := Time.System.timeval.val;
timediff := curtime - lasttime;
IF timediff <= 0.0 THEN
take no control action here
        PrevErr := err;
ELSE
        OutputVar.der := P*(err/I + (err - PrevErr)/timediff);
END;
```

Now prevent the controller from winding up beyond its limits in a manner which does not interfere too much with the ODE solver

```
IF OutputVar.val < MinOutput THEN
        IF OutputVar.der < 0.0 THEN
                OutputVar.der := 0.0;
        END;
ELSIF OutputVar.val > MaxOutput THEN
        IF OutputVar.der > 0.0 THEN
                OutputVar.der := 0.0;
        END;
END;
```

Model name

Ventilation

Inputs

Temperature	from either end of the ventilation duct
AbsHumidity	from either end of the ventilation duct

Outputs

as for Room_Room_IFace

Comments

This model defines the existance of a fixed flow of air between two rooms. Ventilation may start and stop during the day. This Ventilation model is two-way -- if x m^3/s of air flows from room A to room B, then the same volume flows from B to A. It may be convenient for there to be a one-way ventilation model. One-way ventilation would allow ventilation from room A to B to C to D etc, with room B only being affected by the difference between the stream from A and the stream to B. One-way ventilation models would require either that the user remember to always install ventilation models in pairs (to maintain an air mass

balance), or that the room model monitor the air mass balance and track the pressure in the room.

Properties

AirFlowRate	Volumetric flow of air through the ventilation system / m^3/s
OnTime	Time at which the ventilation is turned on each day / s
OffTime	Time at which the ventilation is turned off each day / s
SwitchedOn	TRUE if the ventilation is switched on.

Methods

Evaluate	Calculate the heat and water vapour flow through the ventilation system.
Event	Start and stop the ventilation.
Initialise	Initialise this instance of the Ventilation model.

Parameters

AirFlowRate	Default = 0.5 / m^3/s
Start	Default = 0.0 / hrs
Stop	Default = 24.0 / hrs

```
IF Evaluated THEN
    RETURN;
ELSE
    Model.Evaluate;
END;
IF SwitchedOn THEN
    temp1 := LinkedObjects^.model^.GetValue (SELF,Variable.Temperature);
    hum1  := LinkedObjects^.model^.GetValue (SELF,Variable.AbsHumidity);
    temp2 :=
        LinkedObjects^.next^.model^.GetValue (SELF,Variable.Temperature);
    hum2  :=
        LinkedObjects^.next^.model^.GetValue (SELF,Variable.AbsHumidity);
    dens := (Air.Density (temp2,hum2) + Air.Density (temp1,hum1))/2.0;
    HeatFlow := AirFlowRate * dens
        * (Air.Enthalpy (temp1,hum1) - Air.Enthalpy (temp2,hum2));
    WaterFlow := AirFlowRate * dens * (hum1 - hum2);
ELSE
    WaterFlow := 0.0;
    HeatFlow := 0.0;
END;
```

Model name

Wall

Inputs

Temperature	from either side of the wall

Outputs

as for Room_Room_IFace

Comments

This model represents a straightforward wall model.

Properties

Area	Area of the wall / m^2
OHTC	Overall heat transfer coefficient for heat transfer through the wall / W/m^2K

Methods

Evaluate	Calculate the heat flow through the wall.
Initialise	Initialise this instance of the wall model.

Implementation

```
IF Evaluated THEN
        RETURN;
ELSE
        Model.Evaluate;
END;
WaterFlow := 0.0;
assume the wall is sealed
HeatFlow := OHTC * Area
        * (LinkedObjects^.model^.GetValue (SELF,Variable.Temperature)
        - LinkedObjects^.next^.model^.GetValue (SELF,Variable.Temperature));
```

Appendix 6: Notes on the interpretation of component model descriptions

The model descriptions in Appendix 5 were derived automatically from the RefSim source code, written in TopSpeed Modula-2. Modula-2 was defined by Wirth (1982) and the TopSpeed variant was defined by JPI (1991). The algorithmic descriptions should be understandable to persons familiar with the Pascal programming language with the assistance of the following notes:

- BEGIN .. END bracketing differs from Pascal in that the BEGIN is assumed and END is always required.
- Modula-2 is case sensitive.
- No distinction is made between PROCEDUREs with return values (Pascal FUNCTIONS) and those without return values. Both are PROCEDUREs.
- PROCEDUREs in other modules may be called by statements of the form: "ModuleName.ProcedureName" (a qualified procedure call).
- Classes are implemented in a manner similar to RECORDS, except that they may contain fields which are PROCEDURES.
- Class methods may be called by statements of the form: "InstanceName.MethodName" (a qualified method call).
- The symbol SELF refers to the currently-executing object instance.
- Methods of the currently-executing object instance may be called without qualification.
- Comments in the source code have been rendered into italics.
- Variable declarations exist in the source code, but are not shown in the descriptions. Most of the variables are of type LONGREAL. Loop counters and array indices are of type CARDINAL. Dereferenced pointers are of type ModelPtr (i.e. pointers to the base class *Model*).

Doubly dereferenced pointers are of type ModelPtrPtr (i.e. pointers to ModelPtr).

- The PROCEDURE RefSys.Bounded returns the value of the first parameter unless it is less than the second parameter (in which case the second parameter is returned) or greater than the third parameter (in which case the third parameter is returned).
- Only the code for the Evaluate method is shown. Some models do significant computation in their Event methods. All non-abstract models have an Initialise method in which computation may also take place at the start of the simulation run.

Appendix 7: RefSim runtime options

The user may set a number of options when running a RefSim simulation which control the manner in which simulation is performed.

-m Do dynamic memory checking (default is not to do such checking). This is useful as a debugging tool for simulation utility features. When set on, pointers are checked for a NIL value before being allocated and deallocated pointers are checked to ensure that they were previously allocated and that the size of memory deallocated corresponds to the size originally allocated. Whether this option is on or off, RefSim still checks the difference between initially available memory and finally available memory.

-r Randomize pseudo-random numbers (default is to always generate the same pseudo-random sequence). Pseudo-random numbers are used by some models (e.g. *Door*) and setting this flag results in the sequence of numbers generated being different between different runs. It is useful to leave this option set off for debugging and when consistent simulation results should be obtained. Setting this option on allows statistical sensitivity analysis of the effects of random events.

-e Output a notice when each event takes place (default is not to report events). This is helpful for indicating the influence of discrete events on plant operation. Writes to the screen only.

-o Write initial scalar variable values to the standard output (default is not to do so). This provides confirmation that scalar (not TABLE or string) variable values have been read correctly. The default is off because this output can significantly delay starting a simulation.

-a Report the available models to standard output and halt without running a simulation (default is not to do so). This produces a list of the available model types and the size of an instance of each model type

in bytes. This size does not include the LinkedObjects list which may exist for each model instance, the ModelRec node in the ModelList for each model instance, or the sizes of any dynamic variables in the model instance. It is useful to use this option after implementing a new model type to confirm that the model type has been properly registered with the simulation environment.

-s Report each successful ODE solver step size and the number of failed steps before each successful step if one or more failures occured (default is not to do so). This is useful for debugging the ODE solver and stepsize controllers, and useful to the user because it provides a frequent indication of how fast the simulation is running.

-w Warn of any inconsistent model references (default is not to do so). If model A references model B but B does not reference A, this may indicate a problem in the model input data. If B is an environment model or A is an instrument model, then this warning does not indicate a problem.

-h,-? Output a version notice and a brief summary of the startup options, then halt (default is not to do so).

-c(0|1) Set the ODE solver step size controller to either:

0: The conventional controller of Press *et al* (1986) (default).

1: The PI controller of Gustafsson *et al* (1988)

-t<tol> Set the ODE step size controller tolerance to a relative per step error of <tol>. Default value is 0.01

-R<fac> Set the discrete event scheduler rounding factor to <fac> seconds. Default value is 10 seconds.