ISSN 0284 - 141X
ISRN KTH/IT/M--39--SE

KUNGL TEKNISKA HÖGSKOLAN

Building Services Engineering
Bulletin no. 39

Modelling and Simulation Methods for Modular Continuous Systems in Buildings

PER SAHLIN

Department of Building Sciences
Division of Building Services Engineering
Royal Institute of Technology
STOCKHOLM 100 44, SWEDEN

May 1996

This work is supported in part by the Swedish Council for Building Research, the Swedish National Board for Industrial and Technical Development, the Nordic Construction Company, the Development Fund of the Swedish Construction Industry, the American Society for Heating, Refrigerating and Air-conditioning Engineers, ASEA Brown-Boveri, and the following member companies of the IDA consortium:

ABB Airtech AB, Bengt Dahlgren AB/ CLC VVS-Energiconsult AB, Bo Lönner AB, Confortia AB, Energo AB, ENESSEN AB, EVR & Wahlings AB, FLK AB, Helenius Ingenjörsbyrå AB, INCOORD, LGM Consult AB, NCC, Postfastigheter AB, SCC VVS-Teknik AB, SIKAB Installationskonsulter, Skandia Fastigheter, Skanska Teknik AB, SP, Stifab-Farex AB, Stockholm Konsult, Strängbetong, Svenska Bostäder, Swedisol, TeknoTerm, Theorells, Tour & Andersson AB, Vasakronan AB, ÅF-VVS Projekt AB,

Modelling and Simulation Methods for Modular Continuous Systems in Buildings
Per Sahlin, Department of Building Sciences
Royal Institute of Technology, S-100 44 STOCKHOLM, SWEDEN

Abstract

Many special purpose computer programs for simulation of physical systems are in operation today. In industrial applications, they dominate completely over general purpose simulation environments. To gain runtime efficiency, often on yesterdays computers, the mathematical models of the tailored tools are frequently intertwined with the coded solution procedure, creating a monolithic structure. This makes it difficult to understand and improve the implemented models, also for the original developers. In the field of building simulation, researchers agree that the monolithic programs will be impossible to upgrade and maintain in the long term.

This thesis treats the design of the general purpose simulation environment IDA, and of the Neutral Model Format (NMF), a program independent language for modelling of dynamical systems using differential-algebraic equations with discrete events. IDA and NMF are used to effectively develop special purpose GUI-based tools that (1) are easy to use for end users without simulation expertise, and that (2) have good prospects for long term maintenance and reuse. IDA and NMF also serve as a general modelling and development environment for the sophisticated user.

The development of IDA commenced in 1987 at the Swedish Institute of Applied Mathematics. IDA has been used externally since 1990, primarily for simulation of building and energy systems. IDA applications and NMF model libraries have been developed for, e.g., district heating substations, fire scenarios in offshore structures, natural ventilation of buildings, building energy and loads simulation, refrigeration systems, ventilation of road tunnels under normal and fire conditions.

NMF component model descriptions can be automatically translated into the format of a number of simulation environments. Translators have been developed for ESACAP, HVACSIM+, IDA, MS1, SPARK, and TRNSYS. Some of these have computer algebra capability for automatic model processing.

This thesis consists of seven introductory chapters and seven separate papers. The main design issues for IDA and NMF are listed and briefly discussed in the introductory sections. Among them are: algebraic solution techniques, field models, implementation form and languages, equation based model description languages, modelling language translatability, and access to foreign subroutine based models. The papers give historical overview and insight into selected applications and topics.

Keywords: simulation, modelling, modeling, continuous systems, differential-algebraic, DAE, solver, language, model description, building, energy, load, multizone, air flow

Preface

Before you lie several pages of words about simulation software and languages. The text is one thing, the underlying implementations another. Together they represent a total body of work of some twenty man-years that has been carried out by primarily four people: Axel Bring, Lars Eriksson, Magnus Lindgren, and myself. The four of us have had the opportunity to work intensively together for close to nine years towards a common goal.

The implementations are a reality. All the software components and language constructs discussed in this thesis exist in commercial quality implementations, unless specifically stated otherwise. They have been tested on real-scale problems by independent users.

The modelling language, the Neutral Model Format (NMF), and translators to TRNSYS, HVACSIM+, and IDA, are controlled by ASHRAE[1]. An ASHRAE committee is responsible for approval of amendments to NMF.

Building simulation is an interdisciplinary subject, with ingredients from numerical analysis, information technology, signal processing, as well as the building sciences. This makes it a fascinating field of endless challenge, and of significant potential long term rewards in terms of global energy savings.

As for any young cross-disciplinary subject, there has been a shortage of appropriate scientific fora. The application oriented journals have, until recently, not really accepted the full implications of the computer revolution. IBPSA[2] has with its conference series provided a dedicated forum that we have truly enjoyed. Consequently, most of the papers that are part of this thesis have been presented at IBPSA conferences.

Solna, April 19, 1996

[1] American Society for Heating, Refrigerating and Air-conditioning Engineers
[2] International Building Performance Simulation Association

Contents

List of Papers

1. P. Sahlin, MODSIM - a Program for Dynamical Modeling and Simulation, Proceedings of the annual meeting of the Scandinavian Simulation Society, Espoo, Finland, 1988
2. P. Sahlin, E.F.Sowell, A Neutral Format for Building Simulation Models, Proceedings of the IBPSA Building Simulation '89 conference, Vancouver, Canada, 1989
3. P. Sahlin, A.Bring, IDA Solver - A Tool for Building and Energy Systems Simulation, Proceedings of the IBPSA Building Simulation ‘91 conference, Nice, France, 1991
4. A. Bring, P. Sahlin, Modeling Air Flows and Buildings with NMF and IDA, Proceedings of the IBPSA Building Simulation '93 conference, Adelaide, Australia, 1993
5. P. Sahlin, IDA Modeller - A Man-Model Interface for Building Simulation, Proceedings of the IBPSA Building Simulation '93 conference, Adelaide, Australia, 1993
6. P. Sahlin, C. Johansson, NMF-Based Aspect Models in STEP for Building and Process Plant Simulation, Proceedings of the CIC W78 workshop on computer integrated construction, VTT, Helsinki, Finland, 1994
7. P. Sahlin, A. Bring, K. Kolsaker, Future Trends of the Neutral Model Format (NMF), Proceedings of the IBPSA Building Simulation '95 conference, Madison, Wisconsin, 1995

1. Introduction

Building design involves a multitude of compromises. The quality of design decisions is closely related to the accuracy with which the designer is able to predict the characteristics of the future building. Ability to quickly provide quantitative measures throughout the design process is therefore crucial. Building space and investment cost are examples of rather obvious design parameters, others relate to the dynamical performance of the building, such as energy consumption, indoor thermal climate, air quality, light and sound levels. Dynamical computer simulation can provide an efficient method for prediction of the latter group. For this reason, a large number of building performance evaluation (BPE) tools have been developed. This thesis treats the development of software infrastructure and special languages for rapid development of BPE tools with desirable characteristics. Requirements for general purpose simulation methods for BPE are developed and analyzed.

Repeatable physical experiments on buildings are often difficult to perform. Many building systems are too bulky for laboratory measurements and long term measurement on real buildings are either expensive, due to the cost of an unoccupied building, or disturbed by building utilization. This makes computer based experimentation on building models attractive.

The building industry is oriented towards one-of-a-kind production, with a low relative investment in engineering. Coupled with the difficulty of in situ measurements, this creates unfavorable conditions for innovation and performance feedback. The process of natural refinement of engineering solutions is therefore slow. Ability to experiment with new solutions in the design office by simulation is bound to improve this situation, if the right tools are available.

Furthermore, since malfunctions in the delivered product are difficult to detect and quantify, designs that do not perform correctly, even theoretically, are sometimes realized. If design offices were obliged to demonstrate intended functionality on a computer model prior to realization, many such problems could be avoided.

Only a small fraction of the potentially worthwhile simulation experiments in the building life cycle are exploited today. The heterogeneity of the physical systems, the building traditions, and the stages of the design procedure create a huge spectrum of potential simulation situations. However, in spite of ample opportunities and strong motivation, the process of introducing BPE simulation tools into industry is slow. In this thesis, an attempt will be made at identifying some of the main limiting factors in this process. A subset of these factors form the motivation for the development of the IDA modular simulation environment, and for a suggested standard for expression of building simulation models, the Neutral Model Format (NMF).

Although IDA and NMF have been developed primarily for the building industry, they have wider applicability. Building simulation allows few simplifying assumptions. On the contrary, the requirements posed by this discipline span a very large domain that is sufficient also for many other applications.

In the introductory sections of this thesis, the main design issues for IDA and NMF will be examined in some detail. This discussion provides a framework for the seven separate papers, where the actual design is presented. An early overview of the IDA system is provided in Paper 1. A separate report [Sahlin 1991], that is not part of this thesis, also provides additional overview material. The NMF discussion in Section 4.2 assumes that the reader is familiar with the basic NMF concepts, and frequent references are made to the *NMF Handbook - an Introduction to the Neutral Model Format* [Sahlin 1996a], which should be read in parallel.

1.1 Building Simulation Bottlenecks

The identification of major problem areas in the application of BPE methods is hardly a controversial issue. The problem is not so much to identify the problems adequately but rather to provide solutions. Nevertheless, the route taken in the IDA project is to a large extent based on our particular perception of the dominating bottlenecks.

1.1.1 Technology Transfer

Much of the scientific discussion about building performance simulation focuses on mechanisms for technology transfer. Let us examine some details of the transfer process using a model of an ideal situation, and discuss a common malfunction. The main aspects of this ideal process for production of simulation results can be represented as in *Figure 1-1*.

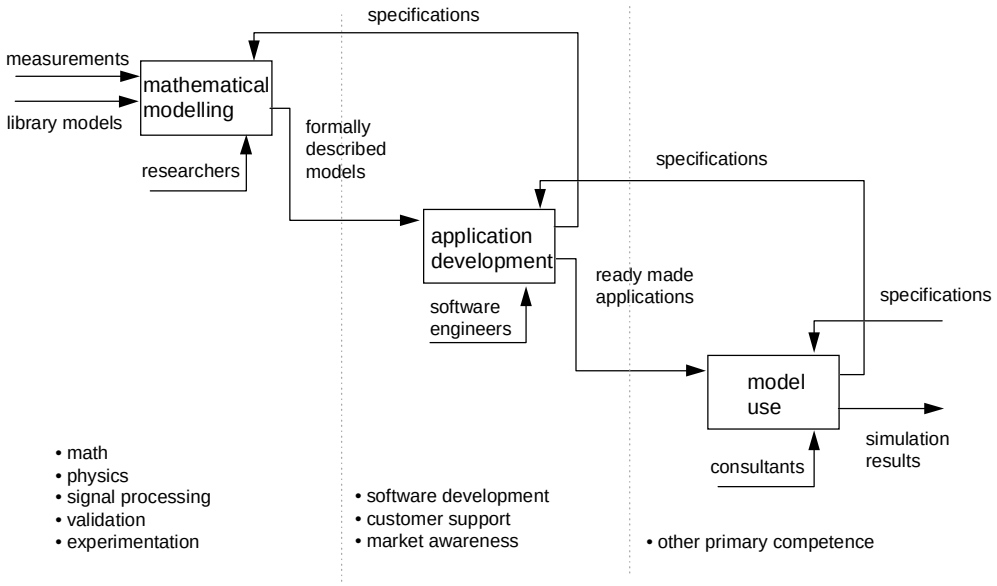


Figure 1-1. An ideal process for production of simulation results

In the process, three distinct professions interact in order to create specified simulation results. The process is driven by specifications from downstream actors, starting with the client in need of simulation results (not in the figure).

Unfortunately this ideal situation rarely occurs in practice. One common malfunction is in the relationship between researchers, doing the mathematical modelling, and application developers. In fact, examples of working business relationships between these groups are hard to find in practice. It seems as if the dominant mechanism of technology transfer between these actors is that of researchers changing profession and becoming instead software developers. Alternatively, researchers feebly try to market simulation applications directly, without the assistance of a professional software developer and vendor. Both methods have obvious disadvantages.

A practical problem is the form in which models are described when they are transferred from mathematical modellers to application developers. The dominating method today is still regular research reports, where model equations are presented and defended. Sometimes this description is accompanied by some implementation of the model, as a stand alone program or as a set of subroutines. More often than not, the software developer must re-implement the models in a more suitable form for the final product. Although this is no insurmountable task, it is certainly a practical obstacle that causes friction in the process, and it is a problem that can be dealt with by standards such as NMF, as we will discuss in Section 4.2.

IDA is designed to be useful for all three actors in the simulation process, and by providing a common framework, it is intended to enhance the overall productivity as well as the quality of the products.

1.1.2 Access to the Right Tool for the Job

The total number of BPE tools that have been presented scientifically or commercially is quite astonishing. It may seem as if there exists a tool for every conceivable design evaluation situation. However, for a BPE engineer with an urgent problem at hand, the number of practically available tools is normally quite low, often zero. A number of practical restrictions create this situation:

- Information about the existence of a tool has not reached the engineer.
- A tool exists only in a "research implementation," practically accessible only to the developers and their immediate circle.
- A commercially supported appropriate tool exists, but is unavailable due to distance, language, engineering culture, or computer platform.
- The tool supports a particular equipment product line, which is suboptimal for the design at hand.
- The introduction time of the tool is too high for the given project.
- The tool itself is too expensive.

A consequence of this situation is that available tools are frequently used far beyond their intended application regime, e.g., an office room model is applied to evaluate the performance of a new solar collector concept. Another common scenario is that intentionally "quick and dirty" spreadsheet models are incrementally improved in several projects until substantial amounts of development time have been invested, but the end result is still a limited tool that is practically useful only to its developers.

Some of the obstacles that prevent tools from reaching end users will surely vanish with time, others are more fundamental. The heterogeneous nature of the different building design problems that could be addressed with simulation will surely remain. Therefore, general purpose simulation tools will be necessary. However, general purpose tools will often be too time consuming and require too much expertise from the user, to make special purpose tools superfluous. There is a fundamental contradiction between the user friendliness and the generality of a tool. A tool with a more limited scope will always be easier to use, given the same general user interface standard.

To cater for continually changing demands, continued development and alteration of large numbers of special purpose tools is therefore unavoidable. To make this economically feasible, the cost of application development will have to be cut far beyond the lowest attainable cost of today. Furthermore, it must be possible to upgrade the performance of an application gradually, both in terms of model capability and user friendliness. Rough implementations must be available within "spreadsheet development" times, and first versions of graphical user interfaces (GUI) must be generated equally quickly, without sacrifice of the possibility of future gradual improvement.

The use of general purpose simulation methods for cost-effective development of end user application tools is the main topic of this thesis.

1.1.3 Integration into Design Environment

When designers are asked what they think would best contribute to increase the use of BPE tools, a common response is "to be able to transfer data directly from CAD" [Wernstedt 1995]. The tedious process of entering the same geometrical data into an array of tools, all requiring similar information, is immediately identified as a potential area of automation. In spite of this, few integrated systems are in operation. The problem of providing solutions with acceptable generality in this area is significant. Contributions have been made by several substantial recent projects, among them the Finnish RATAS project [Björk 1995] and the large EC funded COMBINE projects [Augenbroe 1993, 1995] (http://erg.ucd.ie/combine.html).

COMBINE and related projects have focused on the definition of a general data model for buildings, a so called product model (PM). Interfaces have been developed to a range of CAD and BPE tools. The development of each such interface requires a significant effort, and it is important to develop methods to automate this process. Systematic mapping between different product models of the same object (aspect models) is a crucial technique that has been studied by, e.g., Amor [Amor 1995]. If the production of tools for specific problem types is streamlined as discussed in the previous section, a need to quickly develop corresponding PM interfaces will follow.

The work on IDA and NMF has not been primarily focused on CAD integration. However, some initial work in this field is reported in Paper 6 of this thesis.

1.1.4 Validation

Blind validation tests of BPE software against measured data generally exhibit discouraging results [IEA B&CS Annex 1 1981], [Lomas 1994]. It is not uncommon

that independent testers using different tools on the same problem differ by a factor two on some estimated quantities. This situation is unacceptable. However, a positive trend can be discerned in recent studies. More systematic methods for model screening have been developed [Judkoff 1994]. Many of the problems in this area stem from the monolithic structure of present tools, and from lack of standardization of input data.

The present work does not deal directly with the validation issue. However, the proposed methods can be expected to have positive effects on validity due to the following circumstances:

- Simulation tools that have been implemented with the proposed techniques are accessible in their smallest details. Individual parts of a model can be validated independently. Any variable in a model can be measured in validation work, not just the variables that are presented as output in the finished application.

- Modelling errors can be completely separated from solution errors. In the scope of a validation study, model equations can be resolved with arbitrarily high accuracy. Solution errors are then effectively eliminated at the expense of execution time.

- The proposed methods deal mainly with the form of BPE tool implementation, not the content. For most applications, already existing tools can be re-implemented in the new form. When a tool is selected for re-implementation, validation status is a key selection criterion. It is natural and often quite easy to make sure that the new tool is able to perfectly reproduce results from the template. After this step, it is frequently possible to identify weaknesses in the model of the template tool, and improve the modelling further. In many existing tools, limited solution techniques and computer resources, at the time of the original implementation, have lead to approximations that are no longer necessary.

1.2 Modular vs. Traditional Tools

In 1985, a group of leading building simulation specialists gathered in Berkeley, California, to discuss future directions [Clarke 1985]. There was a consensus that most of the tools, that had been developed until then, were much to rigid in their present structure to be able to accommodate the improvements and flexibility that would be called for in the future. Each added feature to the existing tools required a larger implementation effort than the previous one. The complexity of input data was already overwhelming to most potential users. Basic methodological improvements, such as a complete change in solution strategy, were close to impossible to carry out since most of the program structure would be affected. In response to this meeting, some significant projects were initiated to develop a modular architecture for BPE tools that would give the desired flexibility, maintainability, and that would be possible to upgrade in all respects. In the US, the SPANK project was started (now SPARK) [Sowell 1986]. In the UK, the Energy Kernel System was proposed [Clarke 1986], and in France, the ZOOM project [Bonin 1989] was the main vehicle of renewal.

Eleven years have passed. The tools that were deemed inadequate at the Berkeley meeting are still completely dominant, although today often wrapped in modern GUIs. The Berkeley analysis still applies. Nothing has happened that makes the monolithic

tools manageable in the long run. The difficulties with the practical implementation of good modular tools were obviously underestimated. The industrial demand for tools with new and more flexible architecture has, at the same time, been weak. The drive to incorporate new GUI (Graphical User Interface) technology seems to have absorbed nearly all industrial development ambition.

A workshop with a number of invited experts was organized jointly by the US departments of energy and defense in the summer of 1995, again to discuss the future of the field. A summary from the workshop [Crawley 1995] confirms that a majority of researchers still have high expectations on modular methods, in addition to tool integration by way of building product models. No views were expressed that support a long life expectancy for the traditional tools.

Many different interpretations of the term *modular methods* exist, and others, such as *object oriented* or *modular simulation environments* are often used with roughly the same meaning. Is for example a "traditional" building simulation program that has been implemented in C++ modular? Is a separation into loads, systems, pre- and post-processing, sufficient for a program to be called modular? Most developers call their product modular and this leads to misunderstandings in the scientific debate. For the purpose of this thesis, we have chosen to use the term *modular simulation environments*, the meaning of which we will attempt to define next with appropriate accuracy.

1.3 Modular Simulation Environments (MSEs)

First, MSEs have nothing to do with object oriented programming (OOP). MSEs may or may not be implemented with OOP tools, recent developments generally are.

Two criteria that a modular simulation environment must fulfill are:

1. Models are treated as data. The key characteristic of an MSE is that the mathematical models are exchangeable. The environment allows radically different models to be used for the same physical device.

2. Separation of software modules for modelling and solution. The software architecture allows exchange of solvers. Although only a few MSEs really offer a selection of different solvers, they are open in this respect.

A separate modelling tool is often used for model formulation. This tool generates a system model, generally expressed in a *modelling language*. The model is then treated by a solver and/or other model processing tools. Some MSEs rely on regular programming languages as part of their model description. For these, component models are typically described as subroutines with prescribed structure, while interconnection of pre-programmed component models into system models is described with a dedicated language. Other environments have complete modelling languages, which describe component behavior as well as system structure.

Physical systems that are simulated in MSEs are normally modular in nature, i.e., they naturally decompose into subsystems. Frequently, identical subsystems are repeated a number of times in a model, a fact that is taken advantage of in many tools. Further-

more, the systems should have a basically continuous behavior, meaning that equations used to describe them, as well as forcing functions, will have a limited number of discontinuities. Purely event driven systems are excluded. Discrete time submodels may be encompassed by some MSEs.

If characterized by equations, the physical systems under consideration will require both algebraic and differential equations. Differential equations can be either ordinary (ODE) or partial (PDE), although current tools require that PDEs are explicitly discretized in space and thus turned into ODEs. Note that, in contrast to many widely used commercial tools, the simulation environments we are concerned with here are not limited to ODEs only. They allow a free mixture of algebraic and ordinary differential equations generally referred to as differential-algebraic systems of equations (DAE).

Furthermore, the simulation tools under discussion are rarely used for applications where a strict formalism for generating governing equations exists. In, e.g., electrical circuit analysis, multibody mechanics, or structural strength analysis special purpose systems may be more advantageous.

1.3.1 User Roles and Tool Types

Much of the discussion of this thesis is based on a categorization of user roles and tool types. These issues are discussed from the IDA perspective already in Papers 1 (1988) and 4 (1993). Since then our view has matured, and we will devote some space here for additional discussion.

Most of the tools that qualify as MSEs are exclusively intended for quite sophisticated users, who are well versed in mathematical modelling, advanced use of computers, and have some grip on numerical methods. This type of tool is called a *model-lab* MSE in the future discussion. Additional discussion of this term can be found in Paper 4. The corresponding user will be called a *model-lab user*[3]. Typically, the *researchers* of *Figure 1-1* are of this category. However, a key point of this work is that model-lab tools are not directly suited for building designers lacking special simulation expertise, who use simulation as one of several methods for design evaluation.

This group of users, that run ready-made BPE tools, will be categorized as *end users*, with the further distinction *advanced*[4] and *designer*[5], to emphasize the latter category's main area of competence. Both these categories have the role of *consultants* in *Figure 1-1*. An advanced end user will typically be able to build working system models out of ready-made component models, i.e., interconnect models into a suitable structure, give appropriate parameters, and carry out a simulation exercise. This type of tool will be called a *model-lab design tool*.

The designer end user will work with automatically generated or available template system models. He will normally interact with a tailored GUI that, in principle, could use any type of simulation engine, i.e., monolithic as well as MSE based.

---

[3] In the nomenclature of Papers 1 and 4 this was a *component maker*.
[4] Roughly *system maker* in Papers 1 and 4.
[5] Roughly *black box user* in Paper 1and *end user* in Paper 4.

Another role that will be needed is that of the *application developer*, the *software engineer* of *Figure 1-1*. An application developer prepares the ready-made tools that are required by end users. GUI design and implementation is a main task. The raw mathematical models that are packaged by the application developer are typically formulated by others. The developed GUI assists end users in the formulation of solvable simulation problems based on design parameters of the modelled object, i.e., a task that model-lab users normally carry out for themselves.

The given categorization is exaggerated here for the purpose of clarity. In real life, roles will be mixed and this is a desirable process. Users will hopefully be encouraged to traverse boundaries, given the possibility to do so with the new methods.

	model-lab user	***application developer***	***advanced end user***	***designer end user***
model-lab MSE with application generation support	medium	high	low	low
model-lab MSE	high	n.a.	medium	low
model-lab design tool	high	n.a.	high	medium
simplified design tool	high	n.a.	high	high

Table 1-1. User types vs. tool types, including level of suitability

1.4 The Problems Addressed

After the general discussion of building simulation bottlenecks and remedies, we will now narrow the perspective to more manageable proportions. As we have already mentioned, the present work is primarily focused on the first two problem areas *Technology Transfer* and *Access to the Right Tool for the Job*. The following general problems are addressed:

1. The degree of model reuse in the field is low. Available mathematical models are generally packaged in a form that is unsuitable for direct reuse in diverse settings. Researchers produce validated and documented mathematical models, but these are then presented in a form which is either too general, i.e., written equations in a report, or too specific, i.e., intertwined with a solution algorithm in a special implementation. The same problem applies to model reuse between different projects that are carried out within the same group, but with different simulation tools.
2. With available techniques, the development cost of special purpose application tools is too high. This in turn prevents exploitation of many potential simulation problems, since the market for each individual special purpose tool is small. The collective effect is that simulation is poorly utilized as a general method.
3. Special purpose tools that are developed with available techniques are inadequate for the end user in the following respects:

a) They offer no practical way for a user to adapt the tool, in an unplanned way, to suit the problem at hand, i.e., the ability to re-program is exclusive to the developers.

b) The mathematical models are documented separately, creating a double source problem. The user is often in doubt regarding the correspondence between the documentation and the implemented model.

c) They are generally too inhomogeneous in terms of user interaction principles to allow a user quick transitions between different tools. The required common principles shared by, e.g., two good Windows applications, are too superficial to make it possible to have a large number of different applications simultaneously active (mentally).

d) Only the subset of model quantities that have been selected by the developers is available for study. If for example wall temperatures are interesting, they may not be possible to present.

4. Available MSEs are inadequate, both regarding the encompassed problem types, and the possibility to distribute attractive special purpose applications.

1.4.1 Primary Target Applications

The solutions we will propose to the discussed problems are general in nature, i.e., they are not limited to a certain range of target applications. However, some building simulation applications are more important than others in terms of economical impact and applicability of simulation methods. To facilitate quantifiable measures of success, a list of *primary target applications* have been selected. These applications are briefly presented in the next few sections, along with some motivation for their selection.

1.4.1.1 Building Loads and Energy Calculation

A well motivated concern among building simulation researchers regards the applicability of general purpose MSEs to the thermal modelling of building fabric. Heat transfer and storage in the building structure is naturally an important process, and an implementation must be reasonably efficient in this respect in order to be attractive. It is not immediately obvious that a general purpose simulation tool can show adequate performance on these important special problems.

In addition to the heat equation in one dimension, non linear heat exchange due to convection and radiation must be resolved.

1.4.1.2 Multizone Air Flow

A less established group of simulation problems concern the prediction of pressure levels and air flows between ventilation zones, in ventilation ductwork, and through openings in the building envelope. Existing special purpose tools for this type of problem include Movecomp [Herrlin 1992], CONTAM93 [Walton 1994], and COMIS [Feustel 1990]. The basic equations for multizone air flow are given in Paper 5.

Although the available tools for this type of problem have yet to penetrate into industrial practice, the ability to solve the underlying equations are of significant importance. The same basic equations apply for any fluid distribution network with

turbulent flow, e.g., ventilation ductwork, radiator or sprinkler systems. In the multizone air flow case, equation coefficients vary with several orders of magnitude[6]. The overall structure of the equations is also unfavorable, with multiple nested flow circuits. Both these factors make the multizone air flow application more demanding than many other flow circuit problems. Therefore it represents a suitable test application.

1.4.1.3 Coupled Thermal and Fluid Flow Problems

A major weakness of most existing tools in both the previous categories is that they do not allow mixed problems. The thermal tools require air flows as input, and the air flow tools similarly require given zone temperatures. This is an artificial limitation purely due to the technical difficulty of solving the combined coupled problem. In some applications, e.g., studies of the effect of individual temperature setpoints in zones connected by an open door, the coupled problem must be addressed.

1.4.1.4 Demand Controlled Ventilation

An important mechanism that is lacking in present monolithic tools is the ability to rapidly compose an arbitrary system model and study a certain aspect of it, e.g., the performance of a control algorithm. An appropriate model must be constructed, from scratch or from existing library material. The user must also formulate suitable simulation experiments. The very nature of this task is general purpose, and we will therefore select a sample problem that exhibits some of the general difficulties that must be surmounted, e.g., hysteresis and discontinuities.

The problem we have selected regards demand controlled ventilation of a single zone, by adjustment of the fresh air fraction in a mixing box. The problem has been studied by Emmerich and co-workers [Emmerich 1994].

1.5 Thesis Outline

The background and objectives of this work are presented in the next main section. In Section 3, a review of related work is done. The main design issues of IDA Solver and Modeller are presented in Section 4.1. A similar discussion for NMF is carried out thereafter in the Section 4.2. In Section 5, some comments are made to the seven included separate papers. This is followed by a review of application projects and a general discussion of MSE problems in Section 6. The thesis is concluded and summarized in Section 7.

Depending on previous familiarity with the field, it might be optimal to start with reading the first two papers, to get some overview and introduction, and then return to the main text. Papers 3 through 5 relate directly to the discussion of IDA in Section 4.1, and should be read in parallel.

The reader who is new to NMF is advised to read the NMF Handbook before embarking upon the NMF design discussion of Section 4.2. The last two papers deal with future NMF development, and can be read separately, after Section 4.2. Paper 6 presents our initial work on the relationship between NMF and product models. In Paper 7, some proposed new NMF constructions are presented.

[6] Due to the difference in flow resistance between, e.g., an open doorway and a microscopic crack.

2. Organization, Objectives, and Initial Conditions

2.1 Organization, Project Background, and Author's Role

The IDA project commenced in 1987, as a cooperative effort between the Swedish Institute of Applied Mathematics (ITM) and the Division of Building Services Engineering at the Royal Institute of Technology (KTH). The author was project leader at ITM and Axel Bring was responsible for the application side of the project at KTH.

In a previous series of contract projects for the Norwegian oil company Statoil, a novel numerical technique for modular simulation had been developed at ITM [Söderlind 1984, 1985a, 1985b, 1986]. The Statoil application concerned simulation of oil-gas separation plants. The new method featured pre-compiled component models without the restriction of input-output oriented modelling (cf. Paper 1). The author was aware of the on-going discussion about modular methods in building simulation and initiated a joint research proposal with KTH. A number of subsequent proposals have followed, but the basic organization has remained roughly the same up until 1995, when an industrial consortium was formed around IDA.

The main responsibility of the author has been to compose an adequate set of capabilities for the building simulation application. Implementation has to a large extent been carried out by Axel Bring and Lars Eriksson for IDA Solver, and Magnus Lindgren and the author for IDA Modeller. Some application projects have been carried out by the IDA team, others by independent developers. The author is the main designer of NMF, but many others have also contributed.

2.2 Objectives of the IDA Project

The ambition in the first phase of the project was to create a prototype general purpose MSE, based on the ITM numerical technique, and with a model-lab GUI. The funding came from ITM and the Swedish Council for Building Research. Building simulation was seen as a good reference application, but the aim at that stage did not include the creation of easy-to-use special purpose tools for the designer end user.

In subsequent funding requests, the objectives have shifted towards a more pronounced building simulation orientation. The importance, for this field, of ready made special purpose applications has been realized. The original pronounced "technology push" has gradually been replaced by a demand driven approach, where the simulation needs of the building industry have been the guidance. The problem formulation of Section 1.4 has been the target for the last few years.

2.3 Thesis Objectives

For the purpose of this thesis it is necessary to define a more closely focused set of objectives, which can be clearly associated with the author's role in the overall project.

The goals of this work are the following:

1. The key points of the IDA specification will be presented and motivated with respect to the problem formulation of Section 1.4.
2. The same will be done for the NMF design.
3. The appropriateness of both IDA and NMF will be evaluated with respect to the primary target applications, as presented in Section 1.4.1, with particular emphasis on
   a) the efficiency of the application development process, and
   b) the general usability of the developed applications.
4. The main result will be a conclusion regarding the applicability of MSE based technology as a replacement for traditional design.

2.4 Initial Conditions and Methodology

As mentioned, some design decisions for IDA had already been made at the onset of the project. The appropriateness of these will be evaluated along with the full implementation, but a complete redesign with respect to these aspects have never been considered. First, the basic architecture of the existing numerical method and prototype solver implementation from the Statoil project were adopted as they were. Secondly, the use of Common Lisp for the GUI implementation was also a decision that had been taken at ITM prior to the IDA project. To investigate the applicability of Lisp for this type of application was a goal of the original IDA project.

Some basic methodological principles for the IDA project have been:

1. Learning by doing. We believe that the appropriateness of MSE technology cannot be determined without full-scale implementations that are put to test on real problems by independent users.
2. All of the key software components involved must be in complete control by the development team. The use of available fixed software components, for key parts of the system, will infringe on the possibilities of creating an optimal overall system. Only very stable and commonly accepted tools, such as compilers, GUI packages, numerical library routines etc., are exempted from this principle.

This holistic approach obviously puts limits on the amount of effort that can be devoted to each individual item. Ad hoc solutions must be applied without full scientific rigor to allow the creation of a working whole in the limited scope of the project. However, if the end result is found to be an adequate solution to the problem, all the individual parts are also adequate in their given roles.

3. Literature Review

In this section, an initial overview of related projects and tools will be given. A number of available and emerging MSEs are discussed below, in a loosely defined order of maturity.

3.1 TRNSYS

TRNSYS was developed during the early seventies at the Solar Energy Lab at the University of Wisconsin [Klein 1976] (http://www.engr.wisc.edu/centers/sel/trnsys /index.html). The primary application then was solar energy systems. It was one of the first modular simulation solvers for DAEs and it is distributed at low cost, bundled with some related tools. Several compatible modelling tools have been developed independently, e.g., PRESIM and IISiBat. An important feature of TRNSYS is that component models are pre-compiled. This means that end users may compose system models with fixed components without access to a compiler.

The design of TRNSYS commenced in 1972, i.e., well before many of the monolithic programs. In hindsight, the conception of the simple, yet adequate, program structure is truly commendable. TRNSYS version 14.1 was released in mid 1994 and featured a considerable leap in the offered numerical methods. The traditional successive substitution sequential solver was complemented with an implicit solver, however still operating with a fixed timestep. A bold decision was to introduce a set of non-backwards compatible changes to the prescribed structure of the TYPE subroutines, making it necessary to manually update the entire model library of each user. With the implicit solver came also the possibility to "back solve," i.e., to connect model variables IN to IN and OUT to OUT. Unfortunately, many of the available libraries remain in the old format.

A fundamental limitation with TRNSYS is the fixed timestep. A simultaneous solver is required to provide reasonable robustness for many problem types, but to attain acceptable efficiency with such a solver, the possibility to take long timesteps must be utilized when little is going on in the simulated system. Converting the TRNSYS community to variable timestep would be rather difficult, since such a large number of existing models are based on the fixed timestep assumption.

The TRNSYS solver has also been used for development of end user applications. CASSIS [Rongere 1992b] and CSTBât [Escudié 1994] are examples of rather complex such applications. It is also common to use the TRNSED tool to provide end users with tailored input files, some parameters of which can be altered by the end user, but not the file structure.

3.2 ESACAP

ESACAP is an electrical circuit simulator which has been generalized into a full DAE solver [Stangerup 1988, 1991a, 1991b]. The development of the ESACAP language and solver was initiated in the late seventies at ElektronikCentralen in Denmark. Two prototype NMF translators have been developed [Pelletret 1994a], [Lorenz 1994a]. ESACAP is now commercially available from STANSIM, Denmark.

3.3 HVACSIM+

HVACSIM+ [Clarke 1985] [Park 1985] is a general MSE for building simulation with similar characteristics as TRNSYS in terms of model format and structure, but with more recent numerical techniques than in the original TRNSYS, e.g., a simultaneous algebraic solver (SNSQ), variable timestep, and input event detection. Component models are precompiled and input-output oriented. It was developed by the National Institute of Standards and Technology (NIST) in Maryland and released in the mid eighties on a public domain basis. HVACSIM+ is delivered with a rather comprehensive library of building simulation models. Unfortunately, the support level is currently very low.

HVACSIM+ has an interesting structure with *superblocks*, i.e., modules that are solved in sequence. Each superblock has a separate timestep. The idea is that closely coupled subsystems with a common timestep requirement are to be collected into a superblock. Typically the building model is put into one superblock, the control system into another. Each superblock is divided into a number of *blocks*, each of which is solved with a full matrix technique.

Another feature is variable *freezing*. Variables that have remained constant for some time are simply removed from the system of equations until some condition is fulfilled.

The chosen design puts the burden of partitioning the problem into a suitable structure on the user. A fundamental problem is that tightly coupled subsystems often are distributed throughout the physical structure. In a mixed flow and thermal problem, for example, the flow problem may in reality be decoupled from the thermal problem (no buoyancy effect), but both flow and thermal equations exist in virtually every unit, i.e., an efficient decoupling must be able to cross unit boundaries.

HVACSIM+ is equipped with a primitive text based interactive modelling tool, HVACGEN. The file produced by HVACGEN is difficult to set up manually (too cryptic) and a normal user is therefore left with the tedious exercise of running the interactive tool. This is an unnecessary limitation. With a slightly more legible input file that could be written without HVACGEN, the overall program would be more useful. Furthermore, utilizing the block, superblock and freeze structure effectively to save execution time is a genuinely difficult and (man-) time consuming task. A good modelling tool could be a very useful help with this.

HVACGEN prescribes a limited number of variable types, such as temperature, massflow, pressure etc. with fixed units. Fortunately, an ‘other’ category also exists. This ad hoc selection of certain variable choices is another feature which limits the applicability of the system in an unnecessary way.

In conclusion, HVACSIM+ is a tool that has good potential to become useful. Unfortunately, little or no development has been done since the original release, and without continuous support the program is of limited value to most potential users.

3.4 SANDYS

SANDYS is a general DAE solver and modelling language developed by ASEA, Sweden, in the early eighties. It is commercially available from ABB Corporate Research [Ohlsson 1991].

SANDYS features a comprehensive range of variable timestep integration methods, including explicit methods. Both the language and the solver has good facilities for description and simulation of multiple mode systems with complex logic and internal events. The solver is equation based, and requires an integrated Fortran compiler.

The SANDYS system has, to the best of our knowledge, not been applied to building simulation.

3.5 ALLAN.Simulation

ALLAN.Simulation is a graphical modeller and solver combination developed in the mid eighties by Gaz de France and CISI Engineering. It is commercially available, since a few years, from the developers [Jeandel 1993].

ALLAN relies on the NEPTUNIX solver [Nakhle 1982], which is a commercial DAE solver that currently runs on a limited number of machines. The graphical front end has a clear model-lab orientation.

ALLAN has been widely used for building simulation applications, including thermal modelling of the building fabric. The project has a pronounced objective of constructing well validated and documented model libraries. A new language, ULM [Jeandel 1994a, 1994b], for this purpose has been developed. ALLAN is an important point of reference in the discussion of the applicability of MSE techniques to building simulation.

3.6 Dymola

Dymola is a commercial modelling tool with symbolic algebra capabilities and interfaces to several solvers, available from Dynasim, Lund, Sweden [Elmqvist 1993] (http://www.dynasim.se/).

The Dymola language [Elmqvist 1978], was the first modelling language that was based on declaration of differential-algebraic equations. After some years without development, a number of code generators for industry standard ODE based solvers were developed in the early nineties. These generators require that the translated model can be converted into ODE form, which is not always possible [Mattsson 1986]. Today, Dynasim markets also a DAE solver Dymosim, which is a DASSL[7] derivative.

Dymola is further discussed in Section 4.1.1.1. It has been applied to a range of applications, but to the best of our knowledge, not to building simulation.

[7] DASSL is a standard numerical library Fortran routine (SLATEC library) for solution of DAEs, that are supplied in a user defined subroutine. It was originally developed by Linda Petzold [Petzold 1982]. Several derivatives exist.

3.7 CLIM 2000

CLIM 2000, a graphical modelling tool for building applications, has been developed by Electricité de France for internal use [Bonneau 1993].

CLIM 2000 has recently changed to the ESACAP solver. It is equipped with a comprehensive model library, including advanced thermal building models that have been validated in the scope of international comparative projects [Lomas 1994]. Special library documentation methods have been developed [Rongere 1992a], but the source code for component models are stored in the native language of the solver.

Although not available to the general public, CLIM 2000 is an important project from a scientific point of view. To the best of our knowledge, it represents the largest effort to create an MSE for building applications.

3.8 MS1

MS1 is a graphical multi input language modeller with interfaces to several solvers by Lorenz Simulation, Liege, Belgium in cooperation with Electricité de France [Lorenz 1991, 1994]. MS1 is commercially available from Lorenz Simulation.

MS1 models can be described with Bond Graphs, Linear Graphs, and Block Diagrams [Lorenz 1987]. Model descriptions are converted to a so called pivot language, that is based on NMF. Solver code is then automatically generated using symbolic algebra techniques. Currently supported solvers include ESACAP and a range of ODE based solvers.

MS1 has, to the best of our knowledge, not been extensively applied to building simulation.

3.9 ISE

ISE is a graphical programmable front end that can be used with several building simulation engines such as TRNSYS and COMIS [Pelletret 1995]. A TRNSYS application, IISiBat, is available from the developers at CSTB, France.

Object oriented data models are tailored for the input files of the target environments. These files can then be constructed interactively by end users using a hierarchical structure with graphical objects for each modelled component. A system is available for well structured documentation of component models. This documentation will in future versions be complemented by NMF model descriptions. ISE is currently implemented in ILOG Lisp.

3.10 SPARK

SPARK is a DAE solver under development at Lawrence Berkeley Laboratory, California [Sowell 1986] [Buhl 1989, 1993]. SPARK is intended to be a modular complement to the widely used monolithic DOE-2 building energy simulation tool. A library of native models for building secondary mechanical systems is available, as is a prototype NMF translator [Nataf 1995]. A graphical model editor is also under development.

SPARK is to a large extent based on a general algorithm for sparsity utilization [Sowell 1984]. A similar algorithm has also been implemented in IDA, and has performed well on some large problems. A fundamental problem with any advanced sparsity utilization algorithm lies in the very fact that it makes assumptions about the system structure. Many real problems will change structure during the integration process, e.g., when a flow-path reverses direction or due to some controller action. If the algorithm is robust enough to deal with all such problems, it may become inefficient. There is a fundamental contradiction between efficiency and robustness that makes it necessary to have access to a range of sparsity utilization levels, especially during model development.

3.11 OmSim

OmSim is a graphical modelling tool with accompanying language, Omola, under development at the Dept. of Automatic Control at the Lund Institute of Technology, Sweden [Anderson 1990, 1994, Mattsson 1992] (http://control.lth.se/~cace/). OmSim is integrated with the DASSL derivative DASRT, for solution of DAEs with discrete events.

Omola is a later descendant of Dymola, which originates from the same department. Omola offers an object oriented structure for construction of large scale DAE-based model libraries. In comparison with NMF, Omola has a less pronounced orientation towards model translatability to a range of target environments.

OmSim has a clear model-lab orientation, targeting primarily the sophisticated control engineer. It has, to the best of our knowledge, not been applied to building simulation problems.

3.12 Smile

Smile is a general purpose differential-algebraic modelling and simulation environment, primarily for energy related problems. Modelling language, model library, and related software are under development at the Technical University of Berlin [Jochum 1994, 1995] (http: //www.cs.tu-berlin.de/~smile/synopsis.html). Smile features a selection of dedicated solvers with implicit as well as explicit integration methods. The Smile language is rooted in Objective C. Model descriptions are partly interpreted at runtime and partly translated to C.

The SMILE project is a cooperative project between computer scientists, numerical analysts, and application experts. Smile has been applied to a range of solar and district heating applications.

3.13 THUVAC

THUVAC is a graphical modelling tool and solver for simulation of HVAC related modular systems. It is under development at the Dept. of Thermal Energy, Tsinghua Univ., Beijing, China [Yi Jiang 1994].

THUVAC addresses a more limited problem domain, HVAC system modelling. However, it features a dedicated model language and has a clear vision of the type of end user problem that it addresses. Unfortunately, successful end user tools must generally be specialized for a certain culture. In the case of THUVAC this may well

be a Chinese building design culture that is bound to be different from most western design cultures.

3.14 EKS

The Energy Kernel System, EKS, is a C++ toolkit for development of energy related simulation design tools. It is under development by among others the Univ. of Strathclyde, Scotland [Clarke 1986] [Charlesworth 1991] [Clarke 1993].

EKS has a clear orientation towards efficient production of tailored end user design tools. Drawing heavily on OOP techniques, EKS attempts to automate the procedure of coding BPE tools. Existing BPE tools and methods are re-implemented according to EKS principles and the result is structured into a library of objects. Objects are then mixed and matched to form BPE tools with desired characteristics.

Although general purpose solvers are an integral part of the concept, EKS bears little resemblance to other MSE projects. Unfortunately, the present rate of development is low.

4. Results

4.1 Simulation Environment Design Issues

A well balanced list of capabilities is a prerequisite for a MSE to have a future. In this section, we will present the IDA-version of this list. Some topics, like specific requirements for the solver, are treated in the accompanying papers, and will not be thoroughly discussed here. A complete summary of the main design issues is located in Section 7.1.

4.1.1 Problem categories

The list of targeted problem types lies naturally at the heart of the specification. Most of the discussion of problem types is also presented in Paper 3, *IDA Solver - A Tool for Building and Energy Systems Simulation*, which was presented in 1991. This text repeats some of the main points of that paper, for the sake of completeness. The ideas should also have matured to some extent since 1991.

The presentation will be structured with respect to required capabilities of the simulation environment. Features are motivated with references to typical building simulation problems.

4.1.1.1 Differential-algebraic equations (DAE)

DAE capability was included in the MSE definition in Section 1.3, but some additional motivation may be appropriate, especially since the issue is so often brought to question.

Current commercial continuous simulation systems are still strongly influenced by pioneer numerical techniques for solution of systems of ordinary differential equations (ODE). The CSSL language [Augustin 1967] defined in the late sixties is the base of leading commercial general purpose simulation tools such as ACSL. Modern competitors, such as SIMULINK, VISIM and STELLA, are also limited to ODE-equivalent system descriptions, i.e., no closed loops of algebraic relationships[8] are allowed. They have, nevertheless, managed to capture significant market shares, mainly due to early and skillful exploitation of GUI techniques.

Driven by industrial needs to model mixed static and dynamic systems with DAEs, a few application independent tools with this extended scope began to emerge in the late seventies and early eighties. NEPTUNIX and SANDYS are some examples. SPEEDUP [Perkins 1982] is a (chemical) process plant simulation tool in this category. TRNSYS (first version 1975) and HVACSIM+ are early counterparts in the building simulation field. A notable fact is that special purpose tools for mixed static and dynamic systems have existed since the early sixties. Widely used electrical circuit

[8] Many physical systems contain such loops, e.g., an electrical network without capacitors, a pressure-flow (pipe or duct) network with incompressible fluids. The most common trick to go around the problem is to manually introduce artificial dynamics. The main drawback of this approach, aside from lack of generality, is that it forces the solution algorithm to resolve timescales that are irrelevant to the physics of the problem and that therefore result in an unacceptable computation effort.

simulation tools, such as SPICE , and building simulators, such as BRIS [Brown 1962], further support the industrial relevance of mixed static and dynamic simulation capability.

A fundamental problem with DAEs is that they - in certain seemingly straightforward situations - are very hard to solve numerically. They are then said to have high index. No mathematical definition of index will be given here (several alternative definitions exist, see e.g., [Brenan 1989]). Instead some sample problems that lead to high index will be narrated.

1. The problem of calculating the path in space of a particle with mass, given the forces, is straightforward. However, if instead the path is given and the solver is asked to calculate the forces that gave rise to this path, the problem has high index, and many DAE solvers will have great difficulty with it.

2. Calculating the behavior of a thermal system with two masses and a conductance between them is easy, given the necessary source terms and boundary conditions. But if the conductance is allowed to approach infinity, and the two temperatures become identical, the problem becomes index two.

Although seemingly simple to deal with (using paper and pencil), high index problems have been an important topic for research in numerical methods for several years. In many applications, such as rigid body mechanics, they are more the rule than the exception, and much of the competition between simulation systems deals with ability to handle these problems.

However, a very fortunate effect of using a formal language for model description is that equations can be manipulated symbolically, and this is precisely what is needed to deal with high index DAE:s. If the expression for the particle path is differentiated twice, problem 1 is solved. Similarly, identifying the two temperatures as one avoids the difficulty in problem 2. The Dymola system has in this way been successful on difficult problems in rigid body mechanics [Otter 1993].

Fortunately, high index is rare in thermal and HVAC problems. During the course of the IDA project, high index problems have occurred only a few times. They have not prevented progress in any way and for this reason little effort has been devoted to the topic. Presently, IDA is likely to have the same difficulty with these problems as any other numerical DAE environment. However, the present generation of NMF translators [Grozman 1996] has powerful symbolic processing capability. Symbolic treatment of high index problems is therefore within reach when the need occurs.

4.1.1.2 Algebraic Solution Techniques

An issue of significant practical importance for HVAC problems is the solution of the algebraic part of an initial value DAE problem. Before integration can commence, the algebraic equations must be satisfied, and they must be kept that way throughout the simulation. Many general purpose DAE solvers are ill equipped to deal with this problem; all efforts have instead been devoted to the performance of the actual integration.

Frequently occurring non-linear pressure-flow networks are the primary source of demanding algebraic problems in HVAC applications. As we have pointed out, the equations for multizone air flow, constitute a suitable test problem in this category. Another source of severely non-linear algebraic systems of equations in HVAC applications is refrigeration loops.

Solution of non-linear algebraic equations is easy - if a good quality guess of the final solution is available. It is equally difficult if it is not. In a research situation, the need for a good quality initial guess is less of a problem, since the researcher often will accept poor robustness in this respect and is also able to give good guesses. For mass distributed application programs, the situation is different. It is usually impossible to require any non-trivial initial guess. The typical end user will simply not accept this burden, and the problems may also have such magnitude as to make manual initial value guesses impractical.

In IDA, a comparatively large effort has been devoted to the performance on algebraic problems. We will not go into algorithmic details here. Different angles of the problem have been discussed in other IDA publications [Eriksson 1992] as well as in Papers 3 and 5. Some of the main obstacles that have to be dealt with are:

Disappearing Jacobians Most algebraic solution algorithms rely on an estimate of the system Jacobian[9]. During the course of the algebraic solution, the Jacobian may become singular or nearly singular (ill conditioned), thus making it impossible to continue with the present method.

Discrete Events Real systems generally contain discontinuities. A thermostat may for example decide to switch as the solver is searching for a solution. This creates sharp rifts in the residual surface[10], that may be very difficult to surmount.

Local Minima Similarly, the residual may show local minima that trap the solution algorithm.

To handle these problems in an end user application setting, a range of practical methods have been employed:

Explicit Model Linearization A crude but often necessary method is to explicitly provide a linear approximation to the model. For some models, any linear version may be completely unphysical, but it may nevertheless serve the purpose of effectively generating a user-independent, non-trivial initial guess. NMF has a special function (`LINEARIZE`) for this purpose.

[9] the matrix of derivatives of every equation with respect to every variable

[10] the multi dimensional surface that is created when some norm (estimate) of the equation residuals (a measure of how well the equations are fulfilled) is plotted as a function of all variables.

Algorithm Sequencing A range of methods have been implemented in IDA to deal with algebraic solutions. Several modified Newton methods, including Homothopy methods (incremental loading), and Line Search are available. Furthermore, Jacobian independent methods such as Steepest Descent are employed. An application dependent sequencing of these methods is often efficient.

Analytical Jacobians Analytically given Jacobian matrices have better quality than numerical estimates. IDA therefore allows the model-lab user to explicitly provide Jacobian matrices for selected component models. Such models have until recently been hand crafted in the IDA format, but the latest version of the NMF translator for IDA Solver provides analytical derivatives for variables where they can be computed [Shapovalov 1996].

More work is certainly needed in this field, such as improved scaling of variables and equations, and further hands-on, practical strategies. In fact, this is an area where improvements always will be possible. However, IDA has already a very strong repertoire in this absolutely crucial field.

4.1.1.3 Partial Differential Equations

A common characterization of the class of problems that are generally treated with MSEs is *lumped parameter models*. These words indicate that approximations have been made, and that the model reflects a selected level of resolution. All physical processes that we model as algebraic have in real-life some dynamics, but the dynamics are of a timescale that is irrelevant to the model. The same is naturally true for spatial resolution, i.e., the processes that we model are really best described by partial differential equations. However, PDEs can in general not be solved directly, but must be discretized in space to an affordable spatial accuracy. For some PDE problems, a few million states is not uncommon. These problems must naturally be treated with specialized methods. It is unrealistic to treat anything but comparatively small discretized PDEs with the standard methods of most MSEs. However, working with PDEs at modest resolutions is a frequently recurring task, and the available MSE machinery for these types of problems is of crucial importance to the overall productivity.

Let us look at some building simulation situations, where low-resolution discretized PDEs are necessary. The single most common scenario is to solve the diffusion (heat) equation in one dimension (1D) (see, e.g., the NMF Handbook). This is called for, not only to calculate temperature states in walls, but a sequence of decoupled 1D models is sufficient for many 2D situations. An example is the temperature field in a fin of an heat exchanger. Longitudinal transport in the fin itself can often be neglected. True 2D problems occur for example when media transport tubes are immersed in material. In Scandinavia, common problem types in this category concern floor heating and the piping of district heating (and cooling) systems.

Since IDA and NMF support vectors and matrices, it is comparatively easy to treat low resolution field models. An important feature is then the ability to have model resolution as a parameter. NMF Model Parameters can in IDA be interactively incremented. Many alternative tools are in reality are limited to scalars. In them, even

simple field models become major undertakings, since a change in resolution is not easily accomplished.

Ability to handle vectors and matrices with dynamic dimensions is sometimes erroneously regarded as a petty detail. In terms of the IDA implementation effort, it is however far from a marginal item. We have informally estimated that close to half of the implementation work is due to these features. On the other hand, a large portion of the application projects could not have been done in nearly the same time without them.

Another important side of PDE modelling is of course the integration between MSEs and various specialized field modelling tools. Today, stand-alone tools are generally used for fine resolution field modelling. Computational Fluid Dynamics (CFD) is for example frequently applied to the air flow within and around buildings. Finite Element programs are similarly used for heat conduction problems in, e.g., cold bridges or in the ground. In the long term, it is naturally desirable to create links between MSEs and these different programs. An example of similar work is the integration of a CFD code into the ESP-r tool [Clarke 1995].

The modular architecture of IDA, where module integrity is preserved throughout the solution process is very favorable to such integration. The IDA methodology allows individual modules to operate with their own timestep. Furthermore, modules can utilize special internal module structure (sparsity) if necessary. An example of such utilization has been studied for a finite difference pipe model [Eriksson 1991], but further integration projects have yet to be conducted.

4.1.1.4 Variable Timestep

Variable timestep integration has been the standard in general purpose simulation tools such as ACSL since the seventies. Yet, in building simulation, it is still regarded to be somewhat exotic. This issue is also discussed in Paper 3, but a further key aspect may be worthy of notice.

Physical systems with multiple and complex control laws will at any given time be in a single of several possible states. Switches between states occur at irregular moments in time, and are frequently triggered by each other. NMF modelling of such systems is discussed in Section 5.4 of the NMF Handbook. To follow the time evolution of this type of system and simultaneously the thread of state causality is a demanding task. It is perhaps the single most demanding solver issue, even for variable timestep environments, where the evolution of time can be arbitrarily controlled and repeatedly reversed to explore various alternatives.

If the timestep is fixed and prescribed ahead of time, a solution scheme must be devised to find the right system state at the next point of evaluation. In the special case when the model can only exist in one single state under the given circumstances, the problem of finding the right state among a finite number of discrete mode variables is of combinatorial nature. A scheme for this has been implemented in TRNSYS version 14. However, even if the physical system will always be in a single state, the correct state of any model may in fact be unpredictable, without resolving the actual time evolution of the system.

A simple system comprising a thermostat controlled heater, that heats a thermal mass, which is in imperfect contact with a thermal reservoir (see Paper 3, and ASHRAE NMF Translator installation test case [Grozman 1996]), is sufficient to illustrate this fact. The system will continuously switch between heating the mass and cooling off into the reservoir. The model will have a single mode state, the thermostat on-off control, and a single continuous state, the temperature of the mass. Now, let us assume that we are given the right temperature of the mass, and are asked to predict whether the thermostat is on or off at a given moment in time. Without any knowledge of the immediate past of the system, the task is obviously impossible. Yet, this is in fact the very dilemma that a fixed timestep solver is asked to resolve. For such a simple problem, one may argue that if the fixed timestep is selected small enough, the solver will work fine. This is true. However, for many "natural" HVAC systems the mode logic is much more complex, and switches occur arbitrarily close together.

4.1.1.5 Discrete Time Models

Micro processor based controllers are today in complete dominance. Through sensors, they sample the state of the controlled system at fixed points in time, and then run an internal algorithm to determine actuator action. The process is inherently discrete and although it is often possible to model it using continuous methods, it is obviously desirable to be able to capture the true behavior.

A straightforward way of doing this in a continuous time environment is to fix the timestep of the integration algorithm to agree with the desired sampling rate. This method has been adopted in, e.g., a recent ASHRAE project aiming at the creation of a control testbed for TRNSYS and HVACSIM+ [Haves 1996]. The main drawback of the method is naturally that the controller sampling rate seldom is optimal for the integration of the continuous system.

A more desirable method is to allow the controller to operate with a fixed timestep, while a variable timestep is applied to the continuous part of the system. This functionality remains to be implemented in IDA. The plan is to start this work during the spring of 1996.

4.1.1.6 Delays

So called plug flow models for, e.g., flow in pipes can in some applications be very efficient. To model them, a delay operator is required, i.e., a mechanism to, for a certain time, memorize and then recreate a signal.

No dedicated delay operator has yet been implemented in IDA. This is a planned feature and should not present any fundamental problem. A delay model such as the one in HVACSIM+ is in principle directly applicable. However, a close coordination with the native numerical algorithms is desirable and can be expected to perform better.

4.1.2 Openness

In this section some different aspects regarding requirements on openness will be discussed. Much of this discussion is rooted in Paper 4, which should be read in parallel.

4.1.2.1 Model Transparency

In monolithic tools, the mathematical model is generally presented separately in the documentation. Quite often, essential bits of information have been left out, i.e., it would be impossible to recreate a program with the same behavior based on the given documentation. As with the program itself, the documentation has been written with some typical user in mind, and a user that requires a slightly different view of the implementation may be out of luck.

In the MSE case, it should be possible for every type of user to investigate the model in its smallest detail. Any model equation can be investigated, and, if the user has the appropriate access rights, it can be altered. Similarly, it is possible to individually monitor the time evolution of any variable.

Internal details of the solution algorithms may be hidden from the MSE user. The environment has the responsibility to solve equations within a given accuracy. The exact method for doing this may in principle be concealed. However, for diagnostic purposes, it is often desirable to give the user access to internal steps of the solution sequence as well, e.g., Jacobian and residual elements. Quite often, suspected errors in the solution procedure, in fact, turn out to be errors in the model. If the advanced user has good diagnostic tools at hand, such problems may often be resolved without contact with the developers.

4.1.2.2 Model-Lab Design Tools

Not only inspection of model details is important. The extent at which a user is empowered to manipulate a model is another crucial issue. In Section 1.3.1 we discussed different MSE user types. A pure model-lab user utilizes the environment to develop new mathematical models and to perform experiments on these models. The majority of existing MSEs have been developed to primarily serve this user group. A building designer, on the other hand, will generally have little direct use for a full model-lab environment. The designer is generally not an expert modeller, and requires instead, as we have discussed, first of all access to ready made applications, where simulation problems are described in familiar terms. One might argue then that most model-lab functionality should be removed from an MSE for the designer end user. Ability to manipulate individual equations is hardly motivated for the designer end user. On the other hand, possibility to tailor the structure of the simulation model with respect to the given problem is an obvious requirement. A multizone thermal or air flow model with fixed zone topology would for example be of very limited value. Thus, a suitable amount of model-lab functionality is required. We will briefly present and defend the level that has been selected in IDA. Some necessary background for this discussion is presented in Section 2 of Paper 4.

A guiding principle for IDA application development is the following: Although some users require strong support and guidance by application GUIs, they should always be able to operate the model with full freedom[11], with the exception of reformulation of model equations. The GUI should act as a wizard that helps the user to formulate a suitable simulation problem, but it should never become restrictive. An example that

[11] Changing model parameters, start values, and boundary conditions are examples of such model operations.

illustrates this principle is a recent application called the Pilot, which will be discussed further in Section 6.1.1.1. In this application, a complex heat balance model is generated from a handful of input values given by the user in a single dialog box. The common user will then proceed with the simulation of the generated model, as with any so called simplified tool. However, the more ambitious user may also investigate the result of the automatic model generation and adjust individual data as necessary. Even the topological structure of the model may be changed.

The Multizone Air Exchange (MAE) application that was the main subject of Papers 4 and 5, is another example. Here, more is required from the common user. The application is completely based on model-lab functionality, i.e., any user must be able to build model structure using fixed submodels. (A GUI snapshot and further MAE discussion can be found in Section 6.1.2.) When the basic structure of zones, leaks, and system components has been completed - using the general model-lab machinery - an application dependent algorithm is applied, that checks the structure for inconsistencies, and that sets up a typical solvable simulation problem. Pressure levels are given for the external boundaries of the building[12], and all internal flows and pressures are calculated from this. A common user can accept this problem and proceed with the calculation. Another option is to depart from the base problem, and freely select given and calculated variables, e.g., to find the massflows that are required to maintain a certain pressure level in some part of the system. The latter case requires more from the user, but on the other hand enables the study of many interesting additional cases.

Since March 1993, when Paper 4 was written, IDA development has focused on the creation of a situation where the tools can be tested on real-life applications. Most of this work has been of non-scientific nature, and at this moment we are still in the very early stages of actual application development. However, in the new setting, this development is carried out by industrial actors, and is driven by real end user needs, rather than by academic speculation. A key question is naturally whether the IDA principle of allowing every user (almost) full flexibility, while offering adequate guidance for the inexperienced (or hurried) user, is overly ambitious. If a simpler approach is sufficient, the burden in terms of code size and maintenance of the present alternative will be unacceptable.

This question has been discussed carefully with a number of industrial users during the last year and the answer is (fortunately) that the given flexibility is indeed desirable and worth the additional cost. Next on the development agenda is a more comprehensive IDA-based application for building climate and energy analysis. The specification for this application has been formulated independently by future industrial users, and it clearly calls for model-lab functionality.

In conclusion, although some meaningful applications are simple enough to do without the possibility to edit models interactively, many building simulation applications will indeed require this flexibility.

[12] The given pressures are calculated from wind pressure coefficients and external temperature, all of which are given by the user.

4.1.2.3 Access to Model Libraries

A key factor to rapid development of end user tools is a direct import capability of models from other developers. The ability to reuse models is crucial. Naturally, this depends to a large extent on the existence of standards such as NMF. We will return to NMF in Section 4.2. In present IDA application development projects, roughly half of the effort is devoted to NMF model development, usually based on some existing (monolithic) template application. The modelling part of application development is significantly reduced in the cases when ready-made NMF libraries are available.

Another interesting standard for models is called DSblock (Dynamic System block) [Otter 1992]. This standard is not concerned with the source language for simulation models, but rather the format of the target code. It is a program neutral form for TYPE subroutines, to use the TRNSYS nomenclature. A common description of up to eleven Fortran or C subroutines for a model is suggested. DSblock allows description of ODE as well as DAE based systems of equations, with time and state dependent events.

Neither IDA nor available NMF translators support the DSblock format at this point. However, an NMF translator could easily be developed for this format, and IDA could also with limited effort be configured to call DSBlock models.

4.1.2.4 Product Model Data Import

A similar, but conceptually more complicated, situation exists for the use of simulation applications. To be able to directly import model input data from data-base descriptions of the simulated object is a requirement.

Some initial work in this field is presented in Paper 6. IDA is still lacking any support for such import. This is not only due to our limited efforts in this field, but also to a lack of standards, such as those that have been proposed by, e.g., COMBINE.

4.1.3 Implementation Form

In this section we will discuss some general issues regarding the form and structure of the IDA implementation. A successful MSE implementation must not only provide a well-structured and productive environment for the main developers. It must also accommodate different users, ranging from building designer end users, via model-lab users, to third party application developers. The dependence on non-standardized tools and tools that incur additional cost for distributed applications must be minimized.

4.1.3.1 Distributability

A typical IDA application utilizes a large portion of the IDA kernel, and the amount of application dependent code is significantly smaller than the common IDA runtime system. A fundamental question is whether to package each application with its own copy of the runtime system or to allow applications to share the same. The latter approach is considerably more difficult to realize in the general case, since an end user must then be able to install new applications into his/hers already existing IDA-environment.

The better code size economy of a common runtime system is obvious, and although such considerations are loosing importance these days, they must still be taken into account. The IDA runtime system will generally occupy a few (4-6) megabytes of internal memory. Another, perhaps more important consideration is that some IDA applications will be of a more general nature. They will be able to utilize components, mainly NMF models, from several present applications. The final form for IDA mass distribution has yet to be developed, but it will most likely have three distinct levels:

user type	type of IDA system	need for compiler
end user	• runtime system is attached to a fixed suite of applications • fixed submodel model-lab features available	no
model-lab user	• runtime system is separated from applications • full model-lab features available	simple
application developer	• developer's version, with ability to generate applications that can be distributed	full

Table 4-1. Summary of IDA user types and program versions

4.1.3.2 Portability

Good portability to different platforms is naturally required. For the GUI independent part of the system this can be accomplished rather easily, although the need to integrate with compilers complicates matters. The difficult part concerns the GUI.

A number of toolkits exist that allow a single GUI source code to be portable to all major platforms. The main drawback with this approach is that it is difficult to attain true native look-and-feel on all target platforms. How important is then native look-and-feel for IDA based applications? The answer is: very. Designer as well as advanced end users have, in our experience, low (and rapidly decreasing) tolerance against aberrations from native look-and-feel, and little appreciation for code portability.

The industry standard multi-target GUI toolkit for Common Lisp is CLIM 2.0. Aside from the general problem with native look-and-feel, the present implementations of this toolkit also suffer from size problems. The burden of several extra megabytes of runtime code is, in our situation, another argument against a general approach. Furthermore, it seems as if Windows is in almost complete dominance in the building industry. For these reasons, the IDA GUI is tied to Windows for the time being. A Common Lisp based GUI toolkit, Common Graphics, is used.

4.1.3.3 IDA Implementation Languages

IDA Modeller is implemented in CLOS (Common Lisp Object System), an ANSI-standardized object oriented language. The IDA NMF translator is implemented in straight Common Lisp, and IDA Solver is implemented in Fortran 77 (soon 90). Discussions about language preferences tend to become lengthy. Here we will just touch briefly on some aspects.

In a preparatory language study before the implementation of the OmSim system, it was stated that "a full-fledged Lisp system is one of the most efficient programming environments available" [Brück 1986]. For a number of years, the main drawback was that such a system required either dedicated hardware, so called Lisp machines, or extremely expensive hardware and software. Today excellent Lisp systems are available for a few hundred dollars on PC:s. Furthermore, CLOS has an array of features that makes it attractive in comparison to C++, such as automatic collection of unreferenced objects (garbage) and runtime class redefinition. On the other hand, the main disadvantages of Lisp are:

learning threshold Although most OOP languages take time to master (certainly true for C++), gaining Common Lisp coding proficiency requires a larger initial effort than for most traditional languages. However, the prevalence of environments like AutoCAD and EMACS proves that a Lisp based application development language can be successful.

tool abundance In the last few years, C++ environments and toolkits have grown to almost complete dominance. In consequence, most application development material is available in C++, much less in Lisp. However, it is also apparent that there is room for technical alternatives in an ever expanding market. Visual Basic, Delphi, Smalltalk, Scheme, and even Fortran are still very much alive.

Although the original choice in 1987 of Lisp for the implementation of IDA Modeller was rather ad hoc, we would make the same choice today, were we to start a similar project from scratch.

Dependence on existing factors is a more central consideration for the implementation language for IDA Solver. Fortran certainly leaves much to be desired in terms of language structure. Runtime efficiency is the dominant positive aspect from a language design point of view. This is naturally not unimportant for a pure "number crunch" task. A notable fact is that Fortran usually is one of the very first languages to have compilers implemented on new, inventive, often parallel hardware architectures. It is important for IDA Solver to run well on special processor-intensive machines. However, two other issues completely dominate the arguments in favor of Fortran:

numerical libraries Direct access to numerical library routines is a key ingredient to most large scale numerical development. Fortran is in total dominance for such material. If the solver is written in some other language, automatic translation or foreign function calls are required to access this. Both have significant disadvantages.

numerical experts Numerical analysts are very often Fortran oriented. It is important that the code is quickly accessible to any expert in this field, not only to the main developers.

4.2 Modelling Language Design Issues

IDA, as well as several alternative simulation platforms, are based on machine-readable model representations; the models are regarded as data, rather than being bound to the program code. This makes it possible to collect models into model libraries and provide working mechanisms for model reuse. Usually, a simulation platform is shipped with a library of basic models, and it is up to the user to extend it. However, such libraries are at present strictly tied to the platform itself. Models from other platforms have to be re-engineered and re-implemented into the native format, a laborious and error-prone task. NMF, which is introduced in Paper 2, facilitates model exchange between users of different simulation platforms. NMF development trends are discussed in Papers 6 and 7. The NMF reference report [Sahlin 1996b] and Handbook [Sahlin 1996a] are not part of this thesis, but contain necessary complementary material. Here, we will elaborate on NMF design in the same spirit as we have for IDA.

4.2.1 Background and Objectives

The need for mixed static and dynamic system models was discussed in Section 4.1.1.1. There are many different ways to formally describe such a model. Many systems can, e.g., be expressed in terms of electrical network analogies. The same is true for the Bond Graph formalism. Among experienced simulationists, preferences vary considerably. Most common in all engineering fields is probably still the use of Fortran subroutines, with tool-specific argument conventions, as in TRNSYS and HVACSIM+, or equally non-standardized object message protocols. All such low-level conventions share the fundamental disadvantage of being nearly impossible to process symbolically except for in special cases, e.g., symbolic generation of the derivatives of certain classes of subroutines. Other critical symbolic model transformations, such as index reductions, are quite impossible.

Dymola was proposed in 1978 by Hilding Elmquist as a general modelling environment for dynamic systems using a machine readable equation (DAE) formalism. Expressing model behavior with equations is natural for engineers in most fields. It requires no special training and no manual problem transformation into a special formalism. Bond Graph supporters will argue that physical insight is lost but, on the other hand, DAE-modelling allows straightforward expression of indispensable phenomena that are not encompassed by regular Bond Graphs. Examples are convective transports in media of possibly multiple compounds, control signals, pure delays etc. Similar objections will apply to any high level formalism that makes assumptions about the underlying system being modeled. Thus, an equation based formalism that can be processed symbolically is likely to be an optimal base for a standard with potential use in different domains.

Dymola and later descendants such as LICS [Elmquist 1986] and Omola [Andersson 1990] have not been widely used until recently. Aside from a range of practical issues, the main reason for this is most likely the previous lack of robust and efficient solvers for general DAE-systems with discrete events. Dymola has in the last few years had a revival and is now a popular, but proprietary, language and modelling environment for a broad spectrum of simulation tasks.

A similar language that was developed in direct conjunction with a full MSE environment is the SANDYS language that was introduced in the early eighties [ASEA 1983]. It has good mechanisms for description of discrete events, but has - lacking ways to bundle variables in component connections (cf. NMF links) - less developed facilities for modelling at a higher abstraction level, i.e., without direct knowledge of model content in terms of individual equations and variables.

The ESACAP language is originally based on electrical network analogies, but also allows general DAEs. No link concept is available here either.

Yet another equation based language is ULM (Un Langage de Modélisation) under development at Gaz de France. ULM has a similar scope as NMF, but with more emphasis on the separation between the (pure) mathematical model, and the numerical (technical) manifestation of the model in a particular solver.

A specialized language for digital electronic circuit description is VHDL [Perry 1991] (VHSIC Hardware Description Language). A project supported by IEEE is proposing an extension of VHDL to describe the structure and behavior of analogue circuits and systems as well. An extension based upon a large collection of requirements, VHDL-A, is being developed by a special group in IEEE since 1993. It is expected that results should be merged in VHDL around 1998. This language will support hierarchical decomposition, differential or algebraic equations and different abstraction levels.

NMF is much inspired by Dymola and its descendants, but has a different basic objective: To be a source language for large common libraries of validated and documented simulation models. It is not primarily intended to be effective for a single user or a small group, working with a single simulation tool.

Main NMF objectives are:

Multi-environment compatibility	as proven by the existence of translators to a range of different tool-specific formats.
Engineering usability	NMF expresses not only the model itself, but also relevant information *about* the model, such as range of validity, variable meanings and units.
Neutrality	The evolution of NMF is presently governed by an open ASHRAE committee.

A frequently raised objection against NMF is due to a misunderstanding of the intended scope. NMF is not a complete modelling language. It does not describe all the information that is necessary to run a simulation problem in any of the target environments. Some essential things that are missing are ways to connect component models into system models and ways to describe simulation data, such as start and stop times, tolerances, driving input functions, variables that are to be logged etc. All such information must be described in the format of the local simulation environment, i.e., NMF is a complement to the local format - to facilitate model exchange - not a replacement. There is no fundamental problem to extend NMF to cover more of what

is needed of a complete language. Indeed, such work is underway, e.g., as described in Papers 6 and 7. However, at the same time, it is important to recognize that the present NMF covers a very high percentage of what is needed for model exchange between environments. The need to exchange entire simulation problems is far smaller than that of sharing component models.

The discussion about description languages for dynamic systems is interesting. Some language proposals suggest aesthetically pleasing ways to describe systems, with little concern for practical usefulness, in terms of available solvers that can interpret the description. The boundary conditions for NMF have been chosen to be restrictive. They are defined by the proven capabilities of several existing simulation environments, and not just a single. It is too restrictive to require that any model can be treated to satisfaction in all target environments. Naturally, the abilities of IDA have had some impact on the NMF design. However, and this is important, every effort has been made to keep NMF a neutral model description language that is rooted in state-of-the-art solver technology.

In comparison with the development of regular high level programming languages, equation based model declaration languages are still in their infant stages. It is not unreasonable to expect that a similar degree of heterogeneity will develop in this field, due to differences in taste, functionality, and target audience. A single standard, that satisfies all needs and preferences, is unlikely to ever develop. A range of languages with different profiles will be needed to serve the various user categories.

In the next few sections we will discuss some basic NMF design considerations, and some planned extensions. Headings have been selected to indicate our view of suitable issues in a language evaluation project.

4.2.2 Syntactical Structure

A successful language must be aesthetically appealing to its target audience. Computer scientists are rarely pleased with the look of NMF, while engineers equally often are. From a compiler designer’s viewpoint, NMF is a bit odd. No clear principles prevail in terms of necessary parser look-ahead. The BNF (Backus Naur Form, see Section 5.2.1 of the NMF Handbook) description of the syntax is quite complicated. Prefix, infix, and postfix operations coexist. Naturally, it would be best if all groups could be satisfied. However, if a choice has to be made, as we believe, the end users are given preference in NMF.

The Matlab language - designed by a numerical analyst, Cleve Moler - is an interesting example of another language that has been tailored for end user satisfaction. The expressive power of the language coupled with the underlying Linpak numerical methods has formed a winning combination. For a compiler designer, the language is less appealing. It is unclear whether it is at all possible to express the syntax in BNF.

NMF has a BNF description and is processable by staple compiler tools such as the standard Unix tools LEX and YACC. The SPARK translator [Nataf 1995] has been developed in this setting. The programmable ASHRAE translator [Grozman 1996], uses automatic parser generation in the same spirit, but has been implemented in Common Lisp to facilitate easier inclusion of computer algebra functionality.

4.2.3 Expressiveness

The set of phenomena that can be described with a given language is naturally the first thing to look at. The scope of the present NMF is a balance between need and available resources. In present NMF, the main model type is the `CONTINUOUS_MODEL`, which we will discuss at some length. We will also briefly cover two other model types, algorithmic and field models, formats of which will be suggested in the near future.

4.2.3.1 Global Declarations

A fundamental NMF design decision is to use explicit units for model quantities. An NMF model is always written with respect to a particular set of units. The basic motivation for this is that many engineering models are expressed in terms of unit-dependent empirical laws, often in quite utilitarian units, including IP (Inch Pound) based unit systems. To exclude such models or to treat them as exceptions would be in conflict with the engineering usability objective.

In NMF, the modeller is free to select any mixture of units for a particular model. The idea is that the existence of influential model libraries in certain units will provide an incentive to develop further models in compatible units. In engineering practice, also in SI dominated territory, slightly inconsistent unit families are often used, e.g., `kWh` is a common energy measure, `l/s` or m^3/h is often used for volume-flow. In consequence, NMF models frequently contain internal conversion factors to accommodate for such inconsistencies in the chosen set of units.

A common view is that the numbers that are presented to the user, always should be filtered from those that are actually being processed, and that such a separation facilitates usage of more consistent units in the mathematical model description. What is suggested is a model formulation that allows a complete change of unit system without alteration of model equations. In consequence, the language should require usage of “pure” unit systems such as SI or no units at all, just quantity dimensions. Aside from the problem with empirical models as previously mentioned, two serious objections against this view can be raised. First, for practical application it requires that all compatible environments contain a complete filtering layer. This puts severe restrictions on the number of target environments that can be accessed in practice. Secondly, although it is desirable to have such a filtering layer for certain quantities, also for NMF models, it is confusing with too much filtering from a numerical diagnostic point of view. It is unavoidable that investigation of detailed solver action is required in application development, and the efficiency of this process is a real bottleneck which has to be taken into account in the language design.

One should also recognize that the choice of units for a model can be altered with automatic means at a later stage. The original choice is just a well defined base case, from which any conversion can be made, either symbolically, using computer algebra, or numerically, using conversion factors in the target environment.

In the present NMF version, there are some omissions that complicate automatic unit conversion and consistency checking. The unit declaration string is only informally standardized, and NMF function output is presently untyped. Both these areas will be covered in future releases.

4.2.3.2 Continuous Models

NMF `CONTINUOUS_MODELS` express general systems of non-linear differential-algebraic equations. Equations may contain time and state dependent events. Variables and parameters may be vectors and matrices. Field dimensions may be user controlled parameters, so called `MODEL_PARAMETERS`, and the fact that these may be changed interactively in an instantiated model, gives rise to some syntactical limitations on the way indices may be used in the code. It would, for example, be difficult to handle links with matrix structure. In consequence, there is a limitation in NMF to vector links. The need for field variables as such has been discussed in Section 4.1.1.3; some commented examples of actual models can be found in the NMF Handbook, Section 5.1.

The need to change field dimensions interactively - without re-instantiating and thereby re-parameterizing the model - in an interactive modelling session might need some illustration. Situations where this is convenient is, e.g., when a model has a flexible number of ports[13] (vector links) and the user decides to connect an additional component, or when the resolution of a field model needs refinement. Most alternative languages lack working support for field variables at all, and we have not yet seen any implementations that can be compared with IDA/NMF in terms of interactive abilities. Dynamic fields have major impact on many language constructs, and their value should be accounted for in the comparison of different languages.

Another NMF choice with significant impact on overall structure is the possibility to declare additional information about equations and quantities. Examples of such information items are `GOOD_INVERSES` for equations, and `IN` or `OUT` role for variables. With the `EXTENSIONS` construct that is proposed in Paper 7, any type of information may be added in this way. Examples of useful such information - for a single or a range of target environments - is an explicit inverse to an equation, and labels for equation and variable category[14].

A consequence of the possibility to label equations is that it makes it awkward to allow so called conditional equations, i.e., multiple versions of equations delimited by `IF-THEN-ELSE` constructs. An example with conditional equations could be:

```
IF a < b THEN   /* NOTE! THIS IS NOT CURRENT NMF */

  x + y = f(z);   /* equation 1, case 1 */
  x1 + y1 = g(z); /* equation 2, case 1 */

ELSE

  x = z;          /* equation 1, case 2 */
  x1 = z;         /* equation 2, case 2 */
```

[13] An example of such a model is a zone model to which a flexible number of wall, window, and light fixture models might be attached. A storage tank with a flexible number of pipe terminals is another example.

[14] For many problem types it is advantageous to partition the problem into a sequence of uncoupled or loosely coupled subproblems, that may be solved sequentially. An example is a mass-pressure core problem, with a number of subproblems for transport of various substances through the system that can be calculated rather independently.

```
END_IF
```

In present NMF, this code would be written with conditional expressions instead (`GOOD_INVERSES` declarations have been added to illustrate equation labeling):

```
x =  IF a < b THEN
        - y + f(z)
     ELSE
        z
     END_IF GOOD_INVERSES( x, z); /* equation 1*/

/*  The z good inverse declaration assumes that the modeller
    knows that f and g are well behaved*/

x1 = IF a < b THEN
       - y1 + g(z)
     ELSE
       z
     END_IF GOOD_INVERSES( x1, z);  /* equation 2*/
```

The latter construction makes it easy to keep track of defined equations, and any associated extra information. For models with many alternative modes, it may however become awkward.

Another current NMF choice that can be questioned is that assigned variables are illegal on links, i.e., variables that have received their value through assignment, may not be visible on the boundary of a model. A consequence of this restriction is that models, that are just wrappers to subroutines, must contain trivial equations, the sole purpose of which is to equate subroutine output with link variables. An example of such a model is discussed in Section 5.3.2 of the NMF Handbook. The full consequence with respect to model clarity of changing this rule is difficult to determine. The simplicity of the current rule, that the number of equation signs must agree with the number of `OUT` variables, is perhaps more valuable than the compactness that would be attained by avoiding the trivial equations.

Whether assigned variables should be allowed on links or not is from one perspective naturally a petty detail. On the other hand, it is becoming increasingly clear that the pedagogical issues regarding NMF are perhaps the most important of all. For many physical systems, it is genuinely difficult to formulate well posed DAE models that cover all possible cases. Even experienced modellers get confused among large numbers of equations and the various operative modes that a model may be in. From this perspective, the traditional way of writing models - as subroutines with given input and output - have some merits. At least they have a very clear and easily explained purpose: to calculate outputs, given inputs. The alternative task, to declare a sufficient number of “truths” in terms of equations for a component is much more abstract and therefore more difficult to explain.

The NMF way of writing an equation based component model - with local assignments, and explicitly declared `IN` and `OUT` variables - allows a user to think as if it was a subroutine that was being written, with a single important exception: No solution algorithms for implicit systems of equations have to be hand-coded. This NMF repertoire, which sometimes is criticized by equation puritans, may actually prove to be easier to handle for beginners than a pure equation oriented description. Clearly,

more work is needed to optimize the pedagogical aspects of all declarative equation based languages.

4.2.3.3 Algorithmic Models

Although equation based model descriptions are preferable for most physical components, they are not optimal for modern digital controllers. Sampling controllers are causal and algorithmic in nature. They read input signals, run an internal algorithm, and produce output signals. Controller sampling is generally governed by an internal clock. However, the basic concepts of a controller can be useful also for other processes if the procedure of generating a sampling event is generalized to be an external signal. For a fixed timestep controller, the external signal would simply be a clock. Let us call the more general model an algorithmic object.

The NMF definition of algorithmic objects has not yet been finalized, but it can be accomplished with very similar constructs to those of `CONTINUOUS_MODELS`. It is natural to retain the concept of assigned state variables (`A_S`) and to require that `OUT` variables are calculated by assignments from `IN`, `LOC`, and `A_S` variables. `LINKS` should be allowed to contain `IN`, `OUT`, and `VOID` variable positions. `VOID` being positions that are neither read from nor written to by the algorithm.

Aside from controller models, algorithmic objects would in building simulation be useful for, e.g., weather data processing and on-line post processing (often just summation) of simulation outputs. The sampling events would then coincide with the continuous integration timesteps.

4.2.3.4 Field Models

Another interesting model type in the slightly longer perspective is some framework for pure PDE models. Several levels of generality are possible. A reasonable choice could encompass parabolic equations that are discretized by finite elements. For the computational grid, the most straightforward alternative would be to require an explicit description. However, automatic meshing is a rapidly evolving technology, and it might be possible to base a model format on this. In this case, only the computational domain would have to be described.

Field models are a natural NMF extension, but do not appear on the immediate development agenda. It is also rather clear that, except for the actual equations, the source code of a field model would be rather uninteresting. Domains and meshing would naturally be both generated and interpreted by special tools.

Other inherent limitations of a text based source code language have been discussed in Paper 6. Large system models, e.g., are also awkward to describe and interact with in text based form. The balance between textual language descriptions and data base ditto is another interesting issue that in the future is likely to divide different approaches from each other.

4.2.4 Level of Standardization

Although expressiveness certainly is a key factor for language evaluation, some other aspects have to be taken into account as well. The second most important consideration is probably level of standardization. A language, the evolution of which lies

completely in the hands of independent developers, is a high-risk option for any potential user or independent tool developer.

Presently, NMF is governed by an ASHRAE committee. This is an interim measure pending a formal standardization process. However, it serves the practical purpose of ensuring that the language is reasonably stable, and that proposed extensions are introduced in a controlled fashion.

One should also recognize that the very fact that a language is computer processable represents some security. If the information content is sufficient, the language can often be automatically translated into an alternative format. An example is the generated code for TRNSYS types. If NMF died and a TRNSYS user had a large number of models in NMF form, the generated TRNSYS source code version of these models would be an excellent base for further work, since the structure, comments, naming conventions etc. of the generated code is of better quality than that of most hand written models.

4.2.5 Introduction Threshold

Another consideration is the start up time for beginners. Mastering the full extent of a language may often take some time, but to write and fully understand the first few models should not. Two things that should be taken into account are:

- Initial test models should not be too simple. They should reflect a reasonable real-life situation in the intended area of application.
- Differences in information content of languages should be given special consideration. Many languages attain compactness by omitting certain information, or by treating it as comments, which are difficult to process automatically. This will have significant impact on the translatability of the language. Units and variable descriptions are obvious such examples. A less obvious thing might be variable `CROSS` or `THRU` status, which in some languages, e.g., the SPARK native language, is treated implicitly, in the way equations have been formulated. This is equivalent to having `CROSS` type for all variables.

4.2.6 Translatability and Openness

Interfacing with external programs is of course another key NMF issue. Translatability to various target formats is the most important side of this question. However, a less discussed side concerns the ability to interface with subroutine based models. Access to such models is important for primarily three practical reasons:

1. Key parts of a model may be hidden in binary form for commercial purposes.
2. Large existing packages can be accessed with a minimum effort, without re-writing and re-debugging.
3. Efficiency can often be gained by using a tailored procedure for particular models.

The technical details of NMF subroutine calls are discussed in Sections 5.3 and 5.4 of the NMF Handbook. Here, we will concentrate primarily on the underlying motivation

for the NMF design with respect to both foreign subroutine calls and translatability. The structure of the discussion will be based on some specific NMF constructions.

The interface issues discussed in this section conclude the presentation of suggested language evaluation criteria. One should recognize that the chosen interface characteristics have profound impact on the language design, both in overall structure and in details. The degree of translatability to various target formats is a difficult and interesting subject that we will only be able to touch briefly upon here. Obviously, the design decisions of the individual code generators for different target environments must be examined for a more thorough treatment of the issue.

4.2.6.1 Vector to Scalar Mapping

The issues of dynamic field dimensions - NMF model parameters - have been discussed in Sections 4.1.1.3 and 4.2.3.2. They must also be considered in the treatment of translators. In the simplest case, when the target environment lacks field variables, model parameters must be given values in the translation process, and corresponding scalar code is generated[15]. In this case, model versions must be generated for each interesting combination of model parameter values, or the translator must be integrated into the modelling environment. The latter alternative is naturally the most attractive.

However, if the target environment supports dynamic field variables, it is essential to make sure that the chosen translator preserves this flexibility. Such support is one of the most difficult issues in translator development, especially for translators with computer algebra capabilities, since operations sometimes must be done on subsections of fields. General purpose computer algebra tools such as Mathematica do not support the required field operations.

The TRNSYS and HVACSIM+ solvers can handle dynamic model parameters, and the ASHRAE translator generates proper TYPE routines to support this. However, the present modelling tools for these solvers generally do not support dynamic model parameters. This is a serious disadvantage of, e.g., HVACGEN and PRESIM, which limits their applicability (also for native models, such as building models with flexible number of zones). The ASHRAE translator supports HVACGEN, but with fixed model parameters only. All current IDA translators [Kolsaker 1994c] [Grozman 1996] support dynamic model parameters without restrictions.

4.2.6.2 Assigned States

Models with hysteresis cannot be expressed with equations alone. Some mechanism to memorize past system state must be added. NMF's solution to this problem is the introduction of a special variable type, the assigned state (`A_S` in NMF code, see Section 5.4 in the NMF Handbook). The value of an assigned state is retained between model equation evaluations. A basic motivation for the chosen construction is that assigned states can be passed as memorized workspace to external routines. This significantly increases the domain of applicability for NMF. The assigned states design

[15] Each vector or matrix element becomes a scalar in the target code, e.g., *x*[1], *x*[2], and *x*[3] are mapped to *x1*, *x2*, and *x3*.

is a fundamental choice that has profound impact on language grammar, translatability, and openness to foreign routines.

The IDA, TRNSYS, and HVACSIM+ translators fully support NMF assigned states. To the best of our knowledge, the SPARK and ESACAP translators also support or intend to support them. Most real-scale solvers have some mechanism that can be used to store values as required for NMF assigned states.

4.2.6.3 Event Signals

Explicit signaling of discrete events allows a continuous solver to control the timestep and solution method in order to resolve system behavior to the desired level of accuracy. This can have considerable impact on robustness and also on efficiency for models with discontinuities. The introduction of multiple mode models by assigned states and event signals actually enables NMF description of completely event driven systems[16]. Robust solution of mixed continuous and discrete models is a permanent challenge for all solvers. Not only is the solution difficult; it is also demanding to formulate mixed models that are mathematically well posed in all cases.

Current NMF is limited to real variables. Any other choice would severely limit the range of target solvers. This puts some restrictions on the effectiveness of event signaling between submodels. However, as target solvers are developed, it is natural to allow discrete variable communication at some stage. In contrast, a simulation environment such as OmSim - developed in conjunction with a dedicated language (Omola) - has better potential to treat mixed continuous and discrete models.

IDA has complete support for the localization of discrete events, as signaled in the NMF code. This was first described in Paper 3, in 1991. TRNSYS, HVACSIM+, and SPARK lack, to the best of our knowledge, support for internal events. HVACSIM+, being a variable timestep environment, would be comparatively easy to extend in this respect, since the basic ability exists to reverse propagation of time.

An important aspect of the NMF event signaling mechanism, by calls to special functions, is that it may be employed to its full extent also in external routines. It should also be noticed that short forms of the present event syntax can be constructed to attain compactness. These can then automatically be expanded into the present form in the translation process.

4.2.6.4 Model Linearization

The need for explicit model linearization was mentioned in Section 4.1.1.2. It is also discussed in Section 4.5.1 of the NMF Handbook. Similarly as for event signaling, explicit linearization is expressed with calls to NMF special functions, that may or may not be supported by a particular target solver. The main advantage of this strategy is, again, that linearization can be done also in externally defined routines. A problem is that multiple calls to the `LINEARIZE` function obscures the real non-linear equations in the code.

[16] The dynamical behaviour of event driven systems is generally described by dedicated methods such as Petri nets.

An alternative that has been discussed [Lorenz 1994b] is to declare a completely separate linearized version of the `EQUATIONS` section in a separate main section. The main advantage of such an approach would be to gain model readability. Two objections to this approach are:

1. Only a single level of linearization would be possible in practice, since multiple levels would require a complete model description for each level and this would quickly become cumbersome. To introduce model non-linearities successively in the solution process is a very effective method for many difficult algebraic problems.
2. Many model equations are already linear. A separate section would make it necessary to repeat these, thereby introducing a double source problem with all its disadvantages in terms of manually maintained consistency.

The lack of explicit linearization mechanisms in many of the alternative languages highlights the fact that these are mainly targeted at advanced users working with pure model-lab MSEs.

Presently, IDA Solver is, to the best of our knowledge, alone to utilize the NMF linearize mechanism. However, adding such support should be easily accomplished in all implementations that have any special treatment of initial value calculations. The importance of such treatment in building simulation end user applications has been discussed in Section 4.1.1.2 and in Paper 5.

4.2.6.5 Foreign Functions and Subroutines

The possibility to access foreign functions, without any limitation on the types of models that are packaged in them, is a pronounced NMF design objective. Clearly, the many advantages of having models in processable form are lost for such models.

Openness to foreign routines has influenced the design of many NMF constructs. Limitations in this support, that are introduced by a translator or by a target environment, will have significant impact on the number of NMF models that can be accessed. Full support for subroutine calls, i.e., multiple input - multiple output objects, can be difficult to implement in solvers that are originally designed to operate on individual scalar equations.

The IDA, TRNSYS, and HVACSIM+ translators have full support for foreign routines. If user defined routines are written in the NMF algorithmic notation, or in Fortran 77, the translation process is automatic. For external C code, the foreign function call must be completed manually by the user, since there is no standard for such calls. It is unclear to us to what extent the present SPARK and ESACAP translators support subroutine calls.

This concludes our discussion of NMF translator issues, and we will briefly turn the attention to another type of NMF use, as source language for special purpose tools.

4.2.7 Translator Development for Special Purpose Tools

Until now, the discussion here, as well as nearly all NMF related research and development, have dealt with general purpose simulation tools, i.e., tools which are built to describe and simulate a very wide class of physical systems. However, as we have also pointed out, tools with a more limited scope are generally more useful to designer end users. Many existing such tools have some degree of modularity and it is not uncommon that they have a local component model format that allows significant flexibility.

Large amounts of engineering effort are devoted to model development for these tools, which often are tied to a physical product line. Common examples in the building sector are sizing and system selection tools for air handling units, chillers, boilers, and virtually all other HVAC components with non-trivial internal behavior. A similar situation exists in many other industrial fields. We will briefly discuss the applicability of NMF and translator technology to this class of tools.

A typical situation is that the special purpose tool prescribes a certain selection of variables with given units that are communicated between submodels. This corresponds to a single or a few NMF link types. Models are often static and contain only scalar variables. Most environments in this category are input-output oriented, with a prescribed component call convention, i.e., like a limited version of the TRNSYS TYPE format.

Some development bottlenecks for such systems are:

1. Extension of component models is often time consuming, since various solution methods, often based on successive substitution, are intertwined with the model equations.
2. Inclusion of new models requires complete re-implementation into the designated format.
3. Changing the designated format, e.g., to make it more general, or to add information that is required for a new solution technique, involves manual treatment and testing of the full library.

An NMF based component library with a dedicated translator would be helpful in all of these situations. Although export of component models in source code form from such proprietary libraries rarely will be an issue, being able to import models from public libraries should be attractive, in spite of the fact that models often will have to be manually treated to fit the fixed local link structure. Export of models in some encrypted form for inclusion in, e.g., product models is another interesting perspective.

4.2.8 NMF Discussion

A number of key NMF design considerations have been mentioned. A more exhaustive discussion - of NMF in relation to comparable languages such as Dymola, Omola, and ULM, and of translator issues to the large number of possible target formats - is of course also desirable. We hope that the presented list of issues will be an inspiration for numerous future contributions to this exciting field.

NMF has a distinct and pragmatic profile. As we have mentioned, many alternative profiles are possible. However, a conclusion that can be drawn is that any language that is to be used as a base for common model libraries, must be rooted in the solver technology of today. Such a language must be translatable to most major platforms in its field of application. Otherwise, the incentive to contribute library material will be limited, as will the possibility to utilize what is available.

An analogy can be drawn with Fortran and Lisp, both of which were conceived in the early days of computing. Fortran was designed to replace cumbersome low level languages and it became useful immediately. Lisp, on the other hand, was the result of an academic exercise in the application of so called lambda calculus. It took close to thirty years of computer development to make it practically useful[17]. Had Linpak, NAG, IMSL and similar libraries been implemented in Lisp, they would have been doomed to a life in seclusion.

The theoretical discussion of the aesthetic aspects of modelling languages for dynamical systems is of significant long term interest. It must, however, not be allowed to prevent the development of libraries within the boundaries of present technology. The tendency to postpone the actual library building, while waiting for the next language to be completed, is potentially harmful to the credibility of the whole idea of independent model libraries.

[17] It is still common with conference titles that end with "... on the *Practical Application of Lisp*."

5. Comments to the Papers

In this section, we will attempt to connect the objectives of this thesis with those of the individual papers. For some of them, this will require some additional discussion.

The papers have been written over an eight year period and are presented in chronological order. The authors' maturing understanding of the subject and evolving terminology is easily noticed. This inconsistency can obviously be confusing, but may not always be pointless. Other researchers seem to have followed a similar path of enlightenment, and having the history spelled out, might help in avoiding the repetition of some mistakes.

For the papers that have been co-authored, the author's level of contribution will be indicated.

All of the papers have been presented at conferences. They have not been submitted for publication elsewhere. At the time, this did not seem important, since they reached their target audience rather effectively and stimulated the discussion. All but Papers 1 and 6 have appeared in the IBPSA conference series and proceedings. All but Paper 6 have been subjected to independent review.

5.1 Paper 1: MODSIM - a Program for Dynamical Modelling and Simulation of Continuous Systems

Paper 1, presented to the Scandinavian Simulation Society in 1988, was written less than a year into the IDA project, which then bore the name MODSIM. Several name conflicts, e.g., with the HVACSIM+ internal solver, made us later settle for the name IDA. It originates from "ITM's[18] Differential-Algebraic modelling and simulation environment." The paper gives an overview of the project and the ideas, most of which have actually survived. The research implementation of IDA, which is discussed in several of the subsequent papers, followed the basic design that was outlined in Paper 1. IDA Modeller is currently under revision, to better accommodate new GUI and OOP standards, and to enhance the graphical repertoire in a way that is necessary for a commercial product.

In the paper, a first in-house demo version is projected for the summer of ´88. This stage was not in reality reached until February '89, and the program was demonstrated rather widely in the spring. This first version lacked support for field variables and internal events, and the only algebraic solution method was a full Newton method. The demo version had been implemented in some two and a half man-years. It is somewhat discouraging to realize that the remaining 80-90 percent of the project has been largely devoted to consolidation of the first demo version. Successively more difficult application problems have driven the development of a more comprehensive range of methods. In Paper 1, the remaining work to a "full commercial quality system" was estimated at between four and eight man-years. Discounting time spent on funding issues, this is perhaps not more than a factor two off.

[18] The (Swedish) abbreviation for Institute of Applied Mathematics.

Paper 1 provides the only overview of IDA that is offered in the framework of this thesis. Another overview was written in 1991 [Sahlin 1991]. The current implementation will be presented in the ensuing year.

5.2 Paper 2: A Neutral Format for Building Simulation Models

Paper 2, written in the spring of '89, provides another early point of reference. The text was the result of nearly a year of discussions at ITM, which was fortunate enough to co-host Prof. Ed Sowell during a sabbatical. In the initial stages of the IDA project, the importance of concise and processable component model descriptions had not yet been fully realized. Components were coded directly in Fortran and Lisp, in a similar way that is done for, e.g., TRNSYS and associated modelling tools. The shared need - with the SPANK project (now SPARK) - of comprehensive model libraries, made us look closer at the DAE-based languages DYMOLA and LICS, as a possible base for such libraries.

Paper 2 also existed in a longer version, which did not fit in the IBPSA format. This version has subsequently been updated and amended several times and is now the NMF reference report [Sahlin 1996b].

Slightly more than two thirds of the text is written by the author of this thesis, the rest by Ed Sowell. Equal and inseparable contributions to the ideas have been made also by Magnus Lindgren, Axel Bring, Lars Eriksson, and Gustaf Söderlind.

5.3 Paper 3: IDA SOLVER - a Tool for Building and Energy Systems Simulation

Paper 3 was presented in '91 at the IBPSA conference in Nice, France. It was motivated by a lack of understanding of actual solver capabilities that often is revealed in academic work on modelling and modelling languages. Many groups concentrate on modelling issues alone without close contact with solver developers. Although concentration certainly is necessary, this often leads to a considerable gap between as-designed and actual performance. The paper was an effort to popularize some key solver issues that must be taken into account also in the development of modelling tools. Apparently, we had some success in this, since the paper gave rise to a range of fruitful discussions and contacts.

Roughly equal parts were written by Axel Bring and the author of this thesis.

5.4 Paper 4: IDA Modeller - a Man-Model Interface for Building Simulation

Paper 4 was one of two IDA papers that were presented at the IBPSA Adelaide conference in '93. It attempts to start a discussion about the fundamental characteristics that are desirable for a modern MSE for building simulation. This discussion is continued in Section 4.1 of this thesis.

5.5 Paper 5: Modelling Air Flows and Buildings with NMF and IDA

Also presented in Adelaide, Paper 5, discusses the multizone air flow problem. As we have pointed out, future building simulation software must be able to predict multizone air flow coupled with thermal performance. In this general area, several papers have been written, reporting on a number of projects where thermal programs have

been merged with monolithic air flow codes. In our view, the success of such attempts bear little hope for the future, as we will motivate further in Section 6.1.1.2.

Axel Bring has written the slightly larger part of the paper. The NMF models are almost completely his work, but they are based on an idea for bi-directional flow models by the author.

5.6 Paper 6: NMF-Based Aspect Models in STEP/EXPRESS for Building and Process Plant Simulation

Paper 6 reports our initial work in the area of product models, and their relation with MSE based simulation tools. The paper was presented at the CIC W78 workshop on computer integrated construction, in Helsinki, Finland, 1994, for an audience of mainly product model researchers with limited simulation background. A similar paper [Sahlin 1994] was also presented in the framework of an EC expert group, ESPRIT WG 8467, with the purpose to discuss future needs in the simulation field (not especially building simulation). Both papers were well received, which seems to indicate an understanding from both sides of the necessity of cross fertilization.

The work was done in co-operation with, Curt Johansson, a student of Prof. Bo-Christer Björk, at the Division of Information Technology in Construction at the Royal Institute of Technology. The text is written by the author of this thesis. The EXPRESS models were designed in dialogue. They are similar to internal data models of IDA Modeller.

After a delay, the work of Paper 6 is now to be continued.

5.7 Paper 7: Future Trends of the Neutral Model Format (NMF)

In Paper 7, attention is returned to the text based NMF grammar. The work was presented at the IBPSA Building Simulation ‘95 conference in Madison, Wisconsin. An attempt is made to synthesize several years of discussion and NMF proposals that have been made since the presentation of Paper 2 in 1989. However, little space is sacrificed for background information and introductions and this makes the presentation rather terse for a newcomer. The paper lists six concrete proposals:

1. Hierarchical Modelling
2. Hybrid System/Continuous Models
3. New Function Definitions
4. Property Links
5. Model Inheritance
6. NMF Extension Keyword

All of these proposals require significant amounts of additional work in order to assess their real practical applicability. The ASHRAE NMF committee has, at this point, approved the main principles of the first two proposals, and we will start to work on test implementations of these. The last proposal is also the easiest to accommodate in existing translators, and a slightly modified version of this has already been

implemented in the present ASHRAE translator. Proposals 3 to 5 are still very much in the discussion stages.

The text of Paper 7 is written by the author of this thesis. The code examples are the work of Axel Bring, Kjell Kolsaker, and the author. Several of the underlying ideas are due to Kjell Kolsaker and other active independent NMF supporters.

6. Discussion

In this section, we will discuss the presented results in view of the main objective of this thesis: to determine the applicability of MSE based technology as a replacement for traditional program design. The discussion will be carried by a presentation of some projects in the primary application areas that were defined in Section 1.4.1.

6.1 Summary of Relevant Application Projects

The projects that will be presented have been selected with respect to the primary target applications. They have not been carried out for the purpose of this work. Some of them are complete application development projects, while others are small studies for the investigation of some physical system.

The comparison between traditional special purpose methods and the MSE approach is complicated mainly because relevant evaluation criteria are difficult to establish. The main advantages of most specialized methods are speed and robustness, while MSEs feature flexibility, maintainability, transparency and other alternative qualities. In this discussion we will concentrate on those that are directly comparable.

6.1.1 Building Loads and Energy Calculation

The first primary target application is also the most critical, since the absolute majority of existing BPE tools have been developed for this purpose. MSE based technology can be applied in various ways to this problem area. The most straightforward way is to break the building into suitable component models and treat them with the standard MSE methodology. It is also possible to sacrifice some general MSE advantages for the fabric part of the problem (the zone model) in order to improve overall performance. A further complication is that the preferred balance between accuracy and speed varies considerably between the engineering cultures of different countries. For example, in the United States, it is relatively common to make hourly simulations for a full reference year on a fifty zone building. In Scandinavia, on the other hand, the tradition is rather to study the behavior of a few key zones with more detailed, often non-linear, models. Vast differences in weather, building tradition, and internal climate preferences also exist, and these are reflected in the local tools.

Let us concentrate on three questions here:

1. Given the other advantages of the MSE approach, what is the relative performance with respect to speed and robustness for a true MSE implementation of a detailed heat balance zone model?
2. What can be said about the relative development times for this case?
3. Can previous investments in implementation, user training, and documentation be retained by keeping existing monolithic tools as a part of a future simulation environment?

6.1.1.1 Re-implementation of BRIS

Regarding the first question some relevant work has recently been done at KTH by Mika Vuolle, Axel Bring, and Jan Akander [Vuolle 1996] [Akander 1995]. This work

also builds on results from a project at Chalmers Univ. of Technology [Ljungkrona 1994]. The task has been to develop a set of NMF models that can be connected into zone models in IDA or in some alternative NMF compatible environment. Various versions of the following main models have been implemented:

- a single air temperature zone model (excluding walls) with full non-linear radiative and convective heat transfer, using a general view factor calculation algorithm
- various state-of-the-art window models
- full finite difference and reduced RC-network wall models
- solar radiation calculation models
- local climate control models

For the comparison of performance, versions of these models were especially adapted to replicate the models of the most commonly used Swedish corresponding tool, BRIS [Brown 1990].

An extensive set of comparative runs were performed on a suite of single zone cases to establish agreement between BRIS and the new models (in BRIS version). Disregarding the limited time resolution of BRIS, the results match perfectly. BRIS used fixed 0.5 h timesteps. IDA tolerances were selected to approximately match the same total number of timesteps, although in IDA steps are unevenly distributed over time, which gives significantly better time resolution. Differences in I/O make exact comparisons of execution time difficult, but similar cases show a ratio of 2 - 4, in BRIS favor. Robustness was good for both programs.

In conclusion, single zone cases with on the order of 50 temperature nodes show that for this type of problem, the performance penalty for the true MSE approach is acceptable. Typical IDA execution times for a 24 h period is 5 s on a Pentium PC. Larger multizone cases have yet to be compared. The performance of IDAs sparsity utilization algorithms on larger building models is naturally a crucial issue. Based on experience from other applications, we know that the problem size has to be significantly larger in order to adequately measure sparse performance.

The second question can only be answered qualitatively for the BRIS reimplementation, since more than thirty years have passed since the original work. The NMF models also have a wider scope. However, Axel Bring, who implemented the original BRIS, was also project leader for the NMF based effort. The following are his time estimates, assuming that both projects were carried out on modern platforms.

Table 6-1 summarizes the main steps that are involved in the development of a special purpose application like BRIS, in both traditional and MSE implementations. No GUI issues are regarded here, i.e., only IDA Solver is involved. The special purpose application is assumed to have a traditional input file structure. The corresponding IDA Solver input file will due to generality be slightly larger than the typical special purpose counterpart, but it will also be more legible and provide better error feedback. Let us for the sake of simplicity assume that they are of comparable overall quality.

A large part of the development effort lies naturally in the paper and pencil formulation of the mathematical model. This work is the same for both approaches, assuming that everything must be developed from scratch, and is not included below.

phase	special purpose time in weeks	IDA time in weeks	comment
design program architecture	4	2	NMF model architecture in the IDA case.
design and implement a suitable numerical method	8	0	Method is already available in the IDA case
implement math models as formal code	4	4	More information must be input in the NMF case, but the structure is given. In the special purpose case, the model code interacts with the solution procedure.
write I/O interfaces	12	0	I/O already available in IDA case
testing and tuning of full implementation	10	4	Less new code to be tested in IDA case. IDA tuning includes selection of suitable methods.
documentation of implemented models	4	4	Written account of model equations.
development of user's manual	4	1	IDA Solver file structure is already documented.
total time in man-weeks	**46**	**15**	

Table 6-1. Main phases in the development of a special purpose tool like BRIS

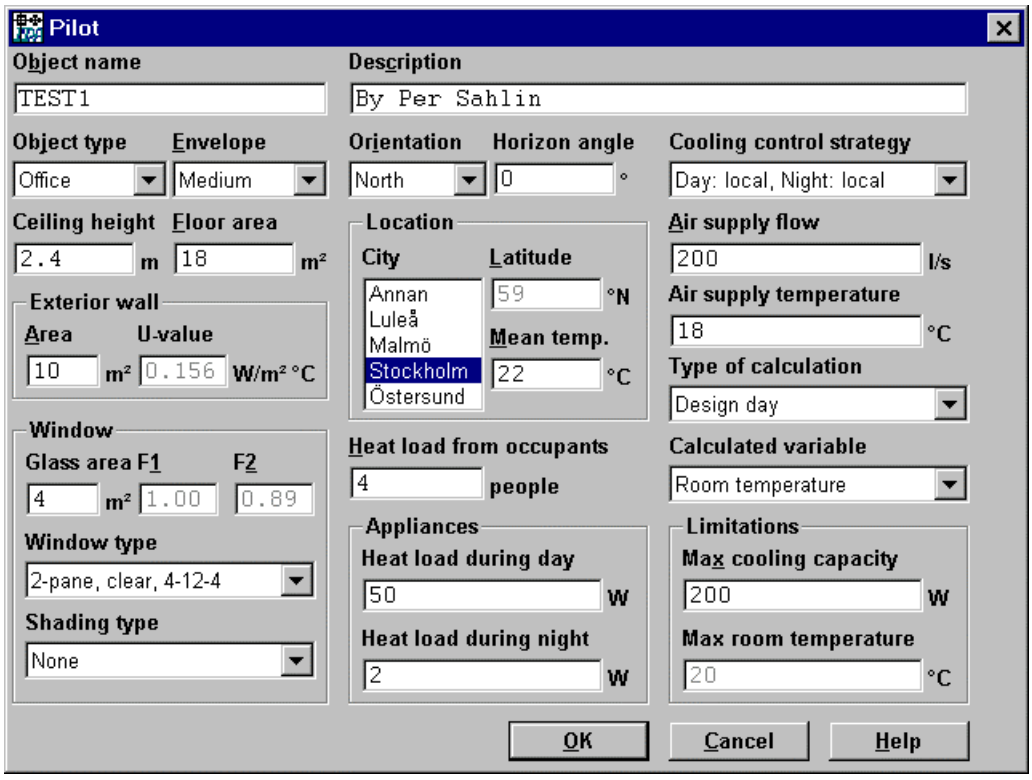


Figure 6-1. Main dialog box for Pilot application

So far, only a very simple GUI has been developed for the MSE implementation. The interface, temporarily named *Pilot*, is intended for cooling load calculations in the early stages of the building design process. It is based on default values for typical Swedish designs of offices, hospitals and school rooms. Only about twenty parameters are given by the user in a single dialog box (*Figure 1-1*). The development time for such an interface in IDA Modeller is comparable to that of any good GUI development platform. General IDA material is used only to organize case files and to display simulation results.

The developed application has recently been shipped to some thirty industrial test users, who have given positive feedback. The model accuracy, and thereby calculation times, are perceived to be somewhat overly ambitious for early stage use. However, the heat balance models are currently being equipped with a more comprehensive GUI that will enable users to utilize the full capability of the underlying models.

In conclusion, for the BRIS case, the special purpose application is about three times faster. Robustness is good for both implementations. Development time is approximately three times longer for the special purpose tool. If the better maintenance and interoperability properties of the MSE implementation is taken into account, the conclusion must be that the MSE approach is highly competitive.

6.1.1.2 Preserving Monolithic Tools

The third question is of a more speculative nature, and we have no first-hand experience of such projects. Several merger projects have been reported and still more are in the planning stages. Haves has investigated performance of the (explicit) HBT2 model in conjunction with HVACSIM+ [Haves 1989]. A merger of IBLAST and

HVACSIM+ has also been tried in a recent project [Metcalf 1995]. In the last few years, the inclusion of SPARK as a separate module of PowerDOE (DOE-2 with new GUI) for systems and plant simulation has been discussed . A plan to merge the two monolithic thermal tools BLAST and DOE-2 has also been proclaimed [Crawley 1995].

Merger projects have also been carried out in order to couple modular programs with special purpose monolithic tools for multizone airflow modelling. Coupling between TRNSYS and COMIS is described in [Dorer 1994].

Large implementation as well as user training efforts have been invested in existing monolithic tools. This is a natural motivation for the merger approach. Other reasons behind merger projects might be a need to quickly compose a program for a specific research task. In this case, the structure of the resulting program is less crucial, since it is not intended for continued development.

In our opinion, a cost effective creation of an attractive simulation environment, for long term satisfaction of developers as well as end users, through the merger approach is very unlikely. The reasons for this negative opinion are the following:

- No research has been presented that changes the long term development prospects of the monolithic tools in a positive direction. Instead, results from the IDA, CLIM 2000, and ALLAN.Simulation projects prove that a truly modular approach is indeed a possible alternative. This means that the work of re-implementing the monolithic tools in a modular architecture can only be postponed, not avoided.
- In order to create acceptable (not to mention desirable) structures for input data and program code, the monolithic tools have to be severely rearranged or encapsulated. In addition, documentation must be revised to reflect the new implementation. The volume of such work is very difficult to predict and could easily turn out to be of comparable order to a complete re-implementation project.
- User migration can be facilitated by import functions in a re-implemented tool. It should therefore not be regarded as a fundamental motivation for retaining the monolithic implementations.
- A dominant part of the value of an existing implementation lies in the knowledge about the implementation, not in the actual source code. This knowledge is tied to the (generally few) individuals that are truly familiar with the program. It is not obvious that this knowledge can be retained in merger projects, unless the original developers of both merged programs are directly involved in the new design.

Unless ignorance is the main reason for the instigation of non-research merger projects, doubts regarding the performance of alternative techniques must prevail. In this case, it seems rational to remove these doubts, before entering a worst-case alternative route - to live with the obviously wanting existing technology. Furthermore, recent results on so called modal reduction techniques, indicate that better performance can be attained in new implementations of building models than with any existing tools [Cools 1989]. By surrendering a modest amount of generality, such techniques can be utilized in re-implementation projects to compensate for the (small)

inherent performance penalty of the general approach. Such a project has been reported for ALLAN.Simulation [Lefebvre 1995].

The current interest in merger projects reveals a pessimistic attitude towards the future possibilities of building simulation.

6.1.2 Multizone Air Flow

IDA work on reimplementation of an existing multizone air flow program, Movecomp, is reported in Papers 4 and 5, where also some conclusions are drawn. Here, that discussion will be complemented with some additional material, and conclusions with respect to the objectives of this thesis.

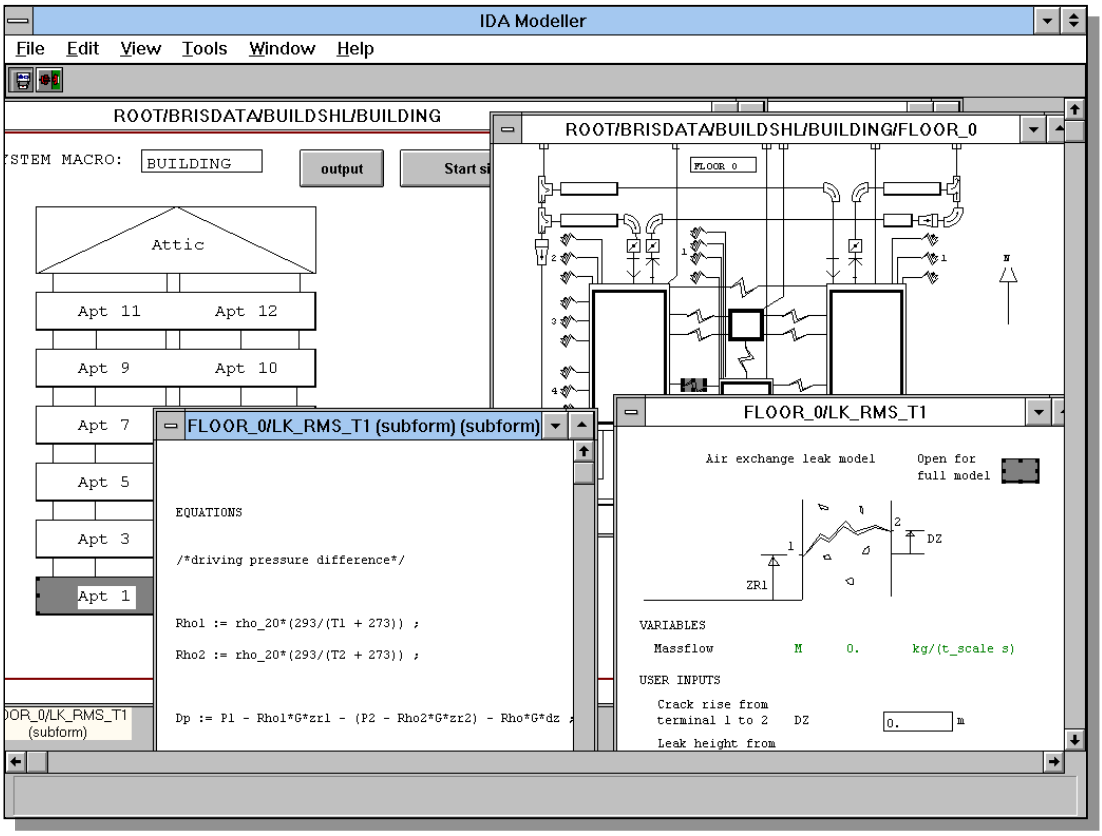


Figure 6-2. The GUI representation of a Multizone Air Exchange (MAE) model, Building, with six floors and an attic. Models are built and represented hierarchically with subsystems within subsystems. Overview as well as detailed presentations are available to accommodate users with varying levels of sophistication.

case	connections	nodes	MC matrix	IDA matrix	IDA cut set
Clean Room	19	6	6	75	-
Building	289	48	50	1050	353

Table 6-2. Characteristic dimensions for the two sample problems

Table 6-2 summarizes some key numbers for two sample cases, *Clean Room* and *Building*. The first case is a model of a sample clean room situation that is commonly referred to in the literature. No ventilation system is modelled. Experiments on variants of this model using IDA are reported in [Isfält 1996]. The *Building* model is due to Herrlin [Herrlin 1992]. It contains an elaborate structure of zones, leaks, and ventilation components, for a six floor building (cf. *Figure 6-2*).

In *Table 4-1*, the number of connections and nodes refer to the generic model types that are introduced in Section 2 of Paper 5. The following two columns indicate the sizes of the iteration matrix for Movecomp (MC) and IDA. The IDA matrix size refers to "the compact method" in IDA terminology. This method is briefly described in Section 5.3 of Paper 5 and more thoroughly accounted for in [Eriksson 1992]. The matrix size is, for this strategy, given by the total number of NMF `OUT` variables. The total number of `LOC` and `OUT` variables is a few times larger[19]. The IDA cut set column indicates the result of an alternate sparse IDA method "upper triangular" which attempts to minimize the size of the iteration matrix.

	preparation (man time)		execution (486 - 50)	
case	research	assembly	MC	IDA
Clean Room	day	hour	1.4 sec	4.3 sec
Building	month	day	5.7 sec	600 sec

Table 6-3. Characteristic times for the two sample problems

Some characteristic times for the two cases are presented in *Table 6-3*. For the larger case, the increase in problem size does indeed have a significant impact on execution time. However, about 80 percent of this execution time has been devoted to trying to find a minimal cut set. For a purely algebraic problem like this one, this is overly ambitious, since only a total of about 10 to 20 iterations are required, each of which takes about 6 seconds. A more optimal approach would be to terminate the cut set algorithm at an earlier stage, i.e., to settle for a less optimal cut set.

Table 6-3 also gives rough estimates of the man time required for model preparation. Research time refers to the gathering of physical data. This is today a purely manual process of scanning a large number of research reports with measurement data, and in a future scenario this could clearly be automated to a large degree. The assembly time is the actual time for the interactive process of describing an IDA model.

The execution speed for multizone air flow problems is rarely a limiting factor in practical work. With Movecomp, it is hardly noticed. However, for large cases, the general approach is obviously wanting in this respect. Some methods to improve this situation are discussed in Paper 5. A new variable partitioning scheme is currently being implemented, but no results are available yet. A conservative estimate is that IDA will remain on the order of ten times slower for large cases.

[19] For equation based solvers, where NMF assignments are mapped to equations, the total problem size for this case would be on the order of four thousand equations.

The robustness of the MAE models is good, and no initial guesses are required from the user. A few problems remain with the current implementation of T-pieces. The selected level of approximation includes functions with several inflection points, and this occasionally gives convergence problems for odd cases. A less ambitious level of approximation for these models is likely to resolve this problem.

Regarding the development time, we will separate GUI from NMF issues. Axel Bring implemented the Movecomp program, in the mid eighties. The program takes its input from a well structured input file, but no GUI is available. According to Bring, the relative time estimates from the BRIS case, in *Table 6-1*, apply qualitatively here as well. The calculation engine of the special purpose application takes roughly three times longer to develop than a MSE based alternative.

No template application was available for the GUI. The current implementation is used to some extent in Swedish industry, but remains rather rough, lacking, e.g., thorough on-line help. The implementation draws heavily on the available methods of IDA Modeller. A rough estimate of the total GUI development time is 8-12 man-weeks. No direct special purpose comparison is available, but it can safely be assumed to be several times longer.

In conclusion, a GUI based tool for multizone air flow has been implemented. It performs satisfactorily for normal industrial use. Some problems regarding execution times for large cases remain. These problems have practical consequence only for cases when repeated calculations are required. The development time is significantly shorter than for a specialized approach.

6.1.3 Coupled Thermal and Fluid Flow Problems

The most significant advantage of the MSE based methodology is that models may be mixed and matched arbitrarily. Thus, coupled problems require no special measures. No direct comparison with existing tools have been performed for coupled problems. The only well-known special purpose tool for building simulation that offers coupling between thermal and flow problems is ESP-r [Clarke 1995]. In the following, we will briefly summarize a selection of IDA application projects that have been carried out in this category.

6.1.3.1 Fire Studies at SINTEF

Models for fire induced heat propagation between zones and in the ventilation duct-work of offshore platforms have been developed at SINTEF in Norway [Kolsaker 1991]. Both buoyancy driven air flow and thermal storage in walls are modelled.

Studies have been performed for fire scenarios in both sleeping quarters and in machine rooms far below sea level. The first few minutes of a fire are frequently critical in terms of injuries and loss of lives due to smoke inhalation. Results show among other things that fire dampers far away from potential fire locations, which is common in current practice, are rather useless from this perspective. The temperatures at the dampers do not reach the closing setpoint during the first few minutes of a fire.

The models were developed with close contact between SINTEF and the client. NMF was used as the sole mode of model documentation in the communication process.

6.1.3.2 Ventilation and Fire Studies in Traffic Tunnels

In tunnels designed for longitudinal ventilation, the traffic itself drives the ventilation air under normal conditions. In accident or fire scenarios, fans play an important role. Modelling of fires calls for coupled models of both air movement and heat transfer. Client specific tools for this application have been developed by Bris Data AB in Sweden [Malmström 1995].

The core model is a straight tunnel segment with a longitudinal air temperature profile. Special difficulties in this model arise when the flow reverses direction, in the fire segment, due to developing buoyancy forces. Additional NMF components enable modelling of complex tunnel networks, with multiple loops, and their mechanical ventilation system. Aside from the basic heat and massflow equations, relations are also included for vehicle contaminant production and transport.

Only IDA Solver based applications have been developed so far, i.e., no GUI. However, non-expert clients use these to perform studies with several thousand equations. Development times for these applications have been on the order of 6 man-weeks, including documentation and testing, given the existence of a complete mathematical model.

6.1.3.3 Natural Ventilation Studies

Active use of window ventilation was considered as an alternative to installing a new mechanical ventilation system in a retrofit of a Swedish hospital. Traditional thermal tools are unable to predict the effect of open windows, and therefore, the IDA models for zone heat balances (cf. Section 6.1.1.1) were combined with multizone air flow models, to study this case [Bring 1994]. The complex geometry of the window openings were divided into small sections, in which powerlaw flow models were applied.

The coupled models gave quick estimates of room temperatures in various ventilation scenarios. The results were promising, and a decision was taken to move ahead with the proposed low cost refurbishment.

6.1.4 Demand Controlled Ventilation

TRNSYS models for a demand controlled ventilation case have been developed and exercised by Emmerich and coworkers [Emmerich 1994]. David Lorenzetti of MIT combined a subset of the MAE models (cf. Papers 4 and 5) with some new controller models to (loosely) replicate Emmerich's models. The resulting model family is presented together with some simulation results as a system example in Appendix 4 of the NMF reference report [Sahlin 1996b].

The implementation was carried out during a three day visit at KTH, primarily to study the model development process of IDA. The resulting implementation has not yet been compared with the TRNSYS counterpart.

6.2 Remaining MSE Problems

In this section, we will discuss the main obstacles to a more wide-spread application of MSE based technology. Some of these are of a practical nature, others will require more research.

6.2.1 Affordable Quality Implementations

The most tangible problem is the shortage of affordable quality implementations. Some of the available model-lab products are extremely expensive. IDA will soon be released in a developer's version, for generation of end-user tools. None of the alternative products have such a version yet.

6.2.2 User Awareness and Training

Another obvious bottleneck regards the industrial awareness of the technology. Currently, only a small fraction of potential users are aware of the merits of general DAE modelling, and of the available tools in this field.

6.2.3 Efficiency on Large-Scale Problems

The computational efficiency is still inadequate on some problem types. As we have seen in the previous section, the general problem formulation can in some cases lead to radically larger problem sizes. An example of a common problem type that is likely to be impractical to tackle with the current methods is duct or pipe sizing. Such a program must be able to produce numerous solutions of pressure and flow for a large network as the algorithm searches for an optimal system size.

No tests of the performance of general purpose sparsity utilization algorithms have been performed within the IDA project on these problem types. However, it is unlikely that they will perform adequately.

Although more work certainly is needed on general purpose sparsity utilization, we feel that only a modest improvement is possible this way on single processor machines. There will always exist special problem structures that remain undetected by the general purpose methods. A remedy to this problem would be to give explicit problem dependent advice on a suitable computational procedure, i.e., to depart from the general purpose approach in order to gain acceptable efficiency for special problems.

This may seem completely counterproductive, since the whole point is to apply general purpose methods to special problems. However, adding extra information to a model to facilitate efficient solution in a certain setting will not compromise its usefulness in a general scenario. The extra information will then simply be ignored.

We believe that utilization of special problem structures will be an important area of future MSE research. Some such work is already underway in the IDA project.

7. Summary and Conclusions

7.1 The IDA/NMF Design Profile

The key points of the IDA and NMF designs have been motivated. They are summarized in the table below.

capability	implemented	sample problems or other motivation
Differential-algebraic equations	yes	heat exchange in zone; multizone air flow; flow in pipe or duct networks
Input-output free models	yes	any potential-flow network, such as RC or multizone air flow
Advanced algebraic solution techniques	yes	multizone air flow; refrigeration loops
Variable timesteps	yes	acceptable efficiency for many transient problems, such as a heat balance zone model with local cooling control
Range of sparsity utilization algorithms	yes	advanced algorithms necessary for, e.g., large multizone models; simple or no sparsity utilization option necessary for maximal robustness during model development
Hysteresis	yes	thermostat or imperfect valve
Time and state events	yes	thermostat controlled system in variable timestep environment
Delays	no	plug flow models for, e.g., pipes
Low-resolution PDE modelling	yes	1D heat equation; tube immersed in slab or semi-infinite space
High-resolution PDE modelling	no	CFD calculation of internal or external flows
Discrete time models	no	sampling controllers; weather data processing
Hierarchical modelling	yes	Structured large-scale models, e.g., a building with a large number of similar floors, zones, walls etc.

capability	imple-mented	sample problems or other motivation
Port level connections	yes	models can be interconnected without detailed knowledge of underlying models, e.g., pump outlet is connected with pipe endpoint
Models with incrementable number of ports	yes	interactive building of, e.g., RC network, thermal zone model, or multizone air flow model
Fast development of end user programs	yes	enables development of end user applications for limited market
Distribution of low-cost (no compiler) applications with ability to graphically interconnect fixed sub-models	yes	multizone air flow; multizone thermal program; duct or pipe sizing tool
Access to external model libraries	yes	necessary when project budget prevents development from scratch
Access to CAD data from end user programs	no	to prevent trivial re-entry of data, e.g., building geometry
Access to foreign subroutine based models with, private memory handling, events, and explicit linearization	yes	fast access to existing subroutine based model libraries; ability to distribute models in binary form for commercial purposes

Table 7-1. The IDA/NMF design profile

With exceptions as indicated in *Table 7-1*, the design profile has been implemented and tested. Application projects have been summarized in all of the selected primary target applications: *Building Loads and Energy Calculation, Multizone Air Flow, Coupled Thermal and Fluid Flow Problems, and Demand Controlled Ventilation.*

7.2 Conclusions

7.2.1 Problem Review

The specific building simulation problem areas that were pointed out in Section 1.4 are repeated below, with corresponding conclusions.

1. *The degree of model reuse in the field is low. Available mathematical models are generally packaged in a form that is unsuitable for direct reuse in diverse settings. Researchers produce validated and documented mathematical mod-*

els, but these are then presented in a form which is either too general, i.e., written equations in a report, or too specific, i.e., intertwined with a solution algorithm in a special implementation. The same problem applies to model reuse between different projects that are carried out within the same group, but with different simulation tools.

NMF provides a concrete alternative model formulation method that has been proven to work on real-scale projects by independent users. Although further development is going on, the present material is sufficient for the establishment of model servers for both internal and external model communication.

2. *With available techniques, the development cost of special purpose application tools is too high. This in turn prevents exploitation of many potential simulation problems, since the market for each individual special purpose tool is small. The collective effect is that simulation is poorly utilized as a general method.*

   IDA applications have been developed within a fraction of the normal cost. This has enabled application development in narrow areas such as ventilation of road tunnels.

3. *Special purpose tools that are developed with available techniques are inadequate for the end user in the following respects:*

   a) *They offer no practical way for a user to adapt the tool, in an unplanned way, to suit the problem at hand, i.e., the ability to reprogram is exclusive to the developers.*

      Developed IDA applications are completely transparent. Small changes to existing models are easy to carry out by users with access to a development version of the IDA system and with sufficient understanding of the mathematical models. Such alterations have been performed on, e.g., the Multizone Air Exchange application to study demand controlled ventilation.

   b) *The mathematical models are documented separately, creating a double source problem. The user is often in doubt regarding the correspondence between the documentation and the implemented model.*

      IDA applications are normally delivered with the NMF source code. This creates a way for users to investigate the equations that are being solved. Additional documentation presents mathematical models also in traditional form to provide overview and discussion. Clear references are made in such text to the relevant NMF code.

   c) *They are generally too inhomogeneous in terms of user interaction principles to allow a user quick transitions between different tools. The required common principles shared by, e.g., two good Windows applications, are too superficial to make it possible to have a large number of different applications simultaneously active (mentally).*

      IDA Modeller provides a common framework for IDA applications.

Unfortunately, to date, only a few applications have been developed that utilize this framework, and it is too early to estimate the impact on end user productivity.

d) *Only the subset of model quantities that have been selected by the developers is available for study. If for example wall temperatures are interesting, they may not be possible to present.*

Most implementations of the MSE approach, including IDA, enable users to monitor every system variable.

4. *Available MSEs are inadequate, both regarding the encompassed problem types, and the possibility to distribute attractive special purpose applications.*

Only IDA has been demonstrated to solve, e.g., multizone air flow and heat balance problems with sufficient robustness. Although execution speed remains an area of improvement, it is currently sufficient for industrial use in many areas.

Distribution of MSE based special purpose applications has been demonstrated in the scope of the IDA project. The only available alternative tool that has been used in this way is TRNSYS. The modelling tools for TRNSYS are not intended for delivery of special purpose tools; the solver is wanting with respect to variable timestep integration and algebraic solution techniques.

7.2.2 Overall Conclusion

Several projects using TRNSYS, HVACSIM+, ALLAN.Simulation, and CLIM 2000 have shown that modular simulation environments are useful for a variety of building simulation problems. Other MSEs, such as Dymola, have been applied to demanding special problems, such as simulation of multibody systems, that previously have been restricted to special purpose tools.

The present work on IDA and NMF serves to increase the range of applicability in some key aspects,

- production of quality end user simulation tools at low cost,
- application to important typical building simulation problems, such as zone heat balances and multizone air flow,
- establishment of base technology for large, program independent, model libraries.

With the exception of a few special situations, all continuous simulation problems of the building sector can be handled by general purpose methods as offered by modular simulation environments. The use of this technique is cost effective already in the development of special purpose applications. The cost of long term maintenance of developed tools can also be expected to be significantly lower.

7.3 Further Work

The rate of development of IDA and NMF is escalating rapidly. On-going work has been briefly indicated throughout the text. The main areas of activity around the original IDA group are briefly summarized below.

7.3.1 IDA Application Development

A group of companies[20] is currently co-funding the development of a range of building simulation applications. The Pilot application, mentioned in Section 6.1.1.1, is the first product of this work. Three larger additional applications are planned for calculation of energy and loads, simulation of typical control problems, and for system selection support. All applications are developed in both Swedish and English. The total effort is approximately ten man-years and it is scheduled to be completed at the end of 1997.

7.3.2 IDA Modeller

IDA Modeller was originally developed on Apollo workstations using non-standard user interactions principles. The original version was ported to Windows in 1992. The implementation is currently being revised

- to harmonize better with the Windows environment,
- to include features that were not originally foreseen,
- to better utilize the Common Lisp Object System, and
- to provide better quality graphics.

7.3.3 IDA Solver

Several projects are underway to further improve IDA Solver.

- Discrete time objects are under implementation.
- A new problem specific variable partitioning scheme is under development.
- Methods for re-scaling of variables and equations are being investigated.
- Better support for application encapsulation in Windows Dynamic Link Libraries is under implementation.

7.3.4 IDA NMF Translators

A new family of IDA NMF translators have been developed during the last year. These translators have computer algebra capability, and this is currently applied to automatic generation of analytical Jacobian routines. The new translators will also be integrated into IDA Modeller.

7.3.5 General NMF Research

The suggested future work of Papers 6 and 7 is underway. This will provide a better foundation for the establishment of NMF model servers.

[20] A consortium of 28 companies that are listed on the inside of the front cover.

Acknowledgments

Axel Bring has been a friend, partner, and mentor for the last nine years. Without you, I would never have endured. With you, it was worth it.

While Axel was my accomplice, my dear wife Gun Rudquist has been an innocent victim. She is no fan of IDA. She even fails to see the beauty of NMF! In fact, she doesn't much enjoy differential-algebraic modelling of piecewise continuous systems at all. No, biological diversity, sustainable agriculture, and such is what she wastes her talents and time on. In spite of all this, she and our daughters, Hanna and Kajsa, have supported me - along with my strange priorities.

The present and past IDA team have on countless occasions put in that extra bit of effort, especially to let me enjoy this selfish exercise. The old gang that Axel and I have enjoyed to work with are Lars Eriksson and Magnus Lindgren. Newcomers are Pavel Grozman, Harald Hermansson, Wille Nordqvist, Alexander Shapovalov, and Mika Vuolle.

The list of people without whom IDA never would have come true is long. Their continued support through long stretches of poor results, and jealous plots against us, has made all the difference. First on this list is Tor-Göran Malmström, who many times has endangered his reputation for our sake. Former and present heads of ITM Gustaf Söderlind and Uno Nävert are also on the list. Engelbrekt Isfält has unselfishly pointed many clients in our direction. Ingemar Nordenadler, president of the IDA consortium, has similarly acted on our behalf. Several other partners in the IDA consortium have also joined primarily to support rather than to gain.

The early IDA users, who have put up with using early and incomplete versions of the code, also deserve our gratitude. First among them is our true pioneer, Kjell Kolsaker, who has worked wonders with IDA. Secondly, Ingegerd Ljungkrona, who also has shown great confidence in our work from the very early days. The same is true for Jan Akander, Dave Lorenzetti, and Niklas Björsell.

Kjell Kolsaker, Ed Sowell, Francis Lorenz, Jean-Michel Nataf, Dave Lorenzetti, Les Norford, Roger Pelletret, Alexandre Jeandel, Chip Barnaby, Phil Haves, and many others have been valued scientific discussion partners and NMF supporters.

Axel Bring, Wille Nordqvist, Bo-Christer Björk, Per Isaksson, Gudni Johanneson, and all others who have given valuable comments on the manuscript are gratefully acknowledged.

Finally, I want to thank the project funders (listed on the inside of the front cover), particularly the Swedish Council for Building Research that has provided continuous support throughout the project. I have personally also been honored by the Royal Institute of Technology in the form of a so called "excellenstjänst," and by a stipend from Stockholms Byggnadsförening.

References

Akander, J., 1995, *Efficient Modelling of Energy Flow in Building Components,* Div. of Building Technology, KTH, Stockholm, Sweden

Amor, R., J. Hosking, 1995, *Mappings for Integrating Design Tools,* Conference proc. Building Simulation '95, IBPSA, Madison, WI, USA, Aug. 1995

Andersson, M., 1990, *An Object-Oriented Language for Model Representation*, Licentiate Thesis, Dept. of Automatic Control, LIT , Box 118, 221 00 Lund, Sweden, May 1990

Andersson, M., 1994, Object-Oriented Modeling and Simulation of Hybrid Systems, PhD thesis, Dept. of Automatic Control, LIT , Box 118, 221 00 Lund, Sweden, Dec. 1994

ASEA, 1983, *SANDYS - Användarhandledning,* ASEA Research & Innovation, S12013-AH01, 1983

Augenbroe, G., 1993, *COMBINE Final Report,* EU-DG XII JOULE Report, 1993

Augenbroe, G., 1995, *COMBINE 2 Final Report*, EU-DG XII JOULE Report, 1995

Augustin, C.D.C., M.S. Fineberg, B.B. Johnsen, R.N. Linebarger, F.J. Sannson, J.C. Strauss, 1967, *The SCi Continuous System Simulation Language (CSSL),* Simulation, 9, pp. 281-303

Björk, B.-C., 1995, *Requirements and information structures for building product data models,* PhD thesis, VTT Publications 245, Technical Research Centre of Finland, ISSN 1235-0621

Bonin, J.L., J.Y. Grandpeix, J.L. Joly, A. Lahellec, V. Platel, M. Rigal, 1989, *Multimodel Simulation: the TEF Approach,* Proc. European Simulation Conference, Rome

Bonneau, D., F.X. Rongere, D. Covalet, B. Gautier, 1993, *CLIM 2000 - Modular Software for Energy Simulation in Buildings,* Conference proc. Building Simulation '93, IBPSA, Adelaide, Australia, Aug. 1993

Brenan, K.E., S.L. Campbell, L.R. Petzold, 1989, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations,* North Holland

Bring, A., 1991a, *IDA SOLVER, a Programmers's Guide,* Research Report, Dept. of Building Services Engineering, Royal Institute of Technology, Stockholm

Bring, A., 1991b, *IDA SOLVER, a User's Guide,* Research Report, Dept. of Building Services Engineering, Royal Institute of Technology, Stockholm

Bring, A., I. Nordenadler, E. Isfält, 1994, *Simuleringsprogrammet IDA öppnar fönstret för ny teknik,* Energi & Miljö, 5-6/94

Bring, A., P. Sahlin, 1993, *Modelling Air Flows and Buildings with NMF and IDA*, Conference proc. Building Simulation ‘93, IBPSA, Adelaide, Australia, Aug. 1993

Bring, A., L. Eriksson, H. Hermansson, M. Lindgren, P. Sahlin, 1995, *IDA - an Environment for Building and Energy Systems Simulation*, Conference proc. Building Simulation ‘95, IBPSA, Madison, WI, USA, Aug. 1995

Brown G., 1962, *BRIS - Method for digital computer calculation of long and shortwave radiation in rooms as well as cooling and heating needs*, KTH, Stockholm, 1963

Brown, G., 1990, *The BRIS Simulation Program for Thermal Design of Buildings and Their Services*, Energy and Buildings, 14 (1990) 385-400

Brück, D.M., 1987, *Implementation Languages for CACE Software*, Dept. Automatic Control, LTH, Lund, Sweden, Sept., 1987

Buhl, W.F., E.F. Sowell, J.M. Nataf, 1989, *Object Oriented Programming, Equation Based Submodels, and System Reduction in SPANK*, Building Simulation '89 Conference, Vancouver, Canada, June, 1989

Buhl, W.F., E. Erdem, F.C. Winkelmann, E.F. Sowell, 1993, *Recent Improvements in SPARK: Strong Component Decomposition, Multivalued Objects, and Graphical Interface*, Conference proc. Building Simulation ‘93, IBPSA, Adelaide, Australia, Aug. 1993

Charlesworth, P., et al., 1991, The Energy Kernel System, Proceedings, Building Simulation '91, Nice, France, Aug. 1991

Clark, D.R., 1985, *HVACSIM+ Building Systems and Equipment Simulation Program - Reference Manual*, National Institute of Standards and Technology, Washington D.C., 1985

Clarke, J.A., 1986, *The Energy Kernel System*, Conference proc. System Simulation in Buildings, Liege, Belgium, Dec., 1986

Clarke, J.A., W.M. Dempster, C. Negrão, 1995, *The implementation of a computational fluid dynamics algorithm within the ESP-r system*, Conference proc. Building Simulation '95, IBPSA, Madison, WI, USA, Aug. 1995

Clarke, J.A., E.F. Sowell, the Simulation Research Group, 1985, *A Proposal to Develop a Kernel System for the Next Generation of Building Energy Simulation Software*, Lawrence Berkeley Laboratory, Berkeley, CA, Nov. 4, 1985

Clarke, J.A., D.F. Mac Randall, 1993, *The Energy Kernel System: Form and Content*, Conference proc. Building Simulation ‘93, IBPSA, Adelaide, Australia, Aug. 1993

Cools, C., R. Gicquel, F.P. Neirac, 1989, *Identification of Building Reduced Models. Application to the Characterization of Passive Solar Components*, Int. J. Solar Energy, 1989, Vol. 7, pp. 127 - 158

Crawley, D.B., L. Lawrie, 1995, e-mail newsletter, US Depts. of Energy and Defence, Sept., 1995

Dorer, V., Weber A., 1994, *Multizone Air Flow Model COMVEN as Type 57 for TRNSYS,* IEA-ECB Annex 23 'Multizone Air Flow Modelling' Technical Note, June, 1994

Dubois, A.M., 1988, *MODEL-BASED COMPUTER AIDED MODELLING: the new perspectives for building energy simulation*, communication from CSTB, B.P. 21, 06561 VALBONNE Cedex, France

Elmquist, H., 1978, *A Structured Model Language for Large Continuous Systems*, Phd thesis, Dept. of Automatic Control, LIT , Box 118, 221 00 Lund, Sweden

Elmqvist, H., **1986,** *LICS: Language for Implementation of Control Systems,* Dept. of Automatic Control, LIT , Box 118, 221 00 Lund, Sweden

Elmqvist, H., 1993, *Object-Oriented Modeling and Automatic Formula Manipulation in Dymola*, proc. SIMS '93, Scandinavian Simulation Society, Kongsberg, Norway, June, 1993

Emmerich, S.J., J.W. Mitchell, W.A. Beckman, 1994, *Demand-Controlled Ventilation in a Multi-Zone Office Building*, Indoor Environ 1994:331-340

Eriksson, L.O., 1983, *MOLCOL - An Implementation of One-leg Methods for Partitioned Stiff ODEs,* Report TRITA-NA-8319, Royal Institute of Technology, Stockholm

Eriksson, L.O., 1991, Personal communication

Eriksson, L.O., G. Söderlind, A. Bring, 1992, *Numerical Methods for the Simulation of Modular Dynamical Systems,* Research Report, Swedish Institute of Applied Mathematics, Gothenburg

Escudié, J.C., L. Laret, 1994, *CSTBât: A Simulation Tool for Building and Heating System Analysis within TRNSYS,* Proc. 4th intl. conf. System Simulation in Buildings, Liege, Belgium, Dec., 1994

Feustel, H.E., A. Rayner-Hooson, 1990, *COMIS Fundamentals,* Lawrence Berkeley Laboratory, CA,

Grozman, P., P. Sahlin, 1996, *ASHRAE RP-839 NMF Translator - User's Guide*, ASHRAE Inc. and Bris Data AB

Haves, P., 1989, personal communication

Haves, P., L. Norford, 1996, *Development of a Simulation Testbed to Evaluate Control Algorithms and Strategies for Variable-Air-Volume Ventilation Systems*, Research report ASHRAE 825-RP, draft version, MIT, 1996

Herrlin, M.K., 1992, *Air-Flow Studies in Multizone Buildings - Models and Applications,* PhD thesis, Bulletin no. 23, Building Services Engineering, KTH, Stockholm, Sweden

ISO TC 184, 1993, *The STEP Standard,* draft international standard DIS 10303, continuously since 1992 published in several different parts

IEA B&CS Annex 1, 1981, *Comparison of Load Determination Methodologies for Building Energy Analysis Programs,* U.S. Department of Energy, Jan., 1981

Isfält, E., B. Ljungqvist, B. Reinmüller, *Simulation of Airflows and Dispersion of Contaminants through Doorways in a Suite of Cleanrooms,* submitted to European Journal of Parenteral Sciences

Jeandel, A., F. Favret, E. Lariviere, 1993, *ALLAN.Simulation - a General Software Tool for Model Description and Simulation,* Conference proc. Building Simulation '93, IBPSA, Adelaide, Australia, Aug. 1993

Jeandel A., Ph. Ravier, A. Buhsing, 1994a, *ULM : reference guide,* GDF internal report M.DéGIMA.GSA1205 , 1994

Jeandel A., Ph. Ravier, A. Buhsing, 1994b, *ULM : user´s guide,* GDF internal report M.DéGIMA.GSA1206, 1994

Jochum, P., 1994, *Einzatz der Simulationsumgebung Smile zur Simulation solar unterstützter Heizsysteme,* Institut Für Heizsysteme, TU-Berlin, 1994

Jochum, P., M. Kloas, 1993, *The Dynamic Simulation Environment Smile*, Institute of Energy Engineering, TU-Berlin, 1993

Judkoff, R., J. Neymark, 1994, *Building Energy Simulation Test (BESTEST) and Diagnostic Method,* International Energy Agency through NREL, NREL/TP-472-6231

Klein, S. A., W. A. Beckman, J. A. Duffie, 1976, *TRNSYS - a transient simulation program* ASHRAE Trans, 1976, VOLUME 82, Pt. 2

Kolsaker, K., 1991, *An NMF-Based Component Library for Fire Simulation,* Proceedings, Building Simulation '91, Nice, France, Aug., 1991

Kolsaker, K., 1994a, *NEUTRAN-supported NMF Enhancements,* presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Kolsaker, K., 1994b, *Simpler NMF Description of Advanced Models Using Hierarchical Modelling and Data Abstraction,* presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994

Kolsaker, K., 1994c, *NEUTRAN - A Translator of Models from NMF into IDA and SPARK,* Proceeding of the BEPAC conference, BEP'94, York, U.K.

Lefebvre, G., J.-M. Nataf, A. Oulefki, A. Jeandel, E. Givois, P. Briand, O. Noel, 1995, *Generating Reduced Modal Models of Buildings for ALLAN.Simulation+*, Conference proc. Building Simulation '95, IBPSA, Madison, WI, USA, Aug. 1995

Lindgren, M., P. Sahlin, *1992, IDA Modeller - A Users's and Programmer's Guide*, Research Report, Swedish Institute of Applied Mathematics, Gothenburg

Ljungkrona, I., 1994, *Thermal Room Model for Dynamic Performance Analysis of Conditioned Rooms - Description and Validation*, Licentiate thesis, Dept. Building Services Engineering, Chalmers, Gothenburg, Sweden, 1994

Lomas, K.J., H. Eppel, C. Martin, D. Bloomfield, 1994, *Empirical validation of thermal building simulation programs using test room data*, Internation Energy Agency B&CS Annex 21, 1994

Lorenz, F., 1987, *Reflections about Representation Methods*, proc. workshop on the future of building energy modelling, Ispra, Italy, Nov. 1987, CEC EUR 11603 EN PREPRINT, May 1988

Lorenz, F., 1990, *Brief Description of the MS1 (Modelling System 1) Project*, private communication

Lorenz F., 1991, *Modelling Platform with Multiple Representation Formalisms*, Proc. of BS'91 IBPSA Conf., Nice, France, Aug., 1991

Lorenz, F., 1994a, personal communication

Lorenz, F., 1994b, *Comments on the Neutral model Format*, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Lorenz, F., 1994, *A multiformalism modelling approach using Bond Graphs, Networks and Block Diagrams together*, keynote lecture, Eurotherm Seminar 36, Poitier Futuroscope, Sept. 21-23, 1994

Malmström, T.-G., A. Bring, 1995, *A Modular Simulation Program for Road Tunnel Ventilation*, KTH, 1995, submitted to Tunneling and Underground Space Ventilation

Mattsson, S.E., 1986, *On Differential/Algebraic Systems*, Research Report CODEN: LUTFD2/(TFRT-7327)/1-026, Dept. of Automatic Control, Lund Institute of Technology, Sept., 1986

Mattsson, S.E., 1989, *On Modelling and Differential/Algebraic Systems*, Simulation, 1989, 52, No. 1, 24-32

Mattsson, S.E., M. Andersson, 1992, *The Ideas behind Omola*, proc. 1992 IEEE Symposium on Computer-Aided Control System Design, pp. 23-29

Metcalf, R.R., R.D. Taylor, C.O. Pedersen, R.J. Liesen, D.E. Fisher, 1995, *Incorporating a Modular System Simulation Program into a Large Energy Analysis*

Program: the Linking of IBLAST and HVACSIM+, Conference proc. Building Simulation '95, IBPSA, Madison, WI, USA, Aug. 1995

Nakhle M., P. Roux, 1986, *NEPTUNIX : an efficient tool for large size systems simulation,* Second International Conference on System Simulation in Buildings 1986, Liege, Belgium

Nataf, J.-M., 1995, *Translator from NMF to SPARK*, Conference proc. Building Simulation '95, IBPSA, Madison, WI, USA, Aug. 1995

Ohlsson, B., A. Persson, 1991, *Sandys - a simulation program for electrical circuits and systems*, ABB Review 5/91

Otter, M., 1992, *DSblock, a neutral description of dynamic systems*, Open CACSD Electronic Newsletter, 1, 3 (available at ftp://mailbase.ac.uk/pub/lists-a-e/engineering-cace)

Otter, M., H. Elmqvist, F.E. Cellier, 1993, *Modeling of Multibody Systems with the Object-Oriented Modeling Language Omola*, proc. NATO/ASI, Computer-Aided Analysis of Rigid and Flexible Mechanical Systems, Troia, Portugal, June, 1993

Park, C., D. Clarke, G. E. Kelly, 1985, *An overview of HVACSIM+, a dynamic building/HVAC/control systems simulation program*, Proceedings 1st. Annual Building Energy Simulation Conference, Seattle, WA, 1985

Pelletret, R., 1994a, Personal communication

Pelletret, R., S. Soubra, 1994b, *Standardizing Model Documentation - The PROFORMA Experience*, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Pelletret, R., S. Soubra, W. Kielholz, 1995, *Transferring Simulation Techniques to End Users - Application to TRNSYS*, Conference proc. Building Simulation '95, IBPSA, Madison, WI, USA, Aug. 1995

Perkins, J.D., R. Sargent, 1982, *SPEEDUP: A Computer Program for Steady-State and Dynamic Simulation and Design for Chemical Processes*, AIChE Symposium Series, 78:214, pp. 1-11

Perry, D.L., 1991, *VHDL*, McGraw-Hill

Petzold, L.R., 1982, *A Description of DASSL: A Differential/Algebraic System Solver*, Proceedings of IMACS World Congress, Montreal, Canada, 1982

Rongere, F.-X., W. Ranval, 1992a, *A Modelling Method for Systems in Building Energy Simulation: MEMPHIS*, Electricite de France, April 1992, HE 12 W 3340

Rongere, F.-X., 1992b, personal communication

Sahlin, P., 1991, *IDA - a Modeling and Simulation Environment for Building Applications*, ITM report 1991:2, Dec., 1991

Sahlin, P., 1994, *The Neutral Model Format - a Possible Starting Point for a Standardization Process*, Progress report 1994 of the CEC SiE-WG 8467, Brussels, Belgium

Sahlin, P., 1996a, *NMF Handbook - an Introduction to the Neutral Model Format,* Research Report, Dept. Building Sciences, KTH, Stockholm, Sweden, Feb., 1996 (available at ftp://urd.ce.kth.se/pub/reports/handbook.ps)

Sahlin, P., A. Bring, 1993, *Applying IDA to Airflow Problems in Buildings*, ITM report 1993:3

Sahlin, P., A. Bring, 1995, *The IDA Multizone Air Exchange Application*, Bris Data AB and Div. of Building Services Engineering, KTH, 1995

Sahlin, P., A.Bring, E.F.Sowell, 1996b, *The Neutral Model Format for Building Simulation*, Version 3.02, Report, Dept. of Building Sciences, KTH, Stockholm, 1996 (available at ftp://urd.ce.kth.se/pub/reports/nmfre302.ps)

Shapovalov, A., 1996, personal communication

Sowell, E.F., K. Taghavi, H. Levy, D.W. Low, 1984, *Generation of Building Energy System Models,* ASHRAE Transactions, Vol. 90. Part 1, 1984

Sowell, E.F., W.F. Buhl, A. E. Erdem, F.C. Winkelmann, 1986, *A Prototype Object-based System for HVAC Simulation*, Proceedings of the Second International Conference on System Simulation in Buildings, Liege, Dec., 1986

Stangerup, P., 1988, *Efficient Implementation of a Description Language for Thermal Simulators*, Proc. 3rd European Symposium of Space Thermal Control & Life Support Systems, Noordwijk, The Netherlands, Oct., 1998

Stangerup, P., 1991a, *Requirements for a General Purpose Modelling and Simulation Language*, proc. 1991 European Conf. on Circuit Theory and Design, Copenhagen, Denmark, Sept, 1991

Stangerup, P., 1991b, *ESACAP: A Simulation Program and Description Language for Interdisciplinary Problems,* proc, 1991 European Simulation Multiconference, Copenhagen, Denmark, June, 1991

Söderlind, G., J. Oppelstrup, 1984, *The Modern Process Simulation Environment: Modelling and Numerical Methods,* ITM internal report

Söderlind, G. 1985, *Numerical Computation and Data Communication in Modular Simulation,* ITM internal report

Söderlind, G., L.O. Eriksson, 1985, *Modular Simulation: Theoretical Aspects and Numerical Experiments,* ITM internal report

Söderlind, G., L.O. Eriksson, 1986, *On the Design of a Modular Simulation System*, ITM internal report

Vuolle, M., 1996, personal communication

Walton, G., 1994, *CONTAM93 - User Manual*, NISTIR 5385, National Institute of Standards and Technology

Wernstedt, G., 1995, *Användarkrav för datorberäkningsprogram för byggnader*, EVR & Wahlings, Stockholm, Sweden, 1995

Yi Jiang , Zhenxi Xu, Feng Chen, 1994, *TUHVAC - an object oriented computer software for HVAC design, analysis and simulation*, preprints of the 4th Intnl. Conf. on System Simulation in Buildings, Dec, 1994, Liege, Belgium

Paper 1, best paper award at the SIMS88 conference, VTT, Helsinki, Finland, April, 1988

MODSIM

a Program for Dynamical Modelling and Simulation of Continuous Systems

Per Sahlin

The Swedish Institute of Applied Mathematics[1]
Box 26300
100 41 STOCKHOLM, SWEDEN

Abstract

A software package for continuous simulation of component based systems, Modsim, is presently under development at the Swedish Institute of Applied Mathematics (ITM). This package is introduced along with the concept of dynamical modelling and simulation (DMS).

Most existing general purpose simulation tools provide insufficient support for the mathematical modelling of physical systems. Usually, the system is described in terms of equations or assignment statements and all users must understand and interact with the simulated system at this low level of abstraction. Contrary to this, DMS programs allow certain classes of users to describe systems at a higher level. Components are the basic building blocks rather than equations. A component in this sense is a physical subsystem, such as a heatpump or the condenser of a heatpump. Component parameters are given from an engineering rather than from a mathematical point of view.

Several present component based packages require predefined signal directions to and from an individual component. Some of the limitations with this input/output approach is pointed out briefly and the possibility of building a system without these limitations is discussed.

The Modsim system is built around a Fortran solver, Modsol, for the integration of differential algebraic systems of equations (DAE) with the particular structure arising from signal direction independent component modelling. The interactive parts of Modsim are written in Lisp and rely heavily on the graphical communication tools of a modern low-end workstation. Modsim is not bound to a particular field. Application dependent parts may be designed separately, thus allowing the core of the system to be used in several fields. ITM's foremost concern is with this core but application dependent parts are written concurrently in cooperation with researchers from the building energy simulation field.

An in-house demonstration version of the program, with a small component library, is planned for the summer of 1988.

---

[1] The Swedish Institute of Applied Mathematics (ITM) is an independent research organization supported by the Swedish National Board for Technical Development (40%) and by a group of Swedish companies (60%). This work was supported in part by the Swedish Council for Building Research under contract no. 870299-8.

1 INTRODUCTION

New workstation hardware with powerful processors and advanced graphical capabilities will put new requirements on the next generation of simulation software. Simulation tools will no longer be exclusively used by specially trained computational engineers. The designers of tomorrow will want to user their 10 MIPS of desktop processing power for something more than computerized drawing boards.

Most simulation papers are written from the perspective of a single application field. In this paper, the focus will be on practical design tools for the simulation of dynamical systems from a general perspective. We focus on simulation tasks where application dependent parts should be added last, not first. With this approach, it is possible to avoid many of the problems with "independent groups creating non-interchangeable software, continually re-inventing different variants of the wheel", quoting J.A. Clarke, a visionary in the field of building design [1]. For applications such as rigid body mechanics and electrical network analysis – where a well defined formalism for generating the governing equations exist – special purpose systems may be more advantageous.

At ITM the objective is to bring forth simulation software which can be of use in several application fields. The programs should, however, not be regarded as finished products but rather as raw material for such.

The present work deals with *modular* dynamical systems, i.e. systems that naturally decompose into subsystems. The dynamics of the systems under consideration are described by sets of nonlinear ordinary differential and algebraic equations (DAE). (PDE:s are reduced to ODE:s by discretization.) Some discontinuities in the system description will be allowed such as those which are introduced by automatic controllers, but inherently discontinouos systems (event driven systems) such as the product line of a manufacturing plant, are excluded.

Physical systems that fall into the category under consideration occur in several fields. The following applications are those which have been touched upon by ITM in the last few years:

- Process industry. ITMs work on modular simulation started with a study of simulation methods for oil-gas separation plants. The study included an evaluation of existing "dynamical flowsheeting" programs and since they proved inadequate from a numerical point of view, a new numerical algorithm was proposed [9–11, 13]. The Modsim project is based on a further developed version of this algorithm.

- Energy and temperature simulation in buildings and their HVAC systems. This is the current test application for the Modsim project. ITM cooperates actively with researchers in this field for the development of application dependent parts.

- Dynamical heat pump simulation. A pilot programming project, Heatsys, was carried out by the author [7].

Many additional items should be included in a complete list of potential applications.

In the first section of this paper the concept of Dynamical Modelling and Simulation (DMS) will be defined and discussed. The remaining sections are devoted to the introduction of Modsim – an attempted DMS program.

2 DYNAMICAL MODELLING AND SIMULATION

Simulation tools can be used for various aspects of physical system development and maintenance. For some aspects the simulation model may be more or less fixed. The basic interconnection structure of the submodels may at least be fixed since most of the necessary model changes will occur within the submodels. Examples range from operator training and operability studies to monitoring of unaccessible process variales by interconnected real-time simulation models.

Here, our primary focus will be on simulation tools for the design phase. The phase when nothing yet is clear, not even the basic structure of the desired system and when the possibility of quick evaluation of different ideas is crucial. This is when a powerful simulation tool might make the difference for a creative engineer who is trying to convince his superiors of the benefits of a slightly unconventional solution.

The design tools we are discussing must do more than just the actual simulation. They must form an environment where strong support is given for the modelling process as well. The system model should obviously be time dependent or dynamic and since also the actual modelling should be dynamic (in the sense *active*), we propose to call this class of tools *Dynamical Modelling and Simulation* (DMS) programs.

Some requirements of such a simulation environment or DMS program is discussed in the following subsections.

2.1 The User Interaction

One ambition with DMS programs is to bring the power of simulation to wider circles. A DMS user should not necessarily be identical with a skilled mathematical modeller. Any engineer with a solid physical understanding of a system should also be able to simulate the system. This puts extreme requirements on the man-machine interaction and on the robustness of the numerical methods used.

A key issue for user interaction is data abstraction. Subdivision into components simplifies the understanding of complicated systems. The graphical representation of the simulated system must take full advantage of this. The user should only see the information which is relevant to his problem and to his level of sophistication. Major contributions to this field have been made by H. Elmquist, S.E. Mattson et al [2, 6].

The basic building blocks in the modelling process should be components rather than equations. The components should in addition be characterized in a language which is

natural for the user – with familiar parameter names – rather than in terms of abstract mathematical coefficients. Furthermore, the underlying mathematical model should behave in a physically correct way under extreme conditions or the user should at least be signaled when the model is operating out of its domain of validity.

The user should be able to follow the progress of the simulation on the screen and have the means of stopping it at any time. Iterative model development with separate simulation of subsystems should be supported and the transition between simulation and modelling mode should be virtually instantaneous.

If the resulting mathematical model becomes ill-posed or unsolvable, a message should be issued which if possible indicates recommended courses of action.

The mathematical model will contain algebraic as well as ordinary differential equations. Unfortunately, DAE systems are often difficult to solve by the numerical methods of today. Seemingly innocent problems can be impossible to integrate even though they are well-posed in a mathematical sense. Mathematicians say these problems have high *index*. As of today, there is not even an algorithm for safe diagnosis of this condition. This is a serious obstacle on the road to a true DMS program, which of course at least should be able to predict the problem and preferably also solve it automatically. These problems and their relation to mathematical modelling are surveyed in Mattson [5].

2.2 The Modelling Method

The mathematical model of the simulated system must be general enough to allow several different combinations of given (input) and calculated (output) variables. An analogy can be drawn to an equation, where any variable can be solved for as long as the remaining variables are given. Traditionally, submodels in simulation programs have been identical with subroutines. For subroutines, certain arguments are given (the input to the submodel) and the the remaining arguments are returned (the output from the submodel). If a user would like to answer the reverse question "What input will result in this output?", he will be forced to either rewrite the subroutine or to run the subroutine over and over and slowly find an approximate solution. This is clearly an unacceptable state of affairs for a DMS program, where the submodels in the library must be general enough for both questions. Indeed, the user should not even have to be skilled enough to rewrite the subroutine or run it repeatedly. The only way to attain this desirable flexibility is to model the submodel as a system of equations rather than as a series of assignment statements (a subroutine). A system of equations where any chosen set of variables can be solved for, as long as the problem is still well-posed.

Another requirement on a DMS program is that submodels should be arbitrarily connectable. This also requires equation rather than assignment modelling. Let us illustrate this by looking at the assignment modelling of a resistor.

$$\begin{aligned} i1 &:= (u2 - u1) / R \\ i2 &:= -i1 \end{aligned} \qquad (1)$$

The fact that this is a purely algebraic model is no restriction here. Each terminal, 1 and 2, carries two variables, i and u, current and voltage. Let i1 and i2 be the output and u1 and u2 be the input, as suggested by the assignments.

Now suppose two such resistors, r1 and r2, are connected in series. The resulting coupling assignments are, using dot notation:

$$\begin{aligned} \text{r1.i2} &:= -\text{r2.i1} \\ \text{r1.u2} &:= \text{r2.u1} \end{aligned} \quad . \tag{2}$$

The first assignment in (2) is a relation between output variables and the second one consists of input variables only. We clearly have a problem. In order to perform the calculation, there must be one input – receiving a value from an output – in each assignment.

In principle, the problem could be temporarily solved by choosing the variables differently, so that the input would match the output. But this would only lead to further problems. If the designation of input and output variables is done ahead of the interconnection phase, two separate assignment models of the same resistor are required in order to make all interconnections possible. The additional one is:

$$\begin{aligned} \text{i1} &:= -\text{i2} \\ \text{u1} &:= \text{u2} + \text{R} * \text{i2} \end{aligned} \quad . \tag{3}$$

For multi-terminal components, even more models may be required. And for every major change, all the separate models for each component may have to be changed. The input-output strategy is clearly impractical.

2.2.1 Symbolic manipulation

One alternative approach is to model submodels as systems of equations and treat them with computerized paper-and-pencil methods, i.e. write down all the equations including coupling equations and simplify.

The above example can be written:

$$\begin{cases} R_1 i_{11} = u_{12} - u_{11} \\ i_{12} = -i_{11} \\ R_2 i_{21} = u_{22} - u_{21} \\ i_{22} = -i_{21} \\ i_{12} = -i_{21} \\ u_{12} = u_{21} \end{cases} , \tag{4}$$

where an extra index has been added for the component number. The system (4) simplifies to e.g.

$$\begin{cases} i_{11} = (u_{22} - u_{11}) / (R_1 + R_2) \\ i_{22} = -i_{11} \end{cases} \tag{5}$$

The automatic symbol manipulation can be carried out by using known techniques. This way a significantly smaller system of equations can be obtained, consisting of interesting entities only. This will often – but not always – make the numerical solution less demanding in terms of processing power. A drawback is that any changes in the equation system structure, e.g. changing the model of one of the submodels, usually necessitates a new symbolic reduction. This is normally a quite costly process.

2.2.2 Residual Form

Yet another alternative is to represent the equations in residual form and solve the whole system numerically as it is.

$$\begin{cases} 0 = -R_1 i_{11} + u_{12} - u_{11} \\ 0 = i_{12} + i_{11} \\ 0 = -R_2 i_{21} + u_{22} - u_{21} \\ 0 = i_{22} + i_{21} \\ 0 = i_{12} + i_{21} \\ 0 = u_{12} - u_{21} \end{cases} \tag{6}$$

The resulting system matrix will be sparse and this method is likely to be competitive only if the sparsity is utilized. One advantage is that changes in the system structure only require rewriting the system matrix. Another one is that the method does not require any automatic symbol manipulation, and hence it is more straightforward to implement.

Both of the latter methods and combinations of them represent interesting alternatives for DMS programs. They must be implemented, tested and compared in several development projects. Prior to this, it is too early to say which one that will ultimately prove the most effective. It is likely that the optimal choice depends on the application as well as on the mode of operation, i.e the amount of iteration between modelling and simulation which is to be done. This has been observed in packages for the simulation of rigid body mechanics, where successful implementations of both approaches are in use today.

3 MODSIM – A DMS PROJECT

The starting point of the Modsim project was the conception of an efficient numerical algorithm for the integration of DAE systems with the special sparse structure obtained with the residual approach. This algorithm has been implemented in a Fortran solver, Modsol, which now is the heart of the Modsim system.

On top of the Fortran solver is a layer of interactive Lisp routines, which are operated by the user in the MacIntosh style. The entire system is coded in a highly modular open fashion. This allows various interfaces with e.g. CAD programs, general engineering data bases or other application dependent programs to be added to the application independent core.

3.1 Component Models, Data Instances and Objects

First, let us define more precisely what is meant here by these terms. A *component model* is a computer model of a physical subsystem such as a pump, a pipe, a wall, a thermostat or even a room. A *data instance* is a collection of physical parameters, e.g. the physical size, and connection references, which characterize a single component. Several data instances may share the same component model. Together a data instance and its component model are referred to as an *object*. The component model breaks down into several parts which perform different operations on the data instance. Each of these parts is called a *method*. Operations include e.g evaluation of equation residuals and screen presentation of the data instance. In Modsim, objects are separated into four groups depending on the way they are modelled: equation objects, algorithmic objects, macro objects and boundary objects.

The most fundamental one, the equation object, contains a set of ordinary differential and/or algebraic equations, modelling the behavior of e.g. a pump or a resistance. It is simply a small differential algebraic system, where the time history of certain variables can be calculated as soon as the remaining variables are given, as functions of time. However, at the time of the component modelling, it is not yet clear which variables will ultimately be given as input and which ones will be calculated as output. Only a temporary selection, which may later on be changed, is done at this stage.

In algorithmic objects, matters are simpler; here the designation of input and output variables can be done at the component modelling stage. The input variables are given as input to an algorithm, which in turn calculates the output variables. Control components such as thermostats are frequently modelled as algorithmic objects.

Macro objects do not have mathematical models of their own, they only contain references to other underlying objects. Their sole purpose is to simplify the modelling and the object administration. It is often convenient to operate on whole groups of objects at once.

Boundary objects are used to provide the simulated system with time dependent input. The details of this process is beyond the scope of this paper. Here, the main focus will be on equation objects, since they are the most fundamental.

In Modsim, the methods of an object are separated into two groups: manipulative and numerical. These groups together with some coordinating code for each group, form the Lisp and the Fortran modules of the Modsim code. Manipulative methods aid the user in the interactive work of building and editing a network of objects into a meaningful simulation model. Most of these methods are included in the application independent core and will be accessed automatically when a new component is introduced. The bulk of the work involved in writing a new component model lies on

the numerical side. In the most straightforward case of setting up a simple equation object, three Fortran procedures are required: one for calculating equation coefficients from user specified parameters, one for the calculation of equation residuals, and one for Jacobian evaluation. For more sophisticated equation objects requiring special methods of integration or differentiation (in order to calculate Jacobians numerically) additional procedures must be provided.

In the remainder of this section, the basic ideas of the Modsim system will be presented. Initially, we will be dealing with the numerical description of equation objects and then we move to successively higher levels of the program, finally arriving at the user interface. At the end of each subsection there will be an example, Sample System, which may enlighten the curious but which can be omitted on first reading.

3.2 Numerical description of the simulated system

This paper deals with the basic structure of the Modsim system. No particulars about the underlying numerical algorithm are treated. These are included in an accompanying paper by G. Söderlind. L.O. Eriksson and A. Bring [12]. Here, the numerical discussion is limited to what a user must know in order to introduce a new component model.

All the differential and algebraic equations for the continuous part of the simulated system are solved simultaneously. All difficulties with algebraic feedback loops are avoided this way, without artificial delays or other tricks common in many of the programs of today.

The mathematical model of the *i*:th equation object is represented in the following way:

$$E_i \dot{x}_i = f_i(x_i, u_i, p_i) \,, \tag{7}$$

where the matrix E_i may be singular, signifying the presence of algebraic equations in the model. x_i is the variable vector, which can be calculated as soon as the variable vector $u_i(t)$ and the parameter vector p_i are given. If the equation object was operated separately, without neighboring objects, x_i could be interpreted as the state of the module and u_i as the external input. Formally, the included variables in the x- and u-vectors should be chosen so that the matrix (pencil) $\alpha E_i - A_i$, where $A_i = \partial f_i/\partial x_i$, is non-singular for some α, while $B_i = \partial f_i/\partial u_i$ may become singular.

The full model, including all individual object models and interconnections, can be written

$$\begin{cases} E\dot{x} = f(x, u, p) \\ E_u u = E_x x \end{cases} , \tag{8}$$

where E is a block diagonal matrix with each block E_i and the corresponding segment of the x vector emanating from an individual object model. (8b) represents the couplings between objects. Each equation in (8b) connects two variables from separate

objects together. The elements of E_u and E_x are either 1, -1 or 0. It is not necessary to always bind u-variables with x-variables, x-x and u-u connections are also permitted, thus allowing signal direction independent modelling. (8b) should only be regarded as a way of computing residuals in the coupling equations. E_u and E_x may both be singular.

3.2.1 Sample System – the equation model

Sample System is a purely academic example, built with components from the building energy simulation field. Its sole purpose is to illustrate some of the ideas behind Modsim.

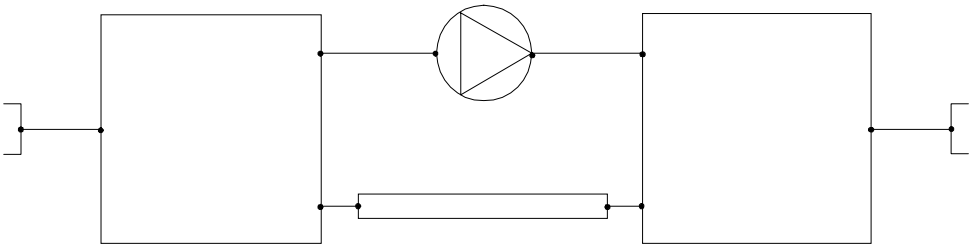


Fig. 1. Sample System

The system consists of a pump and a pipe, transporting a heated fluid between two subsystems in a never ending loop. Each subsystem contains a tank and a thermal conductance, connecting the system to the outside world.

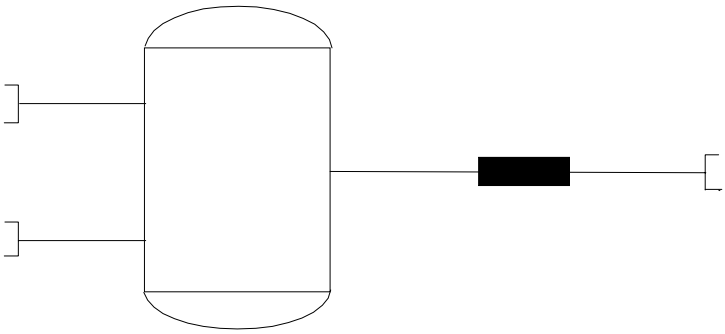


Fig. 2. A subsystem of Sample System

The conductance model is purely algebraic, with only one equation

$$q = kA(T_1 - T_2) , \qquad (9)$$

where q is the state variable for the conducted heat, kA is the conductivity parameter, and T_1 and T_2 are the u-variables for the terminal temperatures.

The tank model is also quite simple, assuming perfect mixing of the entering fluid.

$$\begin{cases} mc_p \dot{T} = q + q_{f1} + q_{f2} \\ k\dot{p} = w_1 + w_2 \end{cases}, \tag{10}$$

where T and p are the state variables temperature and pressure. mc_p and k are the parameters thermal mass and compressibility. q, q_f and w are the u-variables of the model signifying conducted heat (from the conductance), convected heat (from the pump (q_{f1}) and the pipe (q_{f2})) and thermal mass flow (from the pump (w_1) and the pipe (w_2)), respectively.

The pipe model assumes a pipe without thermal mass and with a simple friction parameter, β, governing the flow rate through the pipe.

$$\begin{cases} w = \beta(p_1 - p_2) \\ q = wT_1 \end{cases}, \tag{11}$$

where w and q are the state variables thermal mass flow and convected heat. p_1 and p_2 are the pressures at the inflow and outflow terminals, respectively, and T_1 is the temperature at the inflow terminal.

The pump model is similar, but with some additional parameters (a,b,c,d, and switch) allowing the thermal mass flow a more complicated dependence on the pressure difference:

$$\begin{cases} w = \text{switch} \cdot a\left(1 + b(p_2 - p_1) + c(p_2 - p_1)^2 + d(p_2 - p_1)^3\right) \\ q = wT_1 \end{cases} \tag{12}$$

The equations are described by the user in terms of two Fortran subroutines for each model describing the Jacobians (A,B and E) and the equation residual. Routines for the thermal conductance are provided for reference in fig. 3. These routines are called by the solver, Modsol. The coupling matrices E_u and E_x are generated automatically by the system as the objects are connected together graphically on the screen during the interactive modelling process.

```
* ROUTINES FOR THERMAL CONDUCTANCE
```

```
* JACOBIAN ROUTINE

    SUBROUTINE THERCDJ (
   & NX, NO, NP, NMP,
   & E, AB, PAR, MPAR, XU)

    INTEGER NX, NU, NP, NMP, MPAR
    REAL E, AB, PAR, XU
    DIMENSION
   & E(NX, NX),
   & AB(NX, NX+NU),
   & PAR(NP),
   & MPAR(NMP),
   & XU(NX+NU)

* PARAMETERS IN:
*  NX         NUMBER OF X-VARIABLES
*  NU         NUMBER OF U-VARIABLES
*  NP         NUMBER OF PARAMETERS
   NMP        NO. OF MODEL PARAMETERS
*  PAR(NP)        PARAMETER VALUES
*   PAR(1)         k * A
*   MPAR(NMP)      MODEL PARAMETERS
*    –
*   XU(NX+NU)      X- AND U-VARIABLES
*   XU(l)          Q
*   XU(2)          T1
*   XU(3)          T2
*   E(NX, NX)      CLEARED TO ZERO
*   AB(NX, NX+NU) CLEARED TO ZERO
*
* PARAMETERS OUT:
*  E(NX, NX)       DERIVATIVE WRT X'
*  AB(NX, NX+NU)  DERIVATIVE WRT XU

* WRITTEN BY:   A BRING 8802

* CALCULATE 'E'
*  LEAVE CLEARED FOR ALGEBRAIC EQN

* CALCULATE 'AB'
*  1 EQUATION –> 1 ROW
*  COLUMNS FOR 3 VARIABLES
    AB(1, 1) = 1.
    AB(1, 2) = –PAR (1)
    AB(1, 3) = PAR (1)

    RETURN
    END
```

```
* RESIDUAL ROUTINE

    SUBROUTINE THERCDR (
   & NX, NU, NP, NMP,
   & F, PAR, MPAR, XU)

    INTEGER NX, NU, NP, NMP, MPAR
    REAL F, PAR, XU
    DIMENSION
   & F(NX),
   & PAR(NP),
   & MPAR(NMP),
   & XU(NX+NU)


* PARAMETERS IN:
*  F(NX)       NOT CLEARED TO ZERO
*   OTHERS, SEE JACOBIAN ROUTINE
*
* PARAMETERS OUT:
*  F(NX)       VALUE OF  AB x XU

* WRITTEN BY:    A BRING 8802


* CALCULATE F

   F(1) = XU(1) –
         PAR(1) * (XU(2) – XU(3))

   RETURN
   END
```

Fig. 3. Jacobian and residual Fortran subroutines for the thermal conductance

3.3 Object Manipulation and Administration

In Modsim, the system matrix is not the only representation of the simulated system. Between the matrix description and the user lies a more flexible and complete one – the object representation.

This representation has one object for each physical component of the simulated system. The objects are interconnected into a network of the same structure as the physical system. It is by interacting with and changing this network of objects, that the user manipulates the system model. The underlying matrix is never seen by the user.

The objects of the Modsim system are organized hierarchically, much like the files in the directory system on the disk of an ordinary PC. Starting from the root macro (the root directory), with references to underlying macros (subdirectories) and objects (files), one can reach all available Modsim objects. But unlike PC-directories, the macro object in Modsim does not only serve as aids for book keeping in the object library. On the lower levels of the macro hierarchy, the macros also form meaningful systems for simulation. Thus, we have some macros in the system which only serve as object libraries, where the subobjects are unconnected, and others where the subobjects also are connected into system models. The latter category are potential *system macros*.

The first thing a user does, wanting to simulate or edit an existing system model, is to move down through the macro hierarchy until the model in question is reached. At this point he sets a flag indicating that this is the system he wants to work with. Suppose the user wants to add an additional object to the system model. This is always done by *copying* an already existing object and then subsequentially revising the copy. All the objects under the root macro are available for copying, not just those under the current system macro.

Every submacro under a system macro is also a potential system macro, i.e. it can be simulated as a separate system, given the proper boundary data. This simplifies model validation since models can be built and tested incrementally.

The Modsim *object definitions* are also organized in an hierarchical system, resembling the class system of the Simula language with inheritance and class variables. In this way, any code or data common to more than one component, only has to be stored once. Furthermore, if the information is changed, it changes for every component sharing it.

3.3.1 Sample System – the object representation

This far, we have given the impression that the macro object is the only object with subobjects. This is however not entirely true. All the objects we have mentioned up until now are built from smaller entities, subobjects. An equation object is, for example, built by putting several parameter, variable, and interface objects together. If the user elects to, two equation objects may even share the same parameter object. This can be very practical if several nearly identical objects all need to have the same

parameter changed. Let's say we want to change the thickness of all the exterior wall segments of a building model by editing only one position.

Parameter objects, or other objects, which are shared by more than one object are in fact lifted to the macro above them. They become the property of the macro rather than any particular subobject.

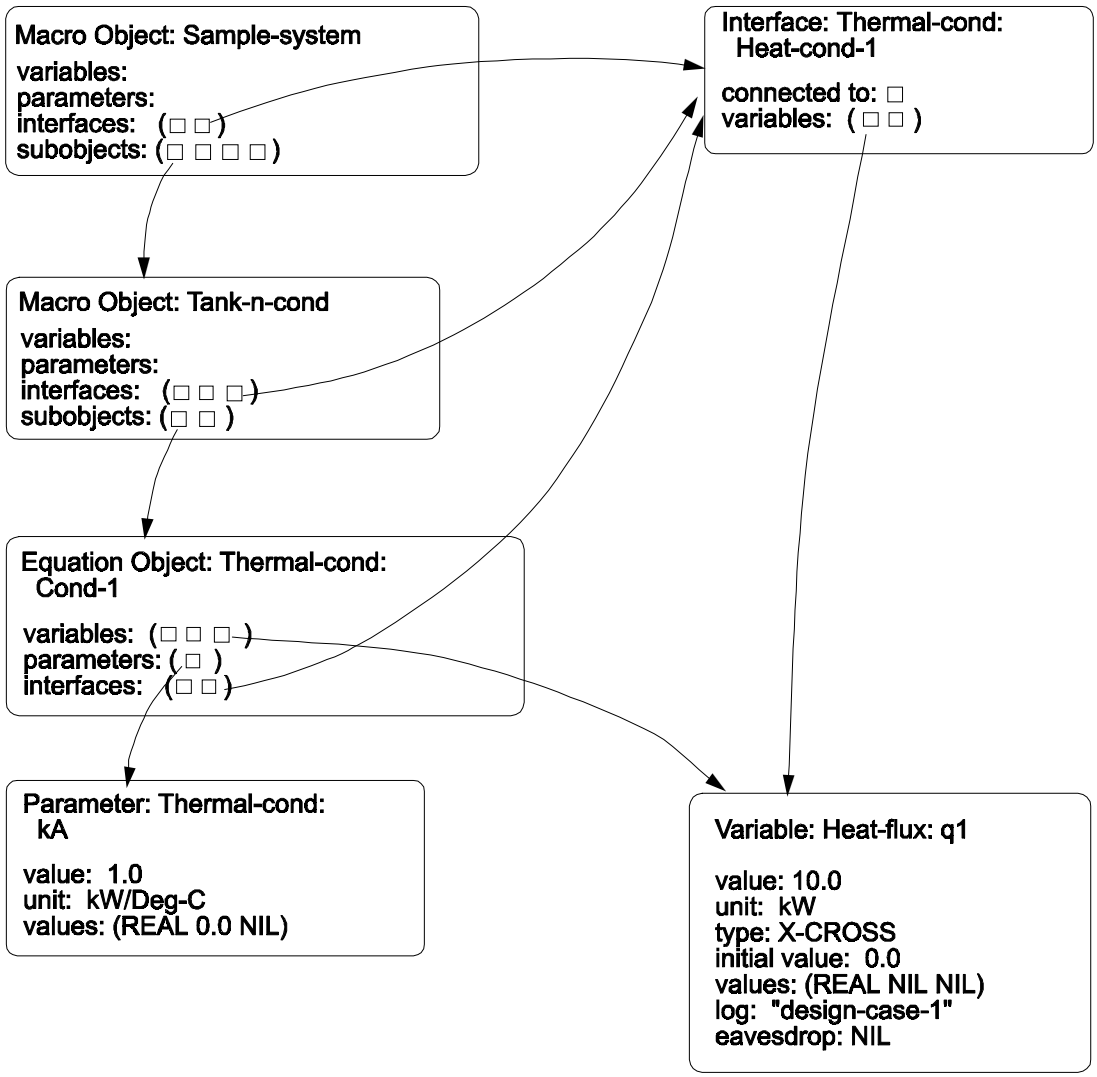


Fig. 4. A schematic representation of selected objects. The box-characters represent pointers to objects. Only pointers to objects within the selection have been drawn.

3.4 User Interface

The object description of the simulated system is complete. All the information is there, organized in a suitable way for the programmers. They can alter its structure and update it easily. Most regular users would, however, prefer the information to be presented differently. Depending on their level of sophistication additional explanatory pieces of information may be necessary. Some users might not need access to all details.

These differing needs are met in Modsim by using appropriate "information filters". We call them *forms*. A form is a set of instructions for presenting an object on the screen. Certain bits of information are concealed, others are highlighted. Forms provide explanatory text and pictures, not needed by the programmers.

The form also contains methods for manipulation of the data instance. For instance, the parameters required in the mathematical model of a component are frequently different from those which would be found natural by an engineer for characterizing the same component. In this case, the conversion routines, from the engineering language to the mathematical, are associated with the form. As a general rule, the object manipulation routines which are associated with the form are on a high level. They in turn call the manipulation methods of the object, which perform the actual manipulation.

The active instructions and routines of a form may change depending on the situation and on the user's level of sophistication. E.g. an object will look different on the screen depending on whether it is presented only for library purposes or if it is actually a part of the simulated system. Similarly, an expert user might be allowed to perform operations which are unavailable to the beginner. Furthermore, the form associated with a particular object is interchangeable. It can be specially tailored to suit the needs of a certain group of users, e.g. a language group. In this manner, Modsim takes on a different face depending on the identity of the user.

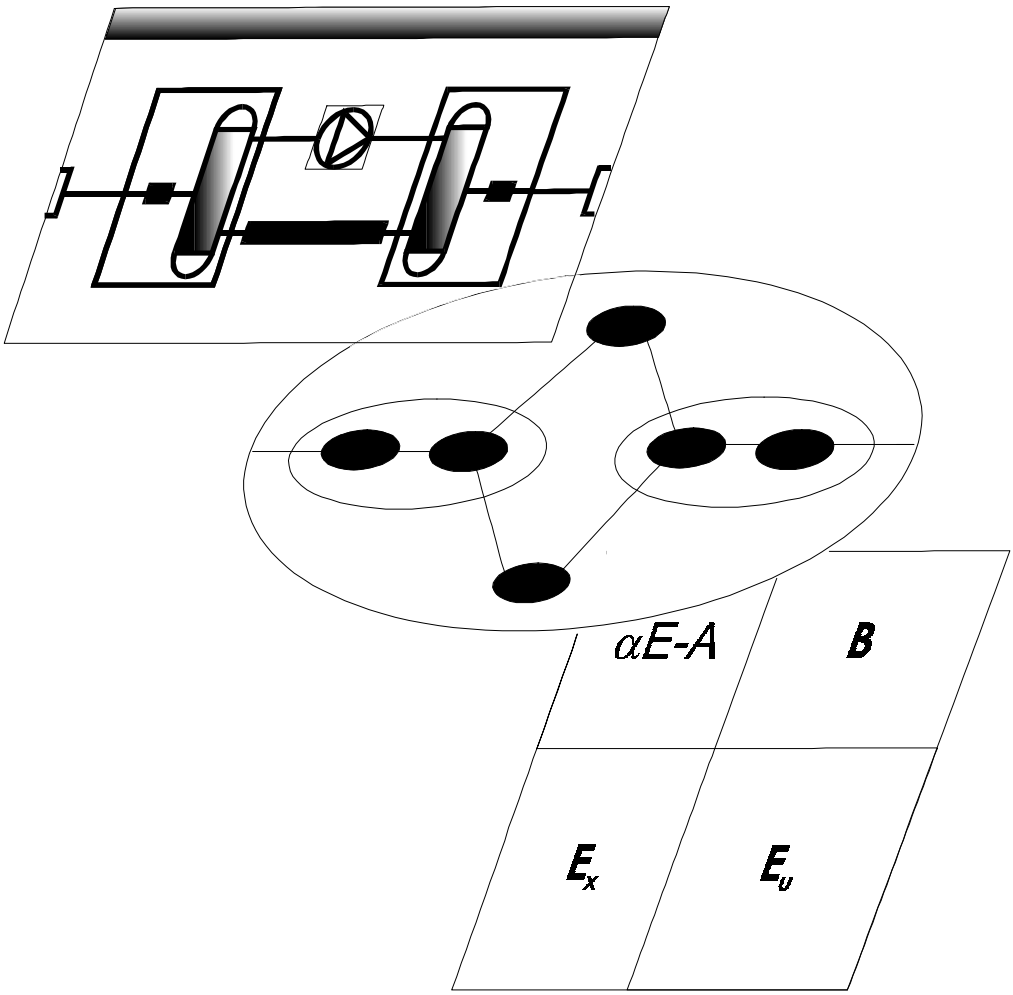


Fig. 5. *The screen, the objects and the system matrix*

3.4.1 User Categories

Component based simulation programs provide a lot of freedom for the user. Freedom to create new component models and to build new systems with them as building blocks. Along with this also comes the risk of making mistakes. Serious errors can be caused e.g. by incompetently made component models.

For this reason it is necessary to limit the freedom of certain users. In the Modsim program, users are separated into three different categories, with support and freedom according to the user's level.

The component maker is responsible for the modelling of new components. He has an understanding of the physical, mathematical and numerical aspects of modelling. He provides the models with explanatory texts and sets the parameter and variable ranges in which the model is valid. Programming in Lisp and in Fortran is done by the component maker.

The introduction of a new equation object involves the following steps:

1. Write three Fortran subroutines for the evaluation of Jacobians, calculation of equation residuals, and for conversion from user to model parameters.

2. Write the Lisp object definition, i.e. assemble the right combination of variables, parameters and interfaces.

3. Write a form for the screen presentation of the new object.

4. Write the on-line documentation for the new model, explaining its merits and shortcomings.

The system maker builds systems out of available components. He has a physical, engineering understanding of the simulated system but does not need a proficiency in mathematical modelling. He operates the program primarily by pointing and choosing in the MacIntosh style.

The following steps are involved in the making of a new system model out of available components:

1. Make a copy of an already existing similar system (or of an empty macro) and make it the system macro.

2. Delete and add (by copying) objects until all necessary objects are present. Move the objects to their proper positions on the screen.

3. Connect the objects with each other and with the boundary objects.

4. Set parameters and initial values by opening the objects and typing the new data over the old.

5. Mark the variables, that are to be logged.

6. Start the simulation. Follow the time development of certain key variables in order to see whether the model seems correct.

7. When the simulation is terminated, inspect the time series of the logged variables in the post processor.

If the system is to be simulated by other users with new input data some additional steps are required:

8. Lift parameters that are to be easily available to the other users, up from the equation object level to the macro level. Do the same with a selection of key variables.

9. Write a form for presenting and editing the lifted parameters – an input questionnaire – with suitable instructions for the inexperienced user. If input is to be given

in terms of brand names and model numbers, write a subroutine in Lisp or Fortran for the conversion of these into sets of physical parameters. Include the subroutine in the form.

The black box user runs available systems with new parameters. Each of the prepared systems and input questionnaires serves as a special purpose program, designed to produce a specific set of results, e.g. the annual energy consumption of a house. The input questionnaire may in this case ask for a few key parameters such as the areas of the exterior walls and the windows, the model number of the installed heatpump, e.t.c. These data are subsequentially converted into physical parameters and inserted into their proper places in the system model.

One major difference between running a Modsim model and running a special purpose program, written from scratch in e.g. Fortran, lies in the possibility of having changes made to the model at reasonable cost. To update an already existing model will be a quick operation for a system maker.

The distinctions between the three user roles have been exaggerated here for clarity. In practice, the same user will interact with the system in more than one of these ways. Switching between roles has been made easy and can be done incrementally, so that a user can move from one level to the next as soon as the interest occurs.

3.4.2 Sample System – the user interface

The Modsim user interface will use a MacIntosh style dialogue with panels, graphical symbols and pop-up menus. Since user interfaces look better live, this description will be kept brief.

The system macro is presented in a window occupying most of the workstation screen (fig. 6). An object, such as the system macro, together with its instructions for screen presentation (its form) is here called a *panel*. The user can perform operations on the system macro panel by pressing screen buttons, invoking pop-up menus or opening submacro panels. All is done by moving the mouse around and pressing its buttons.

A screen button invokes a procedure associated with the form. (See fig. 7. for a sample form.) Button procedures perform operations which may have consequences outside of the current panel, e.g. "Start simulation". Panel procedures may be shared between several panels in need of the same operation, e.g. "Close panel", or may be private to a single panel, e.g. "Start black box simulation" for running a tailor made questionnaire series for a specific system. The pop-up menus of the panel present a selection of operations which can be performed on the panel itself. Examples are "Connect interfaces", which connects objects and "Move field" which moves any of the panel screen objects to another position.

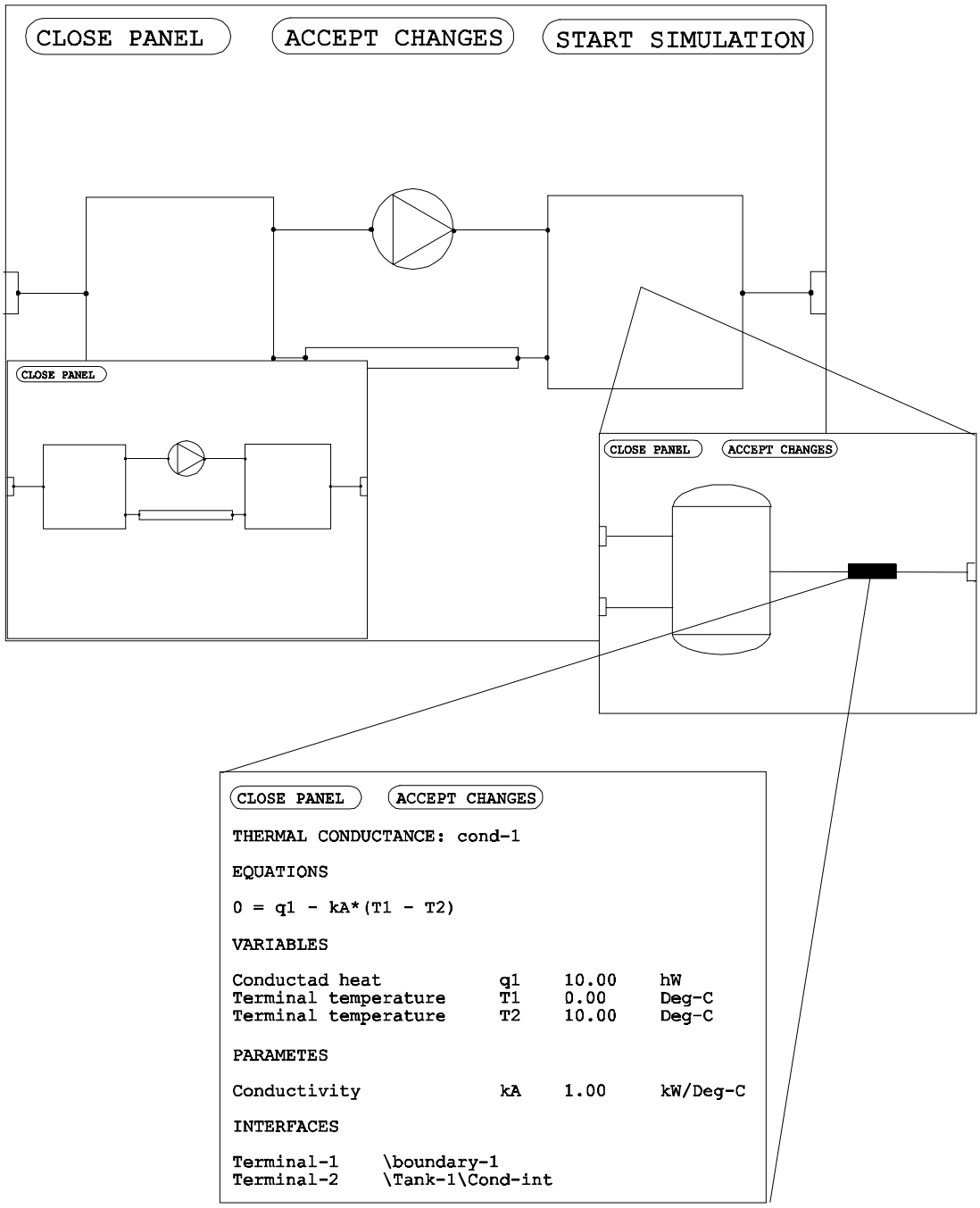


Fig. 6. A schematic representation of the Sample System panel and some subpanels. The library panel (lower left corner) normally displays some other macro.

```
(:name "Sample System"
      :size
      (74 35)
      :symbol
      NIL
;;; Fields are visible panel objects.
      :fields
      ((BUTTON :coord (1 1 17 3) :context (SYSTEM) :user (ALL) :tools NIL
            :body ("Accept changes" PANEL))
       (BUTTON :coord (18 1 31 3) :context (ALL) :user (ALL) :tools NIL
            :body ("Close panel" PANEL))
       (BUTTON ...
            :body ("Start simulation" PANEL))
       (SUBPANEL :coord (7 5 24 17) :context (ALL) :user (ALL) :tools NIL
            :body
            (:closed TANK-N-COND-1
               NIL
               (SUB-CHAR-SIZE CURRENT-CHAR-SIZE CURRENT-CONTEXT)
               (SUB-CONTEXT CURRENT-CONTEXT)))
      (SUBPANEL :coord (33 5 41 10) ...
            :body
            (:closed PUMP NIL 2 (SUB-CONTEXT CURRENT-CONTEXT)))
      .
      .
      .
      (SUBPANEL :coord (64 30 70 31) :context (SYSTEM)
            :user (COMPONENT-MAKER MODEL-MAKER)
            :tools NIL
            :body (:open *ROOT-MACRO* NIL 1 'LIBRARY)))
;;; The tools of a form are invoked by a pop-up menu. This menu also holds
;;; additional tools which are common to all panels, e.g. "Edit field".
       :tools
       ((:name "Connect instance"
            :function-ref MS-CONNECT-INSTANCE
            :context (SYSTEM)
            :user (COMPONENT-MAKER MODEL-MAKER)))
;;; The list of procedures is also completed be generic ones such as "Close panel".
       :procedures
       ((:name "Start simulation"
            :function-ref MS-START-SIMULATION))
       :subforms
       NIL)
```

Fig. 7. The form associated with the Sample System macro. Left-out items are indicated by three dots.

4 CURRENT STATE OF THE MODSIM PROJECT

The first version of the Modsim code is yet to be completed. A great deal of programming has already been done, but much still remains.

4.1 The Integrator

At present, the continuous integrator has been written and to some extent tested. The implemented algorithm belongs to a class of modified backward differentiation methods called MOLCOL methods [3]. These have, during recent years, proven to be very effective on problems with widely differing timescales, so called stiff problems. The MOLCOL methods include, by certain choices of parameters, the backward differentiation methods (or Gears methods) such as the backward Euler. Other included methods are e.g. the implicit midpoint method. If additional integration schemes are needed for the full system, or for a single component, they can easily be implemented within the Modsim framework.

Presently, the numerical work mainly concerns the development of program modules for automatic calculation of initial values that satisfy the algebraic part of the DAE system. Concurrent with this further testing of the integrator on more realistic problems is being carried out.

The next task will be to develop additional code for handling discontinuities. Most dynamical systems occurring in industry are automatically controlled. Controllers introduce discontinuities. Hence, it has been considered essential that Modsim has capabilities to model controllers along with the systems they control.

4.2 Implementation

All non-numerical routines in Modsim are coded in Common Lisp. After many years as a rather exotic language, primarily used by a few experts, Lisp is gradually breaking through into industry. The main reason for this is the acceptance of Common Lisp as a de facto standard, which allows production of portable code. Unfortunately, Lisp's reputation as an AI-language has led people to think that it is unfit for more general applications. This, however, is incorrect. The flexible data structures of Lisp along with several other favorable properties, make Lisp a perfect choice for many projects.

Several object packages such as Flavors or Loops have been developed as Lisp add-ons. None of these has the necessary properties for the Modsim project. This has forced us to develop our own package, with suitable features for the project. The work with this object package is nearly finished and the first Modsim objects have been created and manipulated, using the package.

4.3 The User Interface

The first panels have been opened on the screen and manipulated with the mouse, but a great deal of code still remains to be written before the user interface can be demonstrated.

A small window system has been written, modelled after the GKS graphics standard. This was primarily done in order to get something immediately operational. Depending on the development of window system standards, such as X-windows, the GKS based window system may be replaced eventually.

4.4 Development Time Frame

A first version, for in-house demonstrations only, is planned for the summer of 1988. This version will include a small component library for energy and temperature simulation of buildings.

A full commercial quality system, excluding application dependent parts, will take considerably longer to finish. A very rough estimate is between four to eight man-years.

It is our hope, however, that industry will show enough interest in the project to allow these man-years to be converted into considerably fewer calendar years.

5 SOME FINAL REMARKS

Many of the features discussed in this article will be included in the next generation of simulation software. Additional features, related to artificial intelligence, have been omitted. Questions related to these will briefly be mentioned here in order to stimulate the further discussion.

Some people believe that simulation tools will be integrated into the CAD-systems of the near future. In such a system, the simulation model would be formulated automatically, without human interaction.

In our opinion, the totally automated systems are a long way off. The decisions that have to be made in order to set up a reasonable simulation model, will not be made by computers within a foreseeable future. The systems of the next generation, including Modsim, will however feature automatic data transfer between a CAD data base and the simulation program. Thereby, the process of entering input data will be speeded up considerably.

Another interesting issue regards distributed system development. In the new systems, component modelling will be done at several locations. Models will be exchanged between different groups. This is one of the advantages with the new approach. However, this will also, most likely, be a cause of problems. Poor models will be in circulation, causing problems which might be difficult to detect.

One possibility is to have a central organization – preferably with a commercial interest – which is responsible for maintenance and development of the application independent core. Around this organization there can be several groups, commercial or non-commercial, each maintaining application dependent component libraries and other add-ons for a certain market segment.

Acknowledgements

The ideas presented in this paper have evolved during many fruitful discussions. The author has merely written some of them down. The following people have, among others, participated in these discussions: from ITM, Gustaf Söderlind, Magnus Lindgren and Lars Eriksson and from the Division of Building Services Engineering at the Royal Institute of Technology, Stockholm, Axel Bring, Tor-Göran Malmström, Göran Olsson, and Engelbrekt Isfält.

References

[1] J.A. Clarke (1986): *The Energy Kernel System,* draft paper submitted to the systems simulation conference, University of Liége, December 1986

[2] H. Elmqvist (1985): *LICS – language for implementation of controlled systems,* Lund Institute of Technology, LUTFD2/(TFRT-3179)/1-130/1985

[3] L.O. Eriksson (1983): *MOLCOL – an implementation of one-leg methods for partitioned stiff ODE's,* the Royal Institute of Technology, Stockholm, TRITA-NA-8319

[4] E. Eitelberg (1982), *Modular simulation of large stiff systems,* in "Progress in Modelling and Simulation", F. Cellier (ed.), Academic Press

[5] S.E. Mattsson (1986): On Differential/Algebraic Systems, Lund Institute of Technology, CODEN LUTFD2/(TFRT-7327)/1-026/1986

[6] S.E. Mattsson, H. Elmqvist and D. Brück (1986): *New Forms of Man-Machine Communication,* Lund Institute of Technology, CODEN LUTFD2/(TFRT-3181)/1-025/1986

[7] P. Sahlin (1986): HEATSYS – ett program för simulering av värmesystem, Institutionen för Mekanisk värmeteori och kylteknik, KTH

[8] E.F. Sowell, W.F. Buhl, A.E. Erdem and F.C. Winkelmann (1986): *A Prototype Object-Based System for HVAC Simulation,* presented at the systems simulation conference, University of Liége, December 1986

[9] G. Söderlind (1985): *Numerical Computation and Data Communication in Modular Simulation,* ITM internal report, March 22, 1985

[10] G. Söderlind, L.O. Eriksson (1985): *Modular Simulation: Theoretical Aspects and Numerical Experiments,* ITM internal report, September 16, 1985

[11] G. Söderlind, L.O. Eriksson (1986): *On the Design of a Modular Simulation System,* ITM internal report, January 24, 1986

[12] G. Söderlind, L.O. Eriksson (1988): Numerical Methods for the Simulation of Modular Dynamical Systems, to appear

[13] G. Söderlind, J. Oppelstrup (1984): *The Modern Process Simulation Environment: Modelling and Numerical Methods,* ITM internal report, October 19, 1984

Paper 2, presented at the IBPSA Building Simulation '89 conference, Vancouver, Canada, June, 1989

A Neutral Format for Building Simulation Models

Per Sahlin, Research Secretary
Swedish Institute of Applied Mathematics
Box 26 300, 100 41 STOCKHOLM, SWEDEN
and
Department of Building Services Engineering
Royal Institute of Technology, SWEDEN

Edward F. Sowell, Professor
Department of Computer Science
California State University Fullerton
Fullerton, CA 926 34
and
Department of Building Services Engineering
Royal Institute of Technology, SWEDEN

ABSTRACT

Much research has been directed towards development of software environments that allow easy construction of building simulation models of widely varying structure and purpose. For example, TRNSYS has been in use for a number of years. Recently, several new such environments have been proposed. In spite of a considerable variation in model description formats among environments, the underlying mathematical models of physical processes are very similar. While one of the principal aims has been to allow easy sharing of models between users of the same environment, it has not been clear how portability was to be provided between different environments. Another objective has been ease of component model definition, in order to encourage modifications and additions to model libraries. This paper addresses both of these issues, by proposing a neutral and natural format for component model expression. The proposed format encourages equation based model definition because such models can be converted to efficient algorithmic form if needed, whereas the converse is not always true. Nonetheless, algorithmic component descriptions are also supported in order to allow reuse of existing models. Other key features of the proposed format are typing and declaration of linkage elements between models, which allow development of compatible component families, and enhance model exchange and reuse. The proposal considers underlying system modeling issues, including hierarchical submodel decomposition and methods for formal model expression that allow automatic translation to various simulation environments. Also discussed are the software tools needed for library maintenance and model translation.

1. INTRODUCTION

There are currently several modular programs in use for simulation of buildings and associated service systems, e.g TRNSYS, HVACSIM+ and NEPTUNIX. Additionally, several new modeling environments for the same purpose have recently been proposed, and are in various stages of development [CLARKE 1985, SOWELL 1986, SAHLIN 1988]. All of these alternatives are similar in the sense that the mathematical models of components and subsystems are expressed in program modules that the user can interconnect as needed to define the wanted system model. The usefulness of any such environment depends on the availability of a library of predefined models for components in the intended application area, and on the existence of a simple mechanism for implementing new models when needed.

One might expect that component models could be interchanged among environments because, at a given level of idealization, the mathematical models of the physical processes are virtually the same. Unfortunately, this is not necessarily the case because each environment employs its own semantics and syntax for model expression and interconnection. Without some form of standardization of component model definitions the desired portability will be provided, at most, within modeling environments, but not between them.

This paper suggests a possible starting point for such a standard, namely a Neutral Model Format (NMF). The format is "neutral" in the sense that models are expressed in a general manner, rather than in the format of any existing or planned environment. The standardized definition encompasses only the essential information needed to express a model unambiguously. This information is formalized in order to allow automatic translation to the format of a particular simulation environment. The format is "natural," meaning that the definition employs terms and constructs as close as possible to the experience and training of scientists and modelers.

The focus, in this initial work, is on models with a basically continuous behavior. This includes building envelope as well as HVAC system models. Excluded are components such as thermostats with a dead band

(hysteresis) and micro processor based controllers, which are better described in discrete time. Furthermore, the emphasis is on the machine readable mathematical description of the models. Systematic model documentation has been treated elsewhere [CLARKE 1984, DUBOIS 1988].

The discussion that follows begins with an overview of structured modeling principles that motivate the NMF. Many of these are inspired by the work of [ELMQVIST 1986] and [MATTSSON 1988]. This is followed by a description of the format, supported by small examples.

2. MODEL STRUCTURING PRINCIPLES

The NMF is based on a few principles that ensure generality:

1. Continuous models are expressed in terms of equations.

2. Variables and interconnections are typed.

3. Large models must allow hierarchical decomposition.

4. Validation is integrated into the modeling process.

The principles are briefly described and defended below.

2.1 Equation Modeling

The internal behavior of a continuous component of the NMF is described by a differential-algebraic system of equations which for the general case can be written

$$f(x, \dot{x}, p) = 0 ,$$

where f is a vector function of the variable vector x, its time derivative $\dot{x}$, and a parameter vector p. In all cases of interest here, this system of equations will be underdetermined; some of the x's will have to be given as functions of time.

Let us, for the sake of the discussion, separate between the *equation model* of a component and a *problem* for the same component, where the problem is the underdetermined equation model *together* with a selection of given variables. For example the equation model of a thermal resistance may be written

$$0 = q - UA(t1 - t2),$$

where q is the heat flow through the resistance and t1 and t2 are the terminal temperatures. Now, for this simple one-equation model three different problems, i.e. combinations of given and calculated variables may be posed:

1. t1 and t2 given and q calculated
2. t1 and q given and t2 calculated
3. t2 and q given and t1 calculated

All three problems are *well posed*. In the following, well posed will be used in the sense: able to produce a locally unique non-trivial solution. For more complex models only some selections of given variables will yield well posed problems.

Each component model in most current simulation environments, e.g. TRNSYS and HVACSIM+, is described as an equation model *along with a single input-output selection* (a problem in our sense). The component modeler makes this selection when the model type routine is written.

The pre-selection of given variables leads in some cases to limitations in the actual use of the models. Frequently a system modeler, using available types, would like to connect the inputs of one component with the inputs of another and similarly for the outputs. This, of course, is impossible and one of the component models has to be rewritten, with a different input-output selection. The system modeler is forced to become a component modeler and write, debug and compile Fortran code.

These difficulties are overcome in some of the more recently proposed environments (e.g. SPANK and Ida, formerly MODSIM [SAHLIN 1988]) by leaving the input-output designation to the environment. This will substantially increase the versatility of each component model.

The automatic input-output designation in more recent environments is done by keeping equation models separate from input-output selections until the components are actually connected together. This separation is only possible if equations are declared separately, the way they are in the NMF.

Since some environments can do without explicitly stated input-output designations in their component model format, one could argue that this information is redundant in the NMF, which should be free of environment specific non-essential information. There are however several reasons for including *one possible input-output designation* (one problem) for each NMF component model. Firstly, if this information was to be left out, automatic translation would be

impossible for input-output oriented environments. Secondly, a viable input-output set is a part of the required validation procedure. That is, a component modeler has to demonstrate at least one well posed problem for a model.

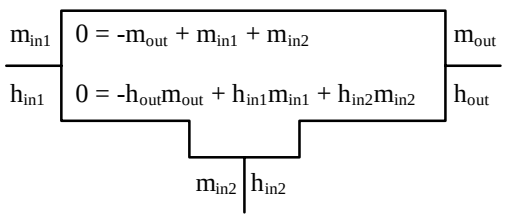
2.2 Component Interconnection

Having focused briefly on the internal behavior of component models we turn to the interconnection mechanism between them. Little attention has been devoted to this topic in many of the past discussions on the development of common component libraries, although model reuse and exchange have been the primary motivations. However, one should be aware that sets of components developed by various groups will remain to be incompatible, even when stored in a common library, unless a structured way of constructing inter-component links is imposed. Otherwise, the sockets and the prongs simply will not fit together.

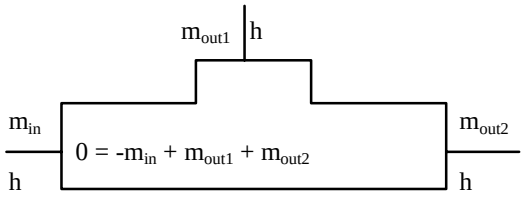
The development of a set of component models for a simulation task involves numerous decisions, some of which are crucial and others which are less fundamental in nature. Unfortunately, all of these decisions, not just the crucial ones, will later on influence the compatibility with other models. It is our aim here to provide a component format which encourages compatible choices among the trivial decisions without imposing any restrictions on the fundamental ones.

One of the initial crucial decisions to be made is the choice of a set of variables that will represent the behavior of the simulated system to an appropriate degree of accuracy. For example, in a simple HVAC circuit without cooling it might be sufficient to choose dry air mass flow rate and air enthalpy as the main variables carrying information between individual components. We are referring here to the set of variables involved in the *interaction* between components; additional variables may be used internally. Once this choice of interaction variables has been done, a *compatible family* of components can be developed. For the HVAC circuit this might involve e.g. a collector, a distributor, a heating coil and a simple zone model as shown in Figure 1.

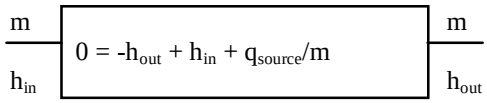
Collector



Distributor



Heating Coil



Zone

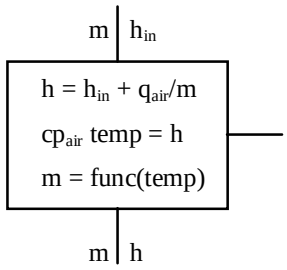


Figure 1.

The zone model has a control function, func, built in which determines the supply air flow rate as a function of zone temperature.

The choice of interaction variables is affected by the component model complexity. For example – if we wish to consider now moist air problems – a cooling coil model should include the effect of condensation, and thus some measure of air humidity must be included in the air characterization. If the cooling coil is to be used in conjunction with the previous models, the list of interaction variables to be carried around the circuit must be expanded. The principle here is that the component in need of the most information determines the interaction set of variables. Components with smaller needs will ignore unnecessary circuit variables.

Now, let us look at some of the trivial decisions for our sample case. Although the simulation in principle can be carried out using vastly different sets of units in each component, compatibility is enhanced if common units are used. This is an area where encouragement, via access to existing models, and mild punishment, via compulsion to write additional declarations, are likely to stimulate uniformity. For the sample HVAC circuit, a similar argument can be

made concerning the choice between temperature and enthalpy as an interaction variable.

A mechanism for increased compatibility in this sense is *variable typing*. All variable types to be used in component models are declared globally in the NMF. A modeler who is about to introduce a new model in the library will use already declared types whenever possible.

The next step is to declare the groups of variable types that characterize compatible families. Such a group is called a *link type* in the NMF. Mass flow rate and enthalpy together and in this order is an example of such a type. Examples of the typing syntax are located in the beginning of section 3.

The link concept also allows an environment user to connect submodels at the interface level rather than variable by variable. This means, for example, that a fan outlet is connected with a cooling coil inlet as far as the user is concerned; in the background however several variables may be involved in the connection. Most current simulation environments, e.g. TRNSYS or HVACSIM+, operate on the variable level, so the link concept would in this respect simply be ignored for these. The more important library structuring effect of link typing is still retained.

In link supporting environments link types can be used to check whether a user is making meaningful connections. There are cases, however, when a strictly imposed typing concept is too restrictive. Controllers, for example, should be allowed to interface with various types of links. This is dealt with in the NMF by providing a generic link type which can contain any number of any type of variables. An environment can then check the individual variables in the connecting links for matching types rather than the links themselves. A generic variable type is also provided in order to allow for suppressed type checking on the variable level as well. [MATTSSON 1988] discusses ways of constructing more elaborate type checking in a modeling environment.

2.3 Hierarchical Decomposition

Another fundamental concept for structured modeling is hierarchical submodel decomposition, i.e. one submodel within another in multiple levels. A composite building model could then, for example, be composed of several submodels, each one representing a floor. A floor is in turn built up of several zone models, which are built from wall models, and so on. One major advantage of this method is that it enables incremental modeling and validation. A modeler can make sure that e.g. a wall model behaves properly before it is used as part of a zone model, which then is similarly validated and so on, incrementally approaching the building level. Another advantage is that good graphical interfaces can be constructed for a corresponding hierarchical presentation of a model, where a user first gets an overall view of the system and then can zoom down for successively increased levels of detail.

Although most component models in the NMF will be used as part of composite or macro models on the environment level, the formatting of composite models themselves, i.e. interconnection templates, is not encompassed by the present proposal. There are several reasons for this, including the observation that small models are inherently more readily reused. At the same time we recognize that the process of component model development, within the NMF, could benefit from hierarchical decomposition, so eventual extension to this capability is an open issue.

2.4 Model Validation

Component model validation – in the sense of making sure that a model to some degree of accuracy reproduces the behavior of the actual physical device – can of course only be done if one has access to the device itself. There is no way to stop someone from using a library component in a non-intended way. The best thing one can do is to require extensive textual documentation to be provided along with the library entry, including the background of the underlying mathematical model. The documentation aspect of library building is beyond the scope of this paper.

The ambition of the NMF is to make sure that the entered models make sense from a mathematical perspective. Unfortunately, even this is quite a task. Existence of solutions to nonlinear equations is a very difficult subject and no general and practical theory exists. A model may work well over a particular parameter and variable range and be ill posed over another. In the end, we are left with the component modeler's ability to write robust models and to document them properly, including their ranges of validity.

What the NMF asks of a modeler is that a single problem – one input-output designation along with an equation model – is provided, and that its range of well posedness is specified. The well posedness range is specified in two different ways: firstly, in terms of explicit limits on the involved parameters and variables and, secondly, in terms of accompanying documentation. Responsibility for the existence of solutions for other possible input-output designations must be left to the targeted environments.

A number of methods of varying degrees of reliability

are available to the component modeler in order to make sure that the single problem is well posed. These methods can be applied separately or jointly. Eventually, there may be software tools to assist in this process. The methods include:

1. *Functional Linear Independence*. This means that no model equation can be formed by a linear combination of the others.

2. *Matching*. There must exist a one to one matching between equations and variables in the designated output set.

3. *Regular matrix pencil*. For a general component model (from section 2.1) call the vector of the designated input set u and the corresponding output vector x yielding f(x, $\dot{x}$, u, p), where dim(f) = dim(x). Then the matrix (pencil):

$$\lambda \frac{\partial f}{\partial \dot{x}} - \frac{\partial f}{\partial x} ,$$

   where λ is some scalar, must be non-singular for all but a finite number of λ's and this must, of course, be true for the entire parameter and variable working range of the component [SÖDERLIND 1988].

4. *Numerical Testing*. This is the most reliable test and also one of the more practical. The idea is that the modeler finds some algorithm for solving the designated problem, e.g. a direct iterative scheme or a general purpose differential-algebraic integrator such as DASSL [PETZOLD 1982] or even a simulation environment, and then, by numerical experimentation, finds the range of well posedness in parameter and variable space. As a minimum, it must be shown that a solution exists in the intended operational regime.

Some solvers take advantage of information about "undesirable inverses" of individual equations. The basic idea here is that a scalar equation, for example h(x,y) = 0, may be readily inverted to yield x = g1(y) where g1 is a well behaved function, but the inverse y = g2(x) may be problematical. One possible problem is that the function g2 may not be well behaved numerically. For example, dg2/dx may become infinite in the range of interest, or for environments that develop the inverses symbolically, g2 may not be obtainable as a closed form expression, or even if obtainable it may have poor numerical properties or be unwieldy. The list of undesirable inverses is optional in the format and can be left out for the convenience of local modeling in environments that do not use this information.

3. THE FORMAT

In this section the basic elements of continuous NMF models are explained and exemplified. Some more advanced features have been omitted due to space limitations. A formal syntax description has been formulated but is not included in this paper.

3.1 Global declarations

As previously motivated variable types and groups of such types, link types, are declared globally. The global declarations are then referenced from each component model declaration. Parameter types and constants are also declared globally within a library of component models.

Some examples of global declarations are

```
VARIABLE_TYPES

/* name          unit          kind*/

temp             "Deg-C"       CROSS
heatflux         "kW"          THRU
massflow         "kg/h"        THRU
enthalpy         "kWh/kg"      CROSS

LINK_TYPES

/* name          variable types ... */

heat_flow        (temp, heatflux)
heat_source      (heatflux)
mass_enthalpy    (massflow, enthalpy)

PARAMETER_TYPES

/* name          unit */

heatflow         "kW"
heat_capacity    "kWh/(kg Deg-C)"
massflow         "kg/h"
temp             "Deg-C"

CONSTANTS

/* name          value         unit  */
stef_bolz        5.77E-11      "kW/(m2 K)"
```

The first two fields of a variable type declaration need no explanation, but "kind" may not be familiar. All variables can be categorized as being of either direction dependent flow-type (e.g. mass flow, heat flow, electrical current, torque and force) or direction independent potential type (e.g. temperature, pressure, enthalpy, voltage and position). The physics of

flow-type variables says that they should sum to zero when two such variables are connected together. They are traditionally called through variables and will be called so here as well. Potential-type variables are on the other hand set equal to each other when connected. They are called cross variables.

A link type declaration is a named list of a set of variable types.

3.2 Continuous Model Elements

The elements of continuous models will be introduced incrementally, starting with the collector model of the sample HVAC circuit. All examples of this paper are designed primarily to illustrate the NMF. Most of them have not been tested in practical simulation.

```
CONTINUOUS_MODEL mh_collector

ABSTRACT "A thermal tee-piece model for bringing together two separate streams of fluid"

EQUATIONS

/* mass balance */
0 = -m_out + m_in1 + m_in2   BAD_INVERSES () ;

/* energy balance */
0 = -h_out*m_out + m_in1*h_in1 + m_in2*h_in2
BAD_INVERSES (h_out, h_in1, h_in2)

LINKS

/*       type          name      variables .... */
mass_enthalpy      outlet    POS_OUT m_out, h_out ;
mass_enthalpy      inlet1    POS_IN m_in1, h_in1 ;
mass_enthalpy      inlet2    POS_IN m_in2, h_in2

VARIABLES

/*type       name   role  def    min    max   descr.*/

massflow     m_out   OUT   0.      0.     BIG   "outlet massflow"
massflow     m_in1   IN    0.      0.     BIG   "inlet 1 massflow"
massflow     m_in2   IN    0.      0.     BIG   "inlet 2 massflow"
enthalpy     h_out   OUT   0.     -BIG    BIG   "outlet enthalpy"
enthalpy     h_in1   IN    0.     -BIG    BIG   "inlet 1 enthalpy"
enthalpy     h_in2   IN    0.     -BIG    BIG   "inlet 2 enthalpy"

END_MODEL
```

3.2.1 Equations

As previously motivated, the internal behavior of continuous components is described by a system of scalar equations, each of which is written

<expression> = <expression>,

where an expression may be a single variable, a first order time derivative, a parameter, a number, or some mathematical combination of the above. The aim is to keep the syntax as "natural" as possible. Expressions may also include references to separately defined functions written in a regular programming language. The order of the equations is completely arbitrary; the solution procedure is beyond the scope of the component model.

The optional list of bad inverses is associated with each equation.

Variables may be arrays, and vector operations can be defined through external subroutine calls. The syntactical details of such operations have been omitted here.

3.2.2 Links

All variables that connect the model with neighboring models must appear in a link declaration. The link type must be either globally declared or GENERIC. Each THRU variable in the link is specified in terms of its direction of definition.

3.2.3 Variables

Each continuous model variable is declared in seven aspects:

1. Type. Each type that is referred to must be either globally declared or of the GENERIC kind.

2. Identifier. For array-type variables, index ranges are given.

3. Role. As mentioned earlier, one feasible *problem* is specified for each model. Variables are cast to play a certain role in this problem as either given (IN) or as calculated (OUT).

4. Default value. Most environments will provide defaults for initial values (of state variables) and of initial value guesses (for algebraic variables).

5.& 6. Minimum and maximum limit of the allowed range.
Each variable is given a range, within which the model is valid.

7. Explanatory text string.

Variables that only appear in the links (interfaces) of a model – i.e. which do not appear in any of the equations – are declared in the same way. Role is irrelevant for these variables, but is conventionally set to be IN.

The mh_zone model involves some additional complexity:

```
CONTINUOUS_MODEL mh_simple_zone

ABSTRACT "A zone model with built in control of supply
air flow rate as an external function of zone temperature"

EQUATIONS

/* zone energy balance */
h = h_in + q_air/m      BAD_INVERSES () ;

/* temperature-enthalpy conversion */
cp_air*temp = h      BAD_INVERSES () ;

/* required supply air mass flow */
m = func (temp, m_max, m_min, t_max, t_min)
BAD_INVERSES (temp)

LINKS

/*  type            name          variables .... */

mass_enthalpy     air_inlet       POS_IN m, h_in;
mass_enthalpy     air_outlet      POS_OUT m, h;
heat_flow         zone_air        temp, POS_IN q_air;

VARIABLES

/*type      name   role  def    min    max   descr.*/

enthalpy    h_in   IN    0.    -BIG    BIG   "entering air"
enthalpy    h      OUT   0.    -BIG    BIG   "zone air"
massflow    m      OUT   .01   SMALL  BIG   "supply air"
heatflux    q_air  IN    0.    -BIG    BIG   "air heat source"
temp        temp   OUT   0.    -BIG    BIG   "air temperature"

PARAMETERS

/* type        name    def      min     max   descr*/

heat_capacity  cp_air  28E-5   0.      BIG   "air heat capacity"
massflow       m_max   50.     SMALL   BIG   "maximum
                                               ventilation rate"
massflow       m_min   .01     SMALL   BIG   "minimum
                                               ventilation rate"
temp           t_max   25.     0.      BIG   "temperature at
                                               which supply air
                                              is set to minimum"
temp           t_min   18.     0.      BIG   "temperature at
                                               which supply air
                                              is set to maximum"

FUNCTION

FLOAT   func (temp, mmax, mmin, tmax, tmin)

LANGUAGE   F77

INPUT
      FLOAT temp, mmax, mmin, tmax, tmin;

CODE

REAL FUNCTION FUNC(TEMP, MMAX, MMIN,
TMAX, TMIN)

REAL TEMP, MMAX, MMIN, TMAX, TMIN

IF (TEMP.GE.TMAX) THEN
      FUNC = MMIN
ELSEIF (TEMP.LE.TMIN)
      FUNC = MMAX
ELSE
      FUNC = (MMIN - MMAX)*TEMP/(TMAX - TMIN)
ENDIF

RETURN
END

END_CODE
END_MODEL
```

3.2.4 Parameters and Model Parameters

Parameters are used to adapt a generally formulated equation model to the behavior of an actual device. They are declared under two separate headings: Model Parameters and Parameters. The former allow a user to adapt a model structurally, to provide internal flexibility in numbers. Their purpose and use are further explained in section 3.2.6. The latter class of parameters are the more straightforward, they specify behavioral properties such as size, heat capacity, thermal conductivity etc.

3.2.5 Functions

Separate functions or subroutines are used either in the equation model or for parameter processing, which is explained below. They may be written in various well known programming languages, thus making it possible to reuse already existing code within the format context.

3.2.6 Flexible Model Descriptions

The main objective of simulation environments, rather than simulation programs, is to provide increased modeling flexibility. This flexibility can be separated in *structural* and *behavioral* flexibility. The division between the two is somewhat diffuse; structural flexibility means that mathematical models of structurally differing physical systems may be built, and behavioral implies that a structurally fixed model may be adapted to simulate quantitatively different systems of the same basic structure.

The possibility to interconnect component models in various configurations provide structural flexibility in simulation environments, while in the more traditional programs the emphasis clearly is on behavioral flexibility.

Flexibility in numbers. It is convenient to provide some degree of structural flexibility *within* a primitive component model. For example, a wall model can in principle be constructed by connecting a number of separate instances of thermal resistance and mass models in series. The potential accuracy of the model is then determined by the number of layers (masses). However, this approach is cumbersome in several ways. The addition of a layer in order to alter the accuracy is a modeling operation to be carried out in several steps, e.g. instantiate a model, set its parameters and connect it. It would be much easier if this flexibility in numbers could be contained within a single wall model. The NMF has a construct for flexibility in numbers, the FOR statement. It is illustrated by a finite difference model of a wall.

```
CONTINUOUS_MODEL   thermal_wall

ABSTRACT "A 1D finite difference wall model"

EQUATIONS

/* space discretized heat equation */
FOR i = 2, n_layers -1
        c*t'[i] = t[i - 1] - 2.*t[i] + t[i + 1] ;

c*t'[1] = taa - 2.*t[1] + t_[2] ;
c*t'[n_layers] = t[n_layers - 1] - 2.*t[n_layers] + tbb ;

/* boundary conditions */
0 = -ta + .5*(taa + t[1]) ;
0 = -tb + .5*(t[n_layers] + tbb) ;
0 = -qa + d*(taa - t[1]) ;
0 = -qb + d*(tbb - t[n_layers])

LINKS

/*  type        name        variables .... */

heat_flow       a_side      ta, POS_IN qa;
heat_flow       b_side      tb, POS_IN qb;

VARIABLES

/*type     name            role  def   min    max    description*/

temp       t[1..n_layers]  OUT   0.   -BIG    BIG   "temperature
                                                      profile"
temp       ta              OUT   0.   -BIG    BIG   "a-side surface
                                                      temp"
temp       tb              OUT   0.   -BIG    BIG   "b-side surface
                                                      temp"
temp       taa             OUT   0.   -BIG    BIG   "a-side virtual
                                                      temp"
temp       tbb             OUT   0.   -BIG    BIG   "b-side virtual
                                                      temp"
heatflux   qa              IN    0.   -BIG    BIG   "a-side
                                                      entering heat"
heatflux   qb              IN    0.   -BIG    BIG   "b-side
                                                      entering heat"

MODEL_PARAMETERS

/*type     name        min      max        decription */

INT        n_layers    3        BIGINT     "number of layers"

PARAMETERS

/*type          name    def    min      max    description */

c-type          c       10.    SMALL    BIG    "rho*cp*dx*dx/
                                                  (lambda*3600.)"
d-type          d       1.     SMALL    BIG    "lambda*a/dx"

/* easy access parameters */

area            a       5.     SMALL    BIG    "wall area"
length          thick   .25    SMALL    BIG    "wall total
                                                  thickness"
heat_trans      lambda  .83    SMALL    BIG    "heat transfer
                                                  coeff"
density         rho     2050.  SMALL    BIG    "wall density"
heat_capacity   cp      1.2    SMALL    BIG    "wall heat
                                                  capacity"

PARAMETER_PROCESSING

wall_par( n_layers, c, d, a, thick, lambda, rho, cp)

FUNCTION

VOID wall_par (n, ccoeff, dkoeff, area, thick, lambda, rho, cp)

LANGUAGE F77

INPUT
    INT n;
    FLOAT area, thick, lambda, rho, cp;

OUTPUT
    FLOAT ccoeff, dcoeff;

CODE
/* The Fortran code is omitted*/
END_CODE
END_MODEL
```

A further example of the FOR statement comes from the mh_zone model. The version we have discussed so far has a single qT-link enabling it to be connected to one external component like a wall or a thermal resistance. It would be better to have a parameter within the model, a model parameter, which determines the number of available qT-links. Then a suitable number can be selected during system modeling and the zone core can be connected to an arbitrary number of walls.

Before we go into the feature provided for run time behavioral flexibility, the parameter processing header of the wall model deserves to be mentioned.

3.2.7 Parameter Processing

All the various named coefficients of the equation model are declared as parameters, but in addition to this, extra parameters may be declared in a model. Frequently, the mathematical characterization of a model, i.e. the parameters that appear in the equation model are quite different from those that an engineer spontaneously would choose to specify the corresponding physical device. For example, a zone model, which accounts for long wave radiation between surfaces, would have view factors (in some form) appearing in the equation model. However, a user of such a model would normally not prefer to specify these directly but rather the sizes, orientation, and reflectance of the surfaces themselves.

The mapping of user given parameters or, more informally, *easy access parameters* onto equation model parameters is done by one or more subroutines. The reference to these routines is declared under the heading Parameter Processing.

3.2.6 Flexible Model Descriptions – revisited

Behavioral Flexibility. Increased behavioral flexibility is provided in the new environments by the possibility of locally replacing component or subsystem models without disturbing the model as a whole. This feature is most frequently used to experimentally find an appropriate level of approximation for a particular submodel. Behavioral flexibility is often needed at run time as well. The internal description of a model may need to change significantly as certain conditions are fulfilled. A typical example is when a component goes into a saturated state, e.g., when a heating coil or a fan has reached its capacity limit, or when a model has a singularity, which can be circumvented by a local re-description.

Changes like these can of course be hidden in external functions, like we did with the ventilation air demand equation of the mh_zone model. It would however make the model more legible if they were declared explicitly. The construct provided for this in the NMF is the *conditional expression* with a structure familiar to every programmer, for example:

```
/* required supply air mass flow rate in the mh_simple_zone */

m = IF temp >= tmax THEN
     mmin
ELSE IF temp <= tmin THEN
     mmax
ELSE  (mmin - mmax)*temp/(tmax - tmin)
END IF
```

3.3 Algorithmic Models

Although continuous elements form the bulk of a building simulation model, algorithmic models operating in discrete time are more suitable for certain components. Sampling, micro processor based controllers are obvious physical examples of such components. A further activity in a simulation that lends itself to algorithmic description is boundary data processing. To calculate, for example, the amount of solar radiation that falls on a certain surface at a particular time of day is more straightforward in algorithmic form. The inherent input-output orientation of an algorithmic description is in these cases only a minor restriction, since one rarely is interested in the reverse question: What input gives rise to this output?

For these reasons, algorithmic components are next on the list of items to be formatted or standardized. Due to space constraints we omit any further discussion about this here.

4. SUPPORTING SOFTWARE

Part of the NMF concept is also a set of supporting software tools. Among these are translation tools for conversion of the models from the standard format to that of particular modeling environments. Also needed are tools for the creation and maintenance of component models, and those for the creation and maintenance of libraries of such models. It is envisioned that some organization will host a "base" library. This base library and the software tools can be ported to any modeling environment. It is anticipated that once ported, the base library will be augmented as required by the needs of the user.

The library itself contains models in the standard format. The translation software should be partitioned into the portion that will be the same regardless of the target environment, called the interpreter, and a portion that generates the models in the format of a particular environment, called the generator. The interpreter knows the standard syntax and library format, while the generator knows the syntax and format of the target environment. Obviously, the interpreter can be delivered with the library, while the generator will have to be customized for each environment. This is not unlike the task of porting systems software to different hardware environments. It can be made relatively straightforward by providing implementers with example generators for well-known environments, and certain software tools that are commonly needed.

5. CONCLUSIONS AND DISCUSSION

The NMF has been proposed as an alternative to crafting component models for each of the existing and evolving simulation environments. The NMF has the following precepts: (1) Essential information about the component models is formalized in order to allow automatic translation; (2) Continuous models are equation based; (3) The concept of typed links encourages libraries of plug-compatible component models; (4) Mathematical validation is required for each model. We have shown some of the details of the NMF specification in a few examples. However, it must be understood that as further component models are developed in the format, changes and additions of this preliminary proposal may be required.

A key part of the concept is automatic translation. Feasibility of translation to a specific environment has been demonstrated for SPANK using MACSYMA [BUHL 1989]. Furthermore, generation of Ida models is straightforward, since they are also equation based. Type routines for TRNSYS and HVACSIM+ involve an additional difficulty because

algebraic equations are solved locally within each subroutine. A general purpose non-linear algebraic equation solver will have to be part of the generators for these environments. There are however several robust solvers readily available in e.g. the SLATEC library.

Given a good generator, the format should provide a TRNSYS component modeler with a time effective alternative to direct Fortran programming.

Perhaps the most significant potential outcome of the NMF would be the inception and growth of a significant public domain library of building component models. For this to occur, six events are required:

1. The acceptance of a standard format, perhaps based on the one outlined herein.

2. The development of the software that interprets the standard format.

3. The development of a model generator for each of several widely used environments, e.g., TRNSYS, and HVACSIM+.

4. The development of software tools for the creation and maintenance of a library.

5. The creation and validation of an initial base library containing frequently used component models, such as those used for energy analysis. The TRNSYS library could be a starting point.

6. Establishment of a mechanism for acceptance of new models for the base library, as well as formation of new specialized libraries.

Once these events have taken place, there should be be a strong incentive to use and extend the library, both because of the economy relative to independent library development by the user communities of each environment, and because of the desire of users to employ accepted component models. Once the library comes into wide use, forces will develop to extend it into new areas, such as control simulation.

ACKNOWLEDGEMENTS

The basic ideas behind the NMF came forth during a series of discussions at the Swedish Institute of Applied Mathematics. The authors have merely put it all on paper. Magnus Lindgren, Axel Bring, Lars Eriksson and Gustaf Söderlind have been participating in these discussions in addition to the authors.

REFERENCES

BUHL 1989 W.F. Buhl, E.F. Sowell, J.M. Nataf "Object Oriented Programming, Equation Based Submodels, and System Reduction in SPANK," Building Simulation '89 Conference, Vancouver

CLARKE 1984 J.A. Clarke, L. Laret "Explanation of the Data Processor Proforma," ABACUS, Strathclyde, and Laboratoire de Physique du Batiment, Liege, working document, Dec., 1984

CLARKE 1985 J.A. Clarke, J.J. Hirsch, W.F. Buhl, A.E.Erdem, F.C.Winkelmann, E.F.Sowell, A. Lahellec, N. Huang, J. Sornay, L. Laret "A Proposal to Develop a Kernel System for the Next Generation of Building Energy Simulation Software." Lawrence Berkeley Laboratory, Nov. 1985

CLARKE 1986 J.A. Clarke "The Energy Kernel System: A Technical Overview," Proceedings of the Second International Conference on System Simulation in Buildings. Liege, Dec., 1986

DUBOIS 1988 A.M. Dubois "MODEL-BASED COMPUTER AIDED MODELLING: the new perspectives for building energy simulation," communication from CSTB, B.P. 21, 06561 VALBONNE Cedex, France

ELMQVIST 1986 H. Elmqvist "LICS: Language for Implementation of Control Systems," Dept. of Automatic Control, Lund Institute of technology, Box 118, 221 00 Lund, Sweden

MATTSSON 1988 S.E. Mattsson "On Model Structuring Concepts," Presented at 4th IFAC Symposium on Computer Aided Design in Control Systems (CADCS), Beijing, China

PETZOLD 1982 L.R. Petzold "A Description of DASSL: A Differential/Algebraic System Solver," Proceedings of IMACS World Congress, Montreal, Canada, 1982

SAHLIN 1988 P. Sahlin. "MODSIM: A Program for Dynamical Modelling and Simulation of Continuous Systems." Proceedings of the 30th annual meeting of the Scandinavian Simulation Society, ISSN 0357-9387

SOWELL 1986 E.F. Sowell, W.F.Buhl, A. E. Erdem, and F.C. Winkelmann. "A Prototype Object-based System for HVAC Simulation." Proceedings of the Second International Conference on System Simulation in Buildings. Liege, Dec., 1986

SÖDERLIND 1988 G. Söderlind, L.O. Eriksson, A.

Bring "Numerical Methods for the Simulation of Modular Dynamical Systems," Report from the Swedish Institute of Applied Mathematics

IDA SOLVER

A Tool For Building and Energy Systems Simulation

Per Sahlin
Swedish Institute of Applied Mathematics
Chalmers Teknikpark
412 88 Gothenburg
Sweden

Axel Bring
Building Engineering Services
Royal Institute of Technology
100 44 Stockholm
Sweden

ABSTRACT

General continuous simulation of today is a handicraft mastered by a small group of experts. Systematic modelling techniques and supporting tools are beginning to emerge, promising access to advanced simulation also for less experienced users. Several ambitious projects around the world are at different stages of completion (e.g. EKS, SPANK, CLIM2000, SEE, MS1). These projects approach the task from widely different angles and the final products, once available, will offer a rich menu of alternatives. To a certain extent the duplication of effort is desirable, since the state of the art is still in its early stages and several different paths of development deserve to be explored.

However, there are some minimum requirements that a truly effective simulation environment must fulfill. As of today, shortcomings in existing solvers still force most users down to the smallest of detail and many of the advantages with high level modelling are thereby lost.

In this paper some of these shortcomings and their remedies are discussed, leading to a list of corner stones for a simulation environment solver specification.

- Modelling is input/output free, i.e. variables have no irrevocable roles as given or calculated. Input/output free modelling leads to models described by equations rather than the traditional calculation procedures, thus getting closer to the physical relationships known to the modeller.
- The system can handle algebraic as well as differential equations, including algebraic loops.
- The integration uses variable timestep in order to supply consistent, easy to use, accuracy control.
- Sparseness in the system of equations is utilized effectively.
- Models can be precompiled and used as ready building blocks, as in TRNSYS or HVACSIM+.
- Discontinuities in driving functions and in model equations can be handled properly.
- Extensions to the basic equation modelling allow handling of discrete system states, as required by e.g. hysteresis.

These requirements are analyzed and IDA Solver is presented.

IDA Solver is the nucleus of IDA, an environment for interactive graphical modelling and simulation, which is under development at the Swedish Institute of Applied Mathematics in cooperation with the Department of Building Engineering Services, KTH, Stockholm.

1. INTRODUCTION

Powerful inexpensive desktop computers in combination with new graphical and object oriented software techniques have brought salient possibilities to the field of continuous simulation. In response, several promising development projects have been started. Some of these, e.g. SEE (Mattson 1989) and MS1 (Lorenz 1990, personal communication), are application independent, while others are to some degree committed to a certain application. In the field of building and energy simulation some of the more well known projects are SPANK (Buhl, Sowell, and Nataf 1989), EKS (Clarke et al 1989), CLIM 2000 (Bonneau et al 1989), and IDA (Bring 1990). The common goal is to create simulation environments, i.e. sets of software tools which make it possible to build simulation models for virtually any aspect of a physical system. The result of these efforts will be a broadening of the traditional boundaries of building simulation. Focus will no longer always be on the envelope, but it will be where it needs to be, envelope, systems, plant, fire, etc.

With the exception of SPANK and IDA, the projects have exclusively been addressing modelling aspects, i.e. the tools for the interactive process of

constructing a simulation model. This is a previously neglected issue of great importance. However, new modelling capabilities will also put new requirements on the solvers of tomorrow. Modelling papers sometimes create the impression that the question of actually solving arbitrary models already has been dealt with to a sufficient degree. This is, as will soon become apparent, far from the truth.

The IDA simulation environment is, since four years, under development at the Swedish Institute of Applied Mathematics in cooperation with the department of Building Services Engineering at the Royal Institute of Technology, Stockholm. The work is pursued along three paths:

1. A solver for large scale differential-algebraic systems of equations has been developed. It has been tested internally for about two years and is now available for external testing.

2. A component model format, Neutral Model Format (NMF), has been developed in cooperation with the SPANK team at Lawrence Berkeley Lab (Sahlin and Sowell 1989). NMF allows component models to be described in equation form in a program neutral way, thus enabling the same model to be used in several environments. The format is available for use and automatic translators are under development for SPANK and IDA. A paper with some examples of NMF models is presented at this conference by Kjell Kolsaker of NTH, Norway (Kolsaker 1991).

3. An interactive graphical modelling tool, IDA Modeller, has been implemented and is currently tested internally as a front end for the solver. Some features are:

   – hierarchical (nested) models in multiple levels,

   – link level connections with compatibility checks, i.e. variables are not connected individually,

   – model presentations can be customized to a high degree,

   – vector valued variables, parameters and links.

The overall ambition with the IDA project is to provide an integrated environment which makes simulation accessible to engineers with limited modelling experience. This makes it necessary to lift the level of abstraction up from the equation (or statement) level to the component level. A user should be able to view, give parameters to and connect a component model instance much in the same way the physical device is treated during design. This not only puts strict requirements on the user friendliness and flexibility of the modelling software, but perhaps even more so on the solver, which must be extremely robust without significant efficiency sacrifices. In this paper we will concentrate on the solver related issues. Aspects on modelling will be treated separately.

The text is organized in three parts. The main, first part is a point by point discussion of solver design. The choices made in IDA are mentioned but not dwelled upon. Most of the issues are worthy of a separate paper, and some of a book, so the account will be far from exhaustive. It should be viewed more as our version of a checklist for someone in search of a good solver. The second part provides a brief overview of IDA in general and the solver in particular. Finally, we will go into some detail on a much neglected solver issue of significant practical importance: the treatment of discontinuities and discrete system states.

2. CONTINUOUS SIMULATION SOLVERS

2.1 Equation Modelling

There are several ways of formally representing a simulation model, e.g. bond graphs, block diagrams with Laplace transfer functions, or block diagrams with mathematical equations. The relative merits of each method in terms of generality, sub model reusability, availability of analysis tools, and user presentability is an interesting and important topic – in particular for the field of building simulation, where the lack of formal coherence of modelling methods seriously hampers development. We have done some work relating to this (Sahlin 1988, Sahlin 1989) and Elmquist and Mattsson have made outstanding contributions by, among other things, bringing forward equation based model structuring principles and corresponding software (Elmquist 1986, Mattson 1988). Lorenz has done interesting work on bond graphs and multi representation modelling software (Lorenz 1989). A thorough review of this work is beyond the scope of this paper and, in terms of modelling we will limit our discussion to a few key questions related to solvers.

The model representation methods mentioned above can to a sufficient extent be translated into each other and the fact that we in this paper and in IDA use equations does not affect generality. There is however another class of frequently used representation methods that yields models that are of a different nature and not lend themselves to cross translation without resort to artificial (or even real!) intelligence. Building simulation style transfer functions in dis-

crete time is one example of such a model type. Subroutine packaged calculation sequences is another frequently used component model form, which in our opinion has serious shortcomings as a primary model representation. These methods mix the modelling and simulation activities and are therefore unsuitable as a base for large scale model libraries. The main reason for this is that they are limited to input/output modelling.

2.2 Input/Output Modelling

Several authors have stressed the limitations of models with fixed information flow. All modelling papers cited here point this out. Lorenz, for example, has even written a separate "Memorandum to Stop Confusion Over Subroutines and Submodels" (Lorenz 1990, personal communication). We will restrict ourselves here to briefly pointing out the problem and some of its background.

Given a mathematical model of a physical device, some of the variables can be calculated (as many as there are equations) as soon as the remaining variables in the model are given, sometimes as functions of time. For most models this selection of given and calculated variables can be done in more than one way. The calculation procedure is different depending on the variable selection. The problem is that, if the calculation procedure in itself is used to document the model, only one of all the possible variable selections can be used and there is no automatic way to change this selection. Consequently, a calculation procedure must be given for each of the interesting selections and it is obviously undesirable to have multiple representations of the same model in a library. That much about the problem. We think that most people recognize the disadvantage and the question is why most programs in fact use input/output modelling. TRNSYS and HVACSIM+ are only a couple in a multitude of examples.

One reason is certainly ease of implementation. The thought of packaging a model in a ready-to-use subroutine, which can be treated as a black box by a user, is certainly appealing and it works fine – as long as we want exactly the information the subroutine is supplying. The problem is that when the writer's opinion of what is interesting differs from the user's, one is forced into the reverse engineering problem of trying to extract an equation model from a calculation procedure. The alternative is to treat the equation model itself, or an equivalent representation, as a source and each time a calculation is needed generate the procedure automatically from the source combined with the user specified input/output selection. This is obviously more complicated from a programming point of view but the difficulties are well within reach of today's methods.

It should be pointed out that while input/output modelling is an obvious handicap for some applications it is less cumbersome for others. Some purely man-made physical systems are in themselves input/-output oriented, i.e. each component is designed to only influence downstream and not affect upstream components in any significant manner. Electronic units are, for example, often designed in this way.

For other applications it is possible to invent rules and conventions for how a user can connect models together in order to circumvent the input/output limitation effectively. Input/output thinking is rooted into our engineering and programming souls to such a degree that these connection restrictions sometimes are not even consciously noticed.

However, for most applications things become simpler when the input/output restrictions are lifted and a user is allowed to focus on real problems. Any potential driven flow system will illustrate this. An input/output model of a simple (electrical, thermal or fluid flow) resistance model must come in two versions. One which calculates the flow, given terminal potentials and another which calculates a potential given the flow and the remaining potential. For multiport components the number of needed model versions increases rapidly with the number of ports.

Input/output modelling is one of the factors that complicate simulation work by forcing users to deal with unnecessary and unstimulating detail. There seems to be little reason to accept it in a good simulation environment.

2.3 Differential-Algebraic Equation Models

The CSSL language standard of 1967 has heavily influenced, and in later years limited, the field of continuous simulation. CSSL-based solvers are still in majority. In 1967 digital solvers for stiff systems of ordinary differential equations (ODE's) were at the forefront of technology and algorithms that could solve implicit sets of algebraic equations (AE's) simultaneously were not generally available, which they are today. Consequently, CSSL-solvers still have weak capabilities for algebraic equations and we have a situation similar to that of input/output modelling, i.e. user communities that have learned to live within the limitations of a dated technology.

It has been claimed that engineers naturally use both differential and algebraic equations (DAE's) for modelling if not restricted by, e.g., a particular solver. This should certainly be true for the field of building and energy simulation, where the slow but influential

dynamics of weather and building envelope make differential equations indispensable. Implicit algebraic equations to be solved simultaneously usually come from pipe networks which distribute air and water throughout the building. These networks involve a set of nodes with different pressures and connections with nonlinear flow-pressure characteristics, due to turbulence or nonlinear fan or pump curves. If the network is closed, the model will contain algebraic loops. The resulting set of nonlinear algebraic equations can be quite difficult to solve, especially if the coefficients of the flow-pressure relations differ by several orders of magnitude. This is the case for air distribution models of buildings where nodes may be in contact via anything from a crack to an open doorway.

In traditional building simulation the numerical problem of algebraic loops is usually handled by avoiding it. The user is required to prescribe all flow rates in the system. This may be acceptable in many cases since design flow rates are supposed to be constant in time, but other times the focus of interest is on the unpredicted or undesirable deviations from the design flow rates. In conclusion, for a general simulation environment we think the ambition must definitely be set higher, since both building and system are to be simulated together with true coupling and without need for special tricks on the user's part.

Another common way to get around the problem of algebraic loops is to introduce auxiliary dynamics in the system and thereby effectively create a system of ODE's. The approach can sometimes be justified on physical grounds but it is hardly acceptable as a general solution to the problem.

For a true DAE solver the problem of detecting and "fixing" algebraic loops does not exist. The solver is designed to automatically solve a fully implicit set of DAE's. A general DAE solver, like IDA, accepts problems of the form

$$0 = F(x', x, u, p, t) , \qquad (1)$$

where x is a set of variables to be solved for, u is an external input vector and p is a set of parameters. Note that the time- derivatives are allowed to appear implicitly in the equations.

An interesting observation is that several authors from different application backgrounds have come to the same conclusion regarding this general model formulation, e.g. (Mattson 1986). In chemical process engineering the starting point was in completely algebraic models and the need to study dynamics has emerged over time, while in other fields the development has gone from ODE to DAE.

The numerically most difficult part of solving a set of DAE's is to start things up and find the first solution which satisfies the algebraic part of the system, given only a very rough initial guess supplied by the user. The phase of finding this starting point is referred to as the initial value calculation and we will return to it later.

Another important, still unsolved, problem with DAE solvers comes from systems of equations with so called high index of nilpotency. Since this is not a real separator between different solvers (no entirely satisfactory solver exists), we will not go into it in any depth. High index systems can, unfortunately, appear from seemingly innocent physical problems, like trying to calculate what the forces are acting on a particle travelling about on a given curve in space. An interesting discussion about similar problems and DAE's in general can be found in (Mattson 1986).

2.4 Algebraic Solution Techniques

For most systems of algebraic equations, the only reasonably safe and efficient way of solving the system is to use a, sometimes damped, Newton-type technique, where the system Jacobian (the matrix of the derivative of every equation with respect to every variable) is calculated and factorized. This is an undisputed fact among numerical analysts. In spite of this, several well known simulation tools use simpler fixed point iteration techniques. The main reason for this is most likely, again, ease of implementation. The result is often poor and unreliable convergence properties at the expense of the users and, as a secondary effect, badwill for simulation in general.

Algebraic equations are solved both during the initial value calculation, and during normal timesteps. In IDA the solution methods for these two cases are separated since the quality of initial guesses are vastly different.

At a regular timestep a prediction from the previous timesteps provides a high quality guess and an undamped Newton step will give optimal convergence. The difficulty lies in the choice of method for calculating the Jacobian matrix. This is a costly operation and a new Jacobian is in most solvers only calculated and factorized when the convergence slows down. Straightforward numerical Jacobian calculation involves in the worst case n+1 evaluations of all equations in the system, where n is the Jacobian dimension. Several more or less sophisticated techniques are used by different solvers for Jacobian approximation. Due to space constraints we will have to leave the issue with this observation. In IDA analytical equation derivatives are given by the user

whenever possible. They are cheap and have optimal quality. For component models where analytical Jacobians are impossible, or during component model development, numerical Jacobians are calculated. For linear component models Jacobians are calculated only once during a simulation.

Robustness of solution methods for initial value calculation is an issue of crucial importance, since this is by far the most common point of breakdown for simulations. Initial values are not only calculated initially but also at each jump discontinuity. Since nonlinear systems of equations can have multiple solutions, it can never be guaranteed that an algorithm will find the right one. This is a case where we never will be able to provide completely satisfactory tools. Two things can be done, however. First, alternative solution algorithms which converge slower (close to the solution) but safer can be used. IDA presently uses a Newton homotopy method, embedding the algebraic equations into a sequence of problems with a continuously changing solution. Other methods are planned to complement this approach. Secondly, one can violate the golden guideline never to mix model with solution procedure. A linearized version of difficult nonlinear models can be provided. These models would be used at the first iteration of an initial value calculation to set the solution off in the right direction. In addition to this, some particularly error prone equation formulations can be avoided in components.

2.5 Integration Methods

Variations in integration methods mainly affect the efficiency of the solver. However, the performance differences may be dramatic. The crucial divider runs between implicit and explicit integration schemes. Explicit schemes can outperform implicit ones by very large factors, hundreds or more, on problems with no algebraic equation and similar timescales throughout the system (non-stiff problems). On the other hand, implicit schemes may easily win by similar factors on stiff problems. If the problem is formulated as in eqn. (1), explicit methods are ruled out, since derivatives appear implicitly and advantage of explicitness is thereby lost. Consequently, IDA uses only implicit methods. However, most DAE problem can also be formulated with some loss of generality,

$$\begin{aligned} 0 &= g(x, z, u, p, t) \\ x' &= f(x, z, u, p, t) \end{aligned} \qquad (2)$$

where x is the vector of dynamic states, z holds the algebraic variables, and u and p are inputs and parameters. For this formulation explicit methods may be used for the dynamic part of the problem. The loss of generality should for practical purposes not be too serious, since derivatives only rarely appear implicitly in equation models. The gains of being able to use an explicit method can, on the other hand, not be expected to be overwhelming, since the algebraic set of equations still must be solved implicitly at each (short) timestep.

In conclusion, IDA's commitment to the general formulation of the problem and to implicit methods might in some contexts be a shortcoming. Non-stiff problems with few algebraic equations are on the other hand extremely rare in building and energy systems.

2.6 Variable Timesteps and Error Estimation

With implicit integration methods each timestep is expensive, but in return one can take long steps when there is little activity in the simulated system since only accuracy and not stability influences the allowable step length. To take advantage of this, modern solvers have step length control algorithms that monitor the activity of the solution and keep the step length optimal with respect to the desired accuracy. Step length control algorithms are usually quite crude and some interesting work is presently done to improve this by applying traditional PI-control methods (Lundh, Gustafsson, and Söderlind 1988). IDA currently uses traditional step length control but the new methods will be implemented when available.

With long timesteps comes also the risk of grossly overshooting important events in the solution such as a sudden fan start up. This necessitates a special system for signaling such incidents to the timestep control algorithm so that a step can be taken to a point just before the discontinuity, which is then passed with special care. IDA's methods to overcome discontinuities are presented in section 4.

For fixed timestep solvers the error fluctuates in an uncontrolled fashion with the activity of the solution. The simulationist must then in order to get some control over the error repeat runs with systematically smaller steps until subjectively "equal" results are obtained. Unfortunately, few people seem to have the patience to do this. The widespread any-solution-is-a-good-solution attitude is another factor which undermines confidence in simulation methods in general.

Good control over numerical errors is in our opinion absolutely crucial to serious simulation, since one must be able to separate numerical and modelling errors. The only practical way to understand modelling errors is to make repeated experiments with

different (sub) models. If another set of experiments must be carried out for each tested model in order to estimate numerical errors, the full procedure is likely to bore even the most determined user into sloppiness.

A solver should have as few error control parameters as possible, preferably only one. IDA today works with one error tolerance for integration error and an additional parameter for event localization accuracy, but the latter will most likely be eliminated in future versions. Integration error is measured relative for large quantities and absolute for small, with automatic shifts between the two. The compromise between user friendliness and good error control is difficult and alterations of the present method will probably be tried.

2.7 Utilizing Sparseness

The heavy part of the work in each timestep in an implicit algorithm is the Jacobian factorization. For a typical simulation problem the number of non-zero elements of the Jacobian is proportional to the number of problem variables, while the total number of elements, of course, is n_2. Thus, an algorithm's ability to utilize this sparseness is crucial. A thorough examination of all the different ways of doing this would fill several books and many such books exist (e.g. Duff, Erisman, and Reid 1986). Here, we will be content with a sketchy discussion of the various approaches pertinent to simulation and their relative merits with respect to a future "typical" large building simulation problem.

Integration algorithms for simulation can be classified as being either equation based or modular. Equation based algorithms operate on individual equations, while in a modular system, like TRNSYS, HVACSIM+ or IDA, the group of equations comprising a component model plays a similar role.

2.7.1 Equation Based Algorithms

Equation based algorithms, such as SPANK (Buhl, Sowell, and Nataf 1986) or NEPTUNIX (Nakhle 1986) analyze the system graph, or equivalently the incidence matrix, and based on this information equations are sorted, in order to reduce as much as possible the work of factoring the Jacobian. The most basic methods reorder equations in order to minimize the bandwidth of the Jacobian. This can be done with a number of standard methods, e.g. the reverse Cuthill-McKee algorithm. Then a factorization algorithm is applied which only operates on elements within a certain distance from the diagonal. For a sequential system where all the models come in a row one after the other this method is very effective, while, if there is feedback in the system, the band width grows large.

A more sophisticated class of equation based methods, e.g. SPANK, reduce the size of the Jacobian by automatically ordering equations in chains which may be evaluated in sequence leaving only so called cut sets or tear variables in the Jacobian (Kron 1963). This method can for many problems reduce the size of the Jacobian drastically.

This approach is carried one step further in so called symbolic reduction methods where symbolic algebra is used to actually eliminate unnecessary and uninteresting variables and equations. This can result in a very small and dense Jacobian, but sometimes unfortunately also in extremely long and complex expressions in each equation.

Some important common characteristics of equation based systems are:

- Input/output free modelling is comparatively easy to achieve.
- The same reduction principle is applied to the entire set of equations, i.e. the possibility to treat individual subgroups of equations with a different method is usually lost.
- A compiler must usually be invoked in the modelling-simulation cycle, since the entire set of equations is frequently evaluated with one single subroutine call. This is not the case with SPANK where each inverse of each different equation is compiled into a separate C-function. This way, only new component models involve compilation, while different configurations of the same components may be simulated without compilation.
- Minor structural model alterations between simulations, e.g. changing one component model for another, lead to a complete re-reduction, and frequently compilation, of the full set of equations. This will slow down the modelling-simulation cycle considerably.

2.7.2 Modular Algorithms

In modular systems the set of equations for each physical component is treated as a group. The same sorting strategies as for equation based systems are used, but on the module rather than on the equation level, thus resulting in blocked matrix structures. A module may contain internal structure which is invisible to the global solution algorithm. Frequently, only a minority of a component's variables are connected externally.

The degree of component individuality varies between different solvers and algorithm modes. At one extreme, module routines merely supply the global integrator with Jacobian elements. IDA has an option for this approach rendering maximum robustness. At the opposite end of the spectrum, components have their own integrators which proceed with a local timestep according to the local solution activity. The coupling between modules is maintained at a 'global timestep'. This class of methods is called multirate and it is available in e.g. HVACSIM+, where groups of modules which are loosely coupled, so called superblocks, proceed at different timesteps. IDA also has a multirate option, where each individual component is integrated with a separate timestep but, so far, with a common integrator. However, the knowledge about error estimation for multirate integration is still very incomplete and therefore we do not recommend the method for serious simulation.

Some principal common characteristics of modular systems are:

- Most modular environments are strictly input/-output oriented due to component models implemented as calculation procedures. Furthermore, for many systems the module procedure code is the only formal model description. IDA is, to our knowledge, the only modular environment which is input/output free. Component models for IDA are described and documented in the Neutral Model Format, NMF (Sahlin and Sowell 1989).

- Special structure within component models, or other properties such as linearity, can be utilized for individual models. For large local finite difference models this is indispensable, since they may contain a substantial number of internal states and have a very specific structure.

- Component models are compiled separately and general system modelling with library components does not involve compilation. This makes it possible to ship systems with fixed component libraries for a certain application without need for an integrated compiler.

- Changing the set of equations for an individual component is done locally within the module. The equation shift can be done either manually between simulations or automatically during simulation.

2.7.3 Trends for the Future

Simulation models of today tend to become as large as available computer processing capacity allows. This makes efficiency the principal solver property. However, as number crunching power rapidly becomes cheaper and more accessible, man time spent on modelling can be expected to play an increasing role as a limiting factor. This will make solver robustness more important than ever, since this is the critical solver property in the modelling phase. The ideal is to have access to a range of solution strategies within the same environment, where different compromises between robustness and efficiency have been made.

Automatic modelling tools will make it possible to keep track of very large models. Some people also envision automatic model generation from CAD-representations of buildings. The battle of efficiency (with no consideration of other properties) between equation based and modular methods will depend on the amount of internal structure of tomorrow's component models. Equation based methods are likely to have the advantage if there will be a lot of rather simple individual models. For modular methods there is always a certain overhead for each module and a component model needs at least a few internal states to make up for this overhead.

We think that engineers will use growing computer power mainly to develop more realistic component models. Examples could be models where things like air or water stratification are modelled with finite differences (or elements) in two or even three dimensions. Large FD models of pipes, heat exchangers, and coils are even closer at hand. The present separation, between field programs – like PHOENICS, FIDAP, and NASTRAN – and the network oriented programs under discussion here, is likely to become more diffuse. It will be the latter class of programs which adopts properties of the first, rather than the other way around. In this engineering oriented scenario the total number of components of a typical system will not exceed what a human mind can fathom with the aid of model presentation tools.

In the alternative, artificial intelligence scenario huge system models are generated more or less automatically from the CAD-representation (product model) of a building. It can be safely assumed that the number of automatically produced submodels indeed will be large, since the engineering judgment required to make wise model reductions is far beyond AI methods of today. Thermal room models can be reduced in size already by identification of central modes (Cools, Gicquel, and Neirac 1988), but climatization and control systems are, and will continue to be, far more difficult to deal with. The large number of submodels puts limits on the complexity of each one and, therefore, we have a case for equation based methods.

The art of simulation environment design is still in its early stages and it is, at this point, important that several projects are allowed to coexist. This will provide a rich material for future evaluation and standardization, and hopefully, the fittest (and not the richest or loudest) will survive.

3. AN OVERVIEW OF IDA SOLVER

In the remainder of the paper we will concentrate more specifically on IDA. Space will unfortunately not allow anything but a very brief account of everything except discontinuities which will be discussed in some detail. The IDA User's Guide (Bring 90) will complete the picture for those who would like to try the program. The numerical methods are more carefully presented in (Söderlind, Eriksson, and Bring 1988).

3.1 Overall System Structure

The IDA simulation environment is organized as two major parts:

- A modelling tool for interactive handling of system descriptions. A graphical user interface lets the user compose and manipulate component based systems, where components are represented by icons in a Macintosh style. The system representation is fully object oriented and hierarchical. Windowing technique allows access to the details of the component descriptions, including parameter values, model equations, etc. The modeller is implemented in Lisp on Apollo workstations.

- A Fortran solver for integration of systems of differential-algebraic equations. This part can be activated from the modeller to integrate systems created there, but it can also be run as a stand-alone program, taking input from a data file. The data file can be produced by the modeller or created and maintained by an editor. The format is key word oriented and uses names for all entities, thus giving good legibility. The Fortran system is easily portable and can be run on PCs.

3.2 Neutral Model Format

The models are formulated in NMF. Some additions have been made to the original format specification. These extensions will be touched upon below. An updated description of the NMF is being prepared.

Model descriptions in the NMF format can be automatically checked and translated to the internal representation required by the modeller. Preparation of screen representation of models is not automated.

3.3 Component Representation in the Solver

In the solver each component model is represented by a group of Fortran subroutines. A translator from NMF is planned and partly implemented, but currently the routines are coded separately. This manual task is made simpler and safer by a set of naming and implementation rules, but debugging of component routines is still a necessary part of model development.

The Fortran routines deal with the following four tasks:

- Evaluate the model, i.e. calculate residuals in the equations.

- Calculate the Jacobians for the model equation system. Three matrices are delivered – derivatives with respect to the x', x, and u vectors.

- Describe the component type, i.e. specify number of equations, variables, and parameters, plus names of all entities including module type.

- Process parameters when relevant. The NMF format contains an option for parameter processing via a routine written in some regular programming language. The set of parameters that are best suited for a mathematical model description are often others than a user would naturally select. The parameter processing takes care of the transformation from the user's 'easy access parameters' to those finally used in the evaluation routines.

The Fortran routines are precompiled and organized in component libraries, which are maintained by some special support utilities.

3.4 Differential-Algebraic Equation Modelling

The basic modelling technique used in IDA is equation based. A general formulation of the equations for a module is a system

$$0 = F(x', x, u, p, t) \qquad (3)$$

The system is fully implicit and in general differential-algebraic, i.e. $\partial F/\partial x'$ may be singular. The declarative equation modelling is complemented by limited assignment modelling in order to cater for handling of hysteresis and discontinuities. This extension is discussed in detail in section 4.

3.5 Input/Output Free Modelling

In the NMF models, the ***x*** variables are specified as OUT-variables and the ***u*** variables as IN-variables. This should not be taken as a permanent interpretation, but rather as one *possible* interpretation. Among the possible designations of variables as *x* or *u*, the one chosen in the model description should have the property that the *x*-variables could normally be calculated from the *u*-variables. For some component models, e.g. thermostats and other control components, only one designation is possible, but in general there is an arbitrary choice between equivalent alternatives. In this case, the roles as IN- or OUT-variables could only be finally assigned in relation to a specific simulation problem, when all connections in a simulated system have been completely specified. This is done automatically in IDA.

3.6 Interconnection of Modules

In the NMF format, the models have interfaces organized as links, where each link has a specified type. A link type comprises a specific set of variables of given types and in a prescribed order. This link concept allows simple and efficient interconnection of components while supporting compatibility checks on connections. Both links and variables can be given generic types to cater for cases where type checking is not desired.

The link concept and the type checking as described above are handled by the modeller. When the solver is run as a stand-alone system, connections are specified at the variable level without type checking.

3.7 Integration Techniques

The integration algorithm chosen for IDA belongs to a class of backward differentiation methods called Modified One-Leg Collocation methods (MOLCOL) (Eriksson 1983). These have proven to be very effective on stiff problems. They include, by choice of parameters, the implicit midpoint method and Gears methods, such as the backward Euler. The implementation use automatic adjustment of timestep and integration order.

The Jacobian matrices needed are generated at the module level, either by a component subroutine calculating derivatives analytically, or by numeric differentiation.

3.8 Modular Algorithms, Utilizing Sparseness

The basic calculation of residuals and Jacobians is always done at the module level. Several different algorithms are available for the solution of the global system.

One of the methods eliminates all coupling equations, identifying connected variables with each other, and generates a global system matrix from the component matrices. Currently, no sparseness technique is applied on the resulting global system.

The other methods retain the modular structure. In each Newton iteration the modular systems are first solved locally, then the residuals in the coupling equations are eliminated, causing updates of the local solutions. In this process the coupling equations are treated differently in the different methods, but one common feature is the generation of a Schur complement from the global system. One alternative treats the full Schur matrix, others presort the modules to make the Schur matrix block band diagonal or the full system bordered block upper triangular. The latter approach is similar to the one used in SPANK, except that it operates on the module rather than the equation level.

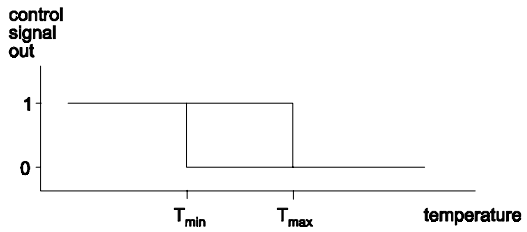
4. DISCONTINUITY HANDLING

4.1 Discrete System States and Hysteresis

So far we have been discussing simultaneous differential-algebraic equations as a means to describe the internal behavior of a continuous component model. Declarative modelling with DAE's is suitable for a very large group of models and an efficient solver explicitly restricted to such systems is a valuable tool per se. However, some frequently needed components do not lend themselves to strict equation modelling so the repertoire has to be extended.

The most obvious example of such a component is probably a simple thermostat with a dead band. In the sequel, we will make heavy use of that example to illustrate important aspects of the related problems and their solutions.

The figure below illustrates the fundamental functional relation describing the behavior of a thermostat closing on low temperature.



The control signal is a function of the temperature, but it is not unique in the dead band between T_{min} and T_{max}. To select the proper solution in that area, we must know the current state of the thermostat. That system state can not be handled as a normal continuous variable. We need some new mechanism to take care of discrete system states and hysteresis phenomena in general.

4.2 Assigned States

A new kind of variable, an ASSIGNED STATE, is introduced for this purpose. A_S-variables are updated by explicit assignments rather than by equation solving. The values are remembered between successive evaluations of the equation model. The A_Ss are similar to SAVED local variables in HVACSIM+.

It should be emphasized that the strong case for equation modelling presented above is still valid. For the great majority of continuous models straight equation modelling is still sufficient. A_S-variables will be used only when there is a genuine need for them.

Let's now formulate the central section of a thermostat model, using an assigned state:

```
EQUATIONS
    0 = – out_signal + IF T > T_max THEN 0.
                       ELSE IF T < T_min THEN 1.
                       ELSE old_signal ;

ASSIGNMENTS
    old_signal := out_signal ;
```

'out_signal' is a normal continuous variable, solved from an equation, while 'old_signal' is an A_S updated by assignment.

4.3 Updating assigned states

In each iteration of each tentative timestep, the A_Ss are updated as directed in the model descriptions. A solver must however also maintain a second copy of all A_Ss, a copy which is updated only at the end of an accepted timestep. Prior to each iteration, every A_S is given the value of its copy, i.e. the value it had after the last accepted timestep. An example might clarify this; let's consider a model with one single A_S, updated by the assignment:

```
i := i + 1 ;
```

At the end of a simulation, 'i' would hold the total number of timesteps, not the total number of iterations.

4.4 Order between equations and assignments

Equations in a wholly continuous model do not have to be written in any particular order, the system is solved simultaneously. The situation changes, however, when assignments are included in the model. Assignment statements are ordered sequentially with respect to each other and the order is normally quite essential for the total effect. When equations are mixed with assignments it must be clear at what stage of the assignment process that the equations apply. Mixing of equations and assignments can create ambiguities and make solutions undefined. To create clarity, it is reasonable to impose the following rule:

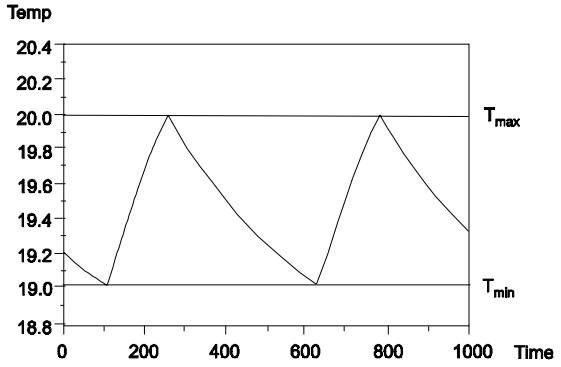
– Assigned states, which are used to carry information from one timestep to the next, are assigned after the equation solving in each timestep. The assignments are also written after the equations in the model description.

4.5 Discontinuities in equations

The extended modelling repertoire presented above has been used to good effect to cater for hysteresis. This has been achieved by allowing system states to

be remembered between timesteps and by supplying mechanisms for their change. However, shifts in system states will quite often be accompanied by discontinuous changes in equation variables and in global functions appearing in equations. These changes will require more attention than they have got so far; in fact the model presented above is flawed in a quite serious manner!

Let's put the thermostat model to use in a simple system. A thermal mass is connected by a linear conductance to a constant ambient temperature and heated by a constant heat source controlled by a thermostat. If the power is sufficient, the heating will be on intermittently and the temperature of the mass will vary like this:



Look at the situation when the rising temperature T approaches T_{max}. Any timestep long enough to let T reach the limit will leave T undefined: If we try with $T > T_{max}$ the heating will be off and T will not reach T_{max}; if we try with $T < T_{max}$ the heating will be on and T should exceed T_{max}, etc. A slight change in the thermostat model will solve the problem:

EQUATIONS

0 = out_signal – old_signal ;

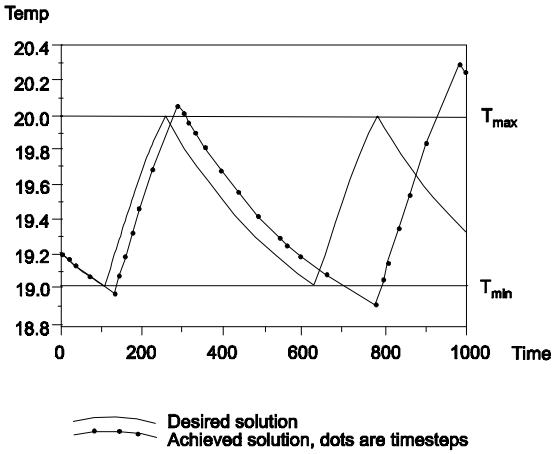
ASSIGNMENTS

IF $T > T_{max}$ THEN old_signal := 0.
ELSE IF $T < T_{min}$ THEN old_signal := 1.
ELSE ;

4.6 Control of timestep at discontinuities

The latest version of the thermostat model finds a solution to the sample problem. It lets the temperature reach the limits, the thermostat state will change between timesteps, and the next step will set off in the new direction.

In fact, the temperature will not only reach the limits but exceed them; the calculated solution will look like this:



The deviation from the desired solution affects the accuracy in an uncontrolled manner depending on the time step. Improved control of the accuracy requires proper choice of timesteps.

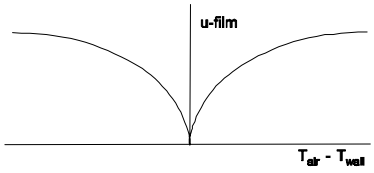
Choosing a suitable step for a solver with fixed timestep is mostly done by guesswork since a proper control can only be achieved by varying the step systematically until a picture emerges of how accuracy depends on timestep.

Using variable timestep, the step is typically controlled by estimates of local error, a procedure which normally gives good control of accuracy. However, the discontinuities currently discussed introduce new features, not handled properly by the normal error estimates. The sometimes very long steps taken by variable timestep solvers could introduce large errors if timestep is not controlled at discontinuities.

Before we look at means to handle this problem, we should make an inventory of the different types of discontinuities we have to cope with and the types of problems they are likely to create.

4.7 Events

Any discontinuous change in a variable or in its derivative will be called an event. Causes of events can be external, i.e. localized in driving functions, or internal, appearing in component model equations or in global functions used in the equations. Our thermostat is an example of a component model where events are generated visibly in the model. As an example of a global function generating events, we might consider convective heat transfer at wall surfaces showing a relation like this:



This film coefficient would normally be calculated, not explicitly in a model, but by a call of a global
function.

Independent of event source, the primary effect of a discontinuity on the regular variables can appear, either in a variable value or in its derivative. Let's call the first case a 'jump' event and the second a 'knee' event.

If convective heat transfer is calculated by the above function, the passage through zero of any corresponding temperature difference $T_{air}–T_{wall}$ should be regarded as a knee event. Other examples of events are e.g. – starting of a fan (jump), P-regulator reaching limit (knee), shift from turbulent to laminar regime (jump or knee depending on modelling).

The computational implications of an event for the solver will depend on event type and possibly on the modelling context of the event.

A knee event will leave all variables continuous and should be possible to negotiate by careful handling of timesteps.

The effect of a jump event will depend on what equations it disturbs. If it only appears in differential equations, the change can be 'absorbed' by the derivatives of the dynamic variables and the overall effect on the system will be of the knee type. If it affects any algebraic equation, however, the jump discontinuity will show up in other variable values as well, and an initial value calculation is required before continuous integration can resume. In our sample system, a jump change in the control signal from the thermostat will only cause a knee type effect for the system.

4.8 Event Handling

The crude way to handle events in a variable timestep solver is to let the events 'surprise' the solution algorithm. A long step across a discontinuity will mostly fail to converge or give unacceptable local errors, but it can also happen that the event passes undetected without any reliable control of accuracy. If the event is detected by its ill effects, the algorithm will repeatedly shorten the step until convergence hopefully occurs. This approach obviously leaves much to be desired.

What is needed is a means to safely detect when an event occurs and a method to pass the event in a controlled manner. When an event is handled in the post equation assignment section, the solver will not feel any ill effects of the discontinuity in the timestep where the state change occurs, since the change is postponed till after the equation solving. Without support for event detection a too long step could be completed before the changed state causes detectable effects.

Thus, in order to notify the solver of events occurring during an attempted timestep, the model descriptions must be given means to explicitly signal events. Furthermore, we want to allow event generation inside functions declared globally but called from different modules. A key feature of event detection must be some type of memory, e.g. the film coefficient function shown above must be able to detect that the current temperature difference has changed sign from the previous evaluation. Since the film function is used by different modules, the old sign can not be stored in the function but must be remembered by the calling module, where an A_S could be the bearer of the memory.The required features could be supplied by a generic system function:

```
REAL FUNCTION event_xyz (event_var,
                         event_expr)
```

where the suffix 'xyz' will differentiate between variants for specific purposes. The functions signal events when their second argument passes through zero. The first argument is an assigned state which tells where the signal was at the last accepted timestep. Variants are used to trigger on rising or falling slopes only, and to differentiate between jump and knee events.

Applied in a film coefficient function using a stepwise linear curve approximation, the use of event functions could in principle look like this:

```
REAL FUNCTION U_film (T_air, T_wall, Diff)
      REAL T_air, T_wall, Diff
C     INPUT T_air, T_wall, Diff
C     OUTPUT Diff
      REAL D
      D = T_air – T_wall
      IF event_k (Diff, D) < 0. THEN
            D = –D
      ENDIF
      IF D < d1 THEN
            U_film = a1 ∗ D + b1
      ELSE IF D < d2 THEN
            U_film = a2 ∗ D + b2
      ...
      ENDIF
END
```

'event_k' signals a knee event, 'Diff' is an A_S carrying the the old sign. A call of the function from

a module could look:

$$c * T' = A * U\_film\ (T\_amb, T, Diff\_old) * (T\_amb - T)\ .$$

'Diff_old' is declared in the module as an A_S.

The event functions fill several tasks, they will:

- signal the occurrence and possibly the location of an event via the time variation of 'event_expr',
- signal the type of event, knee or jump,
- deliver a function value 'event_var',
- update the assigned state 'event_var' with the expression 'event_expr' (any 'event_var' used inside a global function must originally have been defined at the top level).

The solver will use the signals from the event functions to:

- localize events in time,
- place one evaluation close to knee events, then start out with small timesteps,
- place two evaluations close on either side of jump events, invoking initial value calculation after the passage.

Let's now rewrite the thermostat model, introducing event functions:

```
EQUATIONS
    0 = out_signal – old_signal ;

ASSIGNMENTS
    IF event_p (high, T – T_max) > 0
        THEN old_signal := 0.
    ELSE IF event_n (low, T – T_min) < 0
        THEN old_signal := 1.
    ELSE ;
```

'event_p' triggers on second arguments going positive, 'event_n' on second arguments going negative; 'high' and 'low' are A_Ss.

4.9 Summing up discontinuity handling

Handling discontinuities in basically continuous simulation problems, the main concerns are, to:

- safely pass discontinuities without convergence problems,
- retain control of accuracy in the process,
- achieve these goals without overly complicating things for the model developer.

The tools supplied in IDA are a good step in that direction. The area is still under development, so changes in the specification are likely. Improvements of the event localizing process can be implemented in the solver, without affecting the modelling format.

5. SUMMARY

For building simulation to mature and take full advantage of the possibilities offered by improving computer resources, it is essential that development be based on solid foundations, the most important features being:

- systematic modelling methods, supported by user friendly interfaces,
- robust, efficient integration techniques for DAE systems with discontinuities, affording full control of accuracy,
- standardized machine readable model documentation, facilitating international cooperation and reuse of models.

IDA has been developed with such aims in mind. It has proved to be a useful tool for practical simulation work, but much essential RD work remains to do; among the more imminent tasks could be mentioned:

Initial value calculations

- incorporate gradient methods,
- allow for linearized model versions to supply good starting direction,
- implement automated successive choice of methods.

Utilizing problem structure

- utilize separate storage and integration methods for FD components with special characteristics,
- further improve module and equation sorting at large,

Modelling

- refine GUI and develop post processing in cooperation with user groups,

– port modeller to PCs and to other work stations.

In the longer perspective we see hooking up with CAD and product models as an important field of research.

Continuous simulation has for a long time been an art, mainly because mastering of the available tools has required specialized expertise. Hopefully, the emergence of good simulation environments will lead to increased use of quality simulations for research as well as for all phases of the building process.

ACKNOWLEDGEMENTS

The authors are indebted to the following people who have participated in the development of the IDA simulation environment and also contributed to the paper: Lars Eriksson and Magnus Lindgren from ITM, Gustaf Söderlind from the Lund Technical University, and Kjell Kolsaker from the Norwegian Technical University, Trondheim.

REFERENCES

Bonneau, D.; D. Covalet; D. Gautier; and F.X. Rongere. 1989. "Manuel de Prise en Main, CLIM 2000, version 0.0." Research Report HE 12 W 2867. Electricite de France.

Bring, A. 1990. "IDA SOLVER, a User's Guide." Research Report. Dept. of Building Services Engineering, Royal Institute of Technology, Stockholm.

Buhl, W.F.; E.F. Sowell; J.M. Nataf. 1989. "Object Oriented Programming, Equation Based Submodels, and System Reduction in SPANK." Building Simulation '89 Conference (Vancouver).

Clarke, J.A. 1989. "An Object-Oriented Approach to Building Performance Modelling."

Cools, C.; R. Gicquel; and F.P. Neirac. 1989 "Identification of Building Reduced Models. Application to the Characterization of Passive Solar Components." *International Journal for Solar Energy* 7: 127-158.

Duff, I.S.; A.M. Erisman; and J.K. Reid. 1986. *Direct Methods for Sparse Matrices*. Oxford University Press.

Elmqvist, H. 1986. "LICS: Language for Implementation of Control Systems," Dept. of Automatic Control, Lund Institute of Technology.

Eriksson, L. 1983. "MOLCOL – An Implementation of One-leg Methods for Partitioned Stiff ODEs." Report TRITA-NA-8319. Royal Institute of Technology, Stockholm.

Kolsaker, K. 1991. "An NMF-Based Component Library for Fire Simulation" To be presented at *IBPSA BS'91* (Nice, Aug. 20-22).

Kron, G. 1963. *Diakoptics, the Piecewise Solution of Large-scale Systems*. Macdonald, London.

Lorenz, F. 1989. "Acausal Information Bonds in Bond Graph Models" Symposium AIPAC '89 (Nancy, July 3-5).

Lundh, N.; K. Gustafsson; and G. Söderlind. 1988. "A PI Step Size Control for the Numerical Integration of Ordinary Differential Equations." *Bit* 28: 270-287.

Mattson, S.E. 1986. "On Differential/Algebraic Systems." Research Report CODEN: LUTFD2/-(TFRT-7327)/1-026. Dept. of Automatic Control, Lund Institute of Technology (Sept).

Mattsson, S.E. 1988. "On Model Structuring Concepts," Presented at 4th IFAC Symposium on Computer Aided Design in Control Systems (CADCS), Beijing, China.

Mattson, S.E. 1989. "An Environment for Model Development and Simulation." Research Report CODEN: LUTFD2/(TFRT-3205)/1-030. Lund Institute of Technology (Sept).

Nakhle, M. 1986. "Neptunix, an Efficient Tool for Large Scale Systems Simulation." In *Proceedings of the Second International Conference on System Simulation in Buildings* (Liege, Dec. 1-3).

Sahlin, P. 1988. "MODSIM: A Program for Dynamical Modelling and Simulation of Continuous Systems." In *Proceedings of the 30th annual meeting of the Scandinavian Simulation Society*, ISSN 0357-9387.

Sahlin, P. and E.F. Sowell. 1989. "A Neutral Format for Building Simulation Models." In *Proceedings of IBPSA BS'89* (Vancouver).

Söderlind, G; L.O. Eriksson; A. Bring. 1988. "Numerical Methods for the Simulation of Modular Dynamical Systems." Research Report. Swedish Institute of Applied Mathematics, Gothenburg.

IDA Modeller

a Man-Model Interface for Building Simulation

Per Sahlin

Dept. of Building Services Engineering
Royal Institute of Technology

ABSTRACT

The limited development potential of current building simulation programs has spurred the design of a new generation of tools: object oriented simulation environments, where the latest in software technology and numerical methods is employed to provide users with a framework for more flexible, and thereby more appropriate, simulation models. Some of these tools provide the sophisticated user with a rich graphical environment for interactive model design. We call these model-lab simulation environments. However, it is not always clear how the new tools will be brought into every day use by non-experts at design offices. In this paper, some requirements for such use are discussed, and IDA Modeller is presented. IDA Modeller is the front end of IDA, a general environment for building and energy systems simulation. IDA allows use of model-lab model structuring principles for development of end user design tools. Discussed features of IDA Modeller include: hierarchical model structure, object user interfaces, and tailored IDA applications. For illustration, the user interface of a recently developed application, multizone air-exchange, is presented. The mathematical models and methods of this application are treated in an accompanying paper.

1. INTRODUCTION

In recent years, several modelling environments have been developed for flexible construction, manipulation and maintenance of simulation models. The accepted term for the new tools is *object oriented simulation environments*, which highlights the focus on modular structure, both for models and for the software itself. The primary motivation for the new developments comes from the lack of flexibility of current building simulation tools. Building simulation researchers are in remarkable agreement over the need for new technology, and also over some of the key ingredients, such as: object oriented programming techniques; a clear separation between modelling and simulation (solution) activities; and utilisation of new GUI techniques.

The majority of the new environments are primarily intended to offer powerful facilities for interactive model building, using graphical techniques. For an experienced engineer, a system model is easily created by fetching sub models from a library and interconnecting them graphically. Examples of such *model-lab* type environments are ALLAN (CISI 1990), SANDYS (Ohlsson 1991), CLIM 2000 (Bonneau et al. 1989), PRESIM-TRNSYS (PRESIM 1988), OMSIM (Mattson and Andersson 1993), SPARK (Sowell et al. 1993), and MS1 (Lorenz 1991) (In a loosely defined order of completion). However, these new programs will, in their basic form, have little to offer to users of traditional building simulation tools. For these users, the principal interest is not so much efficient model tailoring, but rather, rapid and reliable building performance appraisal.

On the other hand, environments such as the UK Energy Kernel System (Charlesworth et al. 1991) are clearly aimed at efficient end-user tool production, perhaps at the expense of model-lab facilities.

IDA Modeller is a new environment that tries to serve both the above mentioned goals. It is in essence a model-lab program, but it also has facilities for customising of model user interfaces to cater for various end-user needs. Used as a programming tool-kit, the Modeller offers a rich infrastructure for design tool development. Models may easily be docked to foreign input-output programs, such as building product model interfaces. New utilities can be incorporated,

Dept. of Building Services Engineering,
Royal Institute of Technology, 100 44 STOCKHOLM
Phone: +46-8-11 32 38, fax: +46-8-11 84 32,
e-mail: plurre@engserv.kth.se

such as optimisation algorithms requiring repeated simulation runs.

IDA Modeller is primarily intended to serve as front-end to IDA Solver, but may be adapted to drive other differential-algebraic equation (DAE) solvers. Together with Neutral Model Format (Sahlin, Bring and Sowell 1992) translators, IDA Modeller and Solver form the IDA simulation environment for building and energy systems simulation. IDA has been developed at the Swedish Institute of Applied Mathematics in co-operation with the dept. of Building Services Engineering at KTH, Stockholm. IDA Solver has been under testing for a number of years, while a limited version of the Modeller has been released only recently.

This paper starts with a discussion of model-lab environments and their ultimate use to building simulation end-users. This is followed by a brief account of some key features of the IDA Modeller. For illustration a practical example of an IDA application is used: multizone air-exchange. This IDA application is currently tested as a design tool for clean room facilities at ABB Indoor Climate.

2. MODEL-LAB ENVIRONMENTS

In the previous discussion we have loosely introduced the term *model-lab* simulation environment. In this section we will look a little closer at this concept and discuss some implications for end-user tool production. Let us begin by defining a set of key characteristics:

- All models adhere to a set of *model structuring principles*, that facilitate flexible model construction, interconnection, and - above all - reuse. Several structuring principles are possible and formalisms may be fetched from, e.g., Bond-Graphs, equation based languages, graphical languages or combinations of these. The principles used in IDA will be described in some detail.

- Internal behaviour of primitive models is described by *differential-algebraic* equations (DAE), or some equivalent mode of expression. Some environments also provide methods for expression of non-continuous models.

- A user builds a simulation model by interconnection of submodels. This is usually done by selecting models from a library and interconnecting them graphically.

- Simple means are provided to define new library models. Primitive models are formulated in a special *modelling language*. The functionality of the environment may hinge on the characteristics of this language.

- Parameters are given to specialise models and to select a suitable numerical solution scheme.

- Boundary conditions are defined to ensure a solvable problem. This is a comparatively easy task for input-output oriented solvers (like TRNSYS), one simply gives all input variables, but considerably more demanding for input-output free environments (like IDA and SPARK). For these environments, solvable problems will result from many different sets of given variables, and all these sets are permissible. This adds tremendous flexibility by increasing the number of what-if questions that can be studied with a given model, but puts the burden of defining solvable problems on the user.

- Start values are given to state variables, and may be given to algebraic variables to support calculation of initial values for nonlinear systems. Depending on system type this may be a quite demanding task.

In addition to this basic functionality a number of other useful features may be offered. For example, transparent access to programs for collection and treatment of experimental data and to tools for signal processing and system identification.

Examples of environments that fall within the boundaries of this definition were given in Section 1. Some of these environments are mostly intended for a single field of application, although they are in principle general. Others have no application affiliation.

Note that the list of environments would have been many times longer if ordinary differential equations (ODE) had been specified rather than DAEs. We will not go into the difference at length here, but merely point out that (a) there *is* a fundamental difference between ODEs and DAEs (see, e.g., (Mattsson 1986)) and (b) DAEs are often indispensable for building simulation problems due to frequently occurring pressure-flow networks.

All of these model-lab environments have been developed within the last few years, and although some are quite useful today, a number of fundamental aspects need to be studied further. Most of the programs are of considerable complexity and usually a number of supporting programs are utilised at run time, such as compilers and computer algebra packages.

Model-lab simulation tools will enable a closer study of many important physical systems and will be invaluable in the process of optimising their efficiency. However, it should be recognised that these sophisticated tools, in their basic form, will be useful primarily to researchers and research oriented engineers, and will not address the needs of the typical end-user.

2.1. Meeting End-User Needs

Our view is that a model-lab environment can be an ideal platform for development of building simulation end-user tools, but not all model-lab approaches will work. Provisions must be made for (1) shipment of sufficiently simple end-user versions and (2) tailoring

of model interfaces, including model specific code as well as data.

The first requirement deals with the size and cost of supporting software that must accompany an end-user version. Several systems access a (FORTRAN) compiler in the modelling-simulation loop. Shipment of restricted versions, where parameters can be varied but models remain topologically fixed, is generally possible without the compiler. However, this is rarely enough for building simulation purposes, since most standard problems call for a project specific model structure. Thermal zoning must be adapted to the design at hand and the same is true for duct and pipe network topology.

IDA is based on a technique with precompiled primitive models. Thus, a compiler is only needed for adding of new component models. This enables shipment of end-user versions with a fixed library of primitive models. These basic models can then be interconnected into complex configurations by any user. An overview of the numerical techniques of IDA is given in (Sahlin and Bring 1991).

The second requirement concerns added functionality, not just convenience and cost. It deals with the possibility of creating easy-to-use *end-user applications* with a limited and targeted functionality for a specific class of simulation problems. It is necessary to both cut down on user freedom and to provide additional support. In some model-lab approaches, the *modelling language* itself is the only user-accessible mode of expression and the language rarely provides sufficient means for good user interface design.

The route taken in IDA is to limit the role of the modelling language to that of expressing primitive models. Beyond that, an application writer has access to the entire Common Lisp language - plus any other callable language - for application design. Furthermore, the IDA model structure is designed to handle user-defined objects (code and data), specific to an application. The next few sections are devoted to a brief description of the IDA model structure and tools for application writing in the IDA framework.

3. FEATURES OF IDA MODELLER

3.1. Model Structuring Principles

Most primitive models in IDA Modeller are described as differential-algebraic systems of equations, i.e. a free mixture of first order differential and algebraic equations. Such models are referred to as *equation objects* in IDA terminology. Equation objects are specialised into classes corresponding to models of physical components, like walls, windows, boilers, coils, etc. Equation objects are in turn built up as aggregates of lower level objects such as parameter and variable objects. The IDA concept of an equation object is roughly similar to the *continuous_model* type in the Neutral Model Format (NMF) (Sahlin, Bring, and Sowell 1992) and IDA equation object class definitions are generally obtained by automatic translation from NMF.

Complex models are formulated by connection of equation objects. More specifically the *interfaces* of two equation objects are joined. The interfaces of an object define which internal variables that may be connected to the outside world. The interfaces of an IDA object correspond to the *links* of an NMF model.

Macro objects are collections of other objects, e.g. equation objects or other macro objects. Macro objects are very fundamental to the IDA model structure since they provide the means to organise models hierarchically. The IDA macro object is similar to a directory on the disk of a computer; it holds other macros (directories) and objects (files).

Macro objects can play two different roles. Macros can - like directories - be used for library purposes only, i.e. as nodes in a hierarchical access structure. In this role they are referred to as *library macros*. However, if the submodels of the macro are connected into a meaningful system model, the macro is referred to as a *system macro*. Any system macro may be simulated as it is or used as a building block for creation of more complex models (during creation of a meaningful system, a system macro may be only partially connected and not suited for simulation.)

All IDA objects are organised in a single tree of macros under a library macro named `root`. The first levels of this tree, hold library macros only, but towards the leafs, system macros start to appear.

The analogy with directories on the disk is more than just a pedagogical tool. In the Modeller each macro - system or library - actually does correspond to a physical directory with the same name, i.e. there is a one to one mapping between the macro tree and a corresponding directory tree on the disk. When experiments are performed on a system macro all files related to this activity are stored in the corresponding directory.

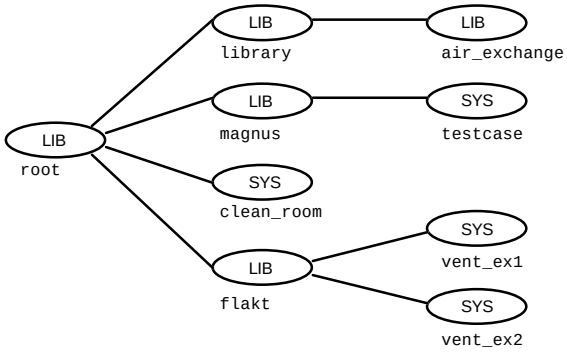


Figure 1. A selection of macros.

Any object in the tree can be copied and used as building material for new models. There is no formal distinction between models from previous projects, the ongoing project, or the "library" branch of the tree. The library branch is merely a convenient location for models that are of more general interest; models intended for reuse are naturally moved there, once they are sufficiently tested.

In a file sharing environment all IDA users should normally interact with the same model tree, and user subdivisions should be allocated within this tree. There are facilities for exporting and importing entire branches between different IDA environments, provided the same class definitions have been made in both places. Automated export and import of class definitions are planned facilities.

3.2. Model User Interfaces

Interacting with IDA involves navigation in the model tree. All available information is located in this structure, or in the associated disk structure. Each IDA object is equipped with its own user interface. A special part of the object, called the *form*, specifies how the object is to be presented to the user and what actions the user may perform. The form of an object enables object presentations to vary according to context and user category. Presentations and support facilities can be tailored for three different user types:

Component Makers is the most proficient group of users, with access to all interactive facilities;

System Makers build system models out of existing component models;

End Users perform parameter studies on already existing system models.

Some object types, such as continuous variables, have standard forms that rarely change. Other objects such as component models have simple, automatically generated, default forms, for the model experimentation phase. They can be improved once the model is ready for external use.

Object forms can be substructured in multiple levels, as indicated in Figures 2 and 3, providing a hypertext oriented presentation method. Subforms may be automatically opened as the main form is opened, to provide explicit instructions for, e.g., End Users. The form presenta-

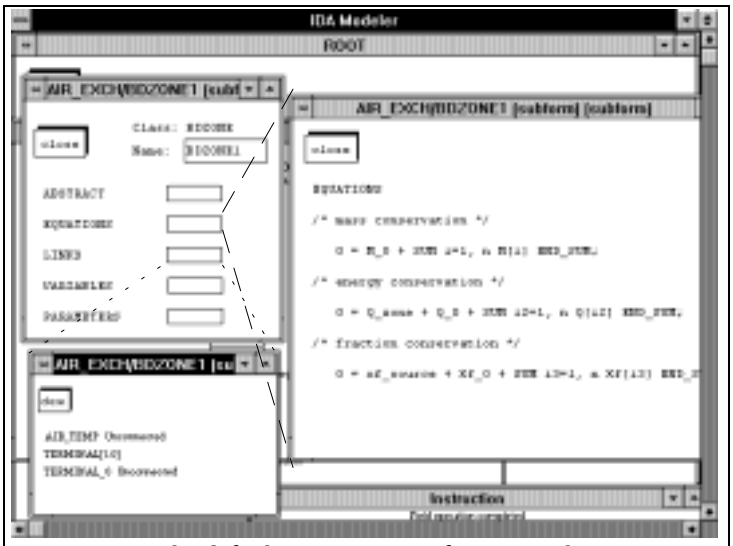


Figure 2. The default presentation of an air exchange zone model, with two open subforms, showing some details of the model

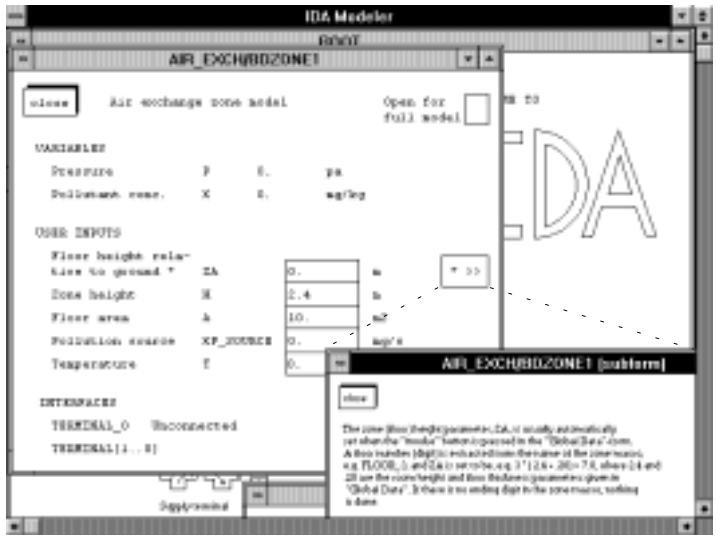


Figure 3. A tailored form for the zone model, with a supporting subform open.

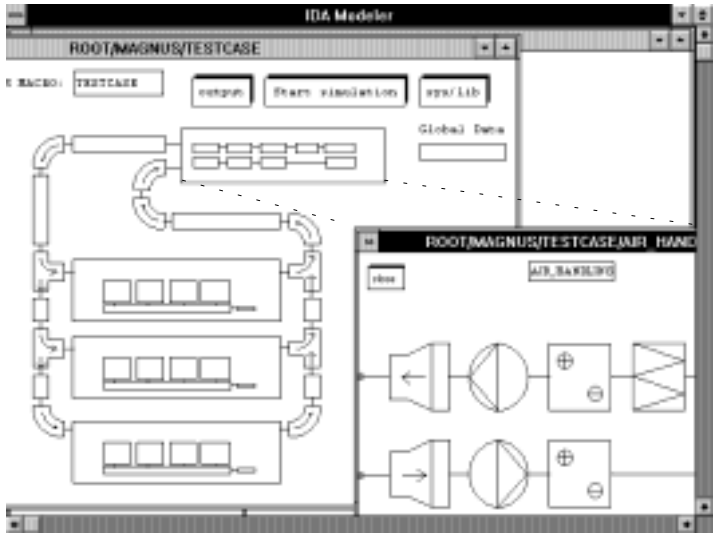


Figure 4. An air exchange building model (a system macro) with one open submacro.

tion can also be enhanced with figures, built with polylines and arcs.

The object interfaces that have been developed so far are mostly simple ones, like the one in Figure 3, but they can in principle be quite sophisticated with various chains of dialogue boxes, etc.

The main feature of a system macro form is generally the schematic model itself, which can be opened and further explored (fig. 4 and 5).

3.3. Writing and Using Applications

An IDA application is (1) a *compatible family* of component models and (2) a *set of tools* for simplifying user interaction with the models.

A *compatible model family* is a group of models that are designed to operate together and cover the modelling needs of a certain problem area.
A sample family (fig. 5), for air-exchange and infiltration modelling, is presented below.

Application support tools simplify or completely automate model building, parametrization, simulation, and post-processing. They bridge the gap between straight model-lab usage of the models - which is always possible - and *design tool usage*, i.e. using the simulation environment as a collection of separate but similar programs (applications). In this perspective the simulation environment is conceptually akin to, e.g., MS-Windows - a collection of separate programs having the same "look and feel" and sharing system resources.

The tools are tied to one or more *application macro* classes. To have access to the application tools, a user must build models within an instance of an application macro. A new work session is generally initiated by copying and naming an application macro, either a previous project macro or a blank, empty macro. This is similar to Windows file associations, i.e. a file which "knows" what application it was created with. Contrary to Windows, this is the only way of starting an IDA application; the application cannot be started on its own.

Application support tools are required to use the IDA (macro) structure for data organisation. Other system resources may or may not be used. There is, e.g., no strict requirement that IDA forms are used for user interaction. In principle, IDA can call any program for model processing, e.g. an established simulation tool interface can be called to request data from a user in a familiar way.

However, in most cases application writers will use the native resources, since significant savings can be expected. IDA offers complete automated support for object storage, retrieval and copy as well as the forms package for user interaction.

3.4. A Sample Application - Multizone Air-Exchange

The Multizone Air-Exchange (MAE) models predict pressure levels, flows and transports of contamination and energy in a zone network and in the associated ventilation system. Presently, the functionality is roughly the same as the Movecomp program (Bring

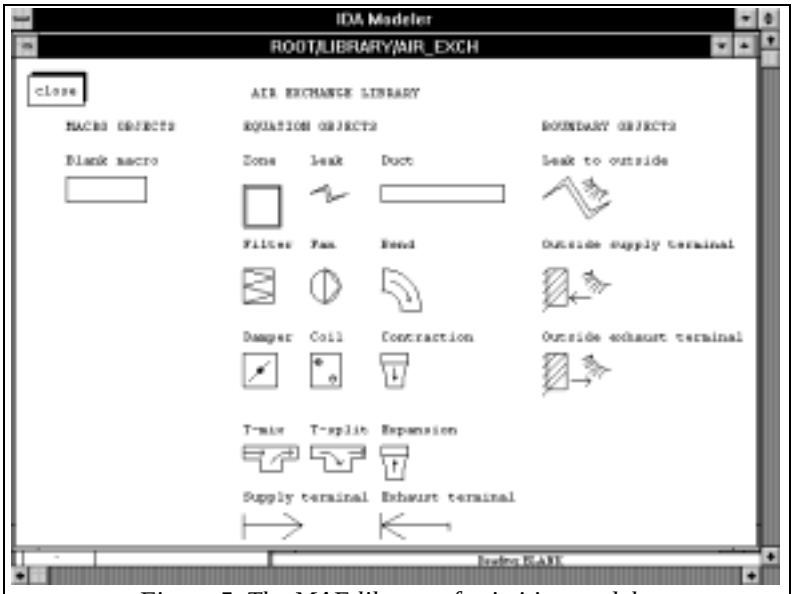


Figure 5. The MAE library of primitive models.

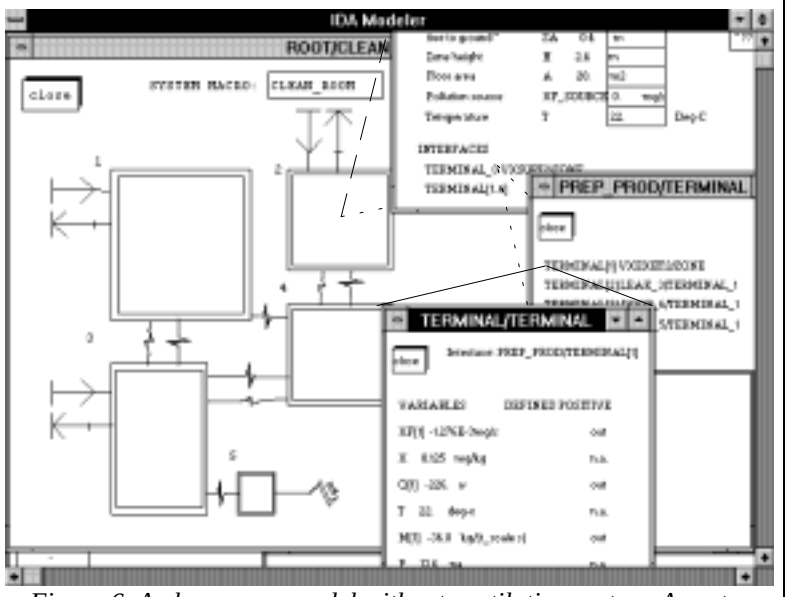


Figure 6. A clean room model without ventilation system. A vector terminal and one of its components has been opened.

and Herrlin 1991). An overview of the models and numerical methods is presented in an accompanying paper (Bring and Sahlin 1993). The primary advantage with the IDA implementation of the MAE models is the possibility of coupling them with thermal models. However, here we will concentrate on the MAE application alone.

The present version of the MAE application utilises the basic model-lab functionality of IDA for model building. The user copies, connects, and gives component parameters in the standard fashion. In the future much of this work could be done by accessing some CAD-representation of the system. MAE support tools are used for checking model consistency, automatic parameter propagation, simulation problem formulation, and post-processing support.

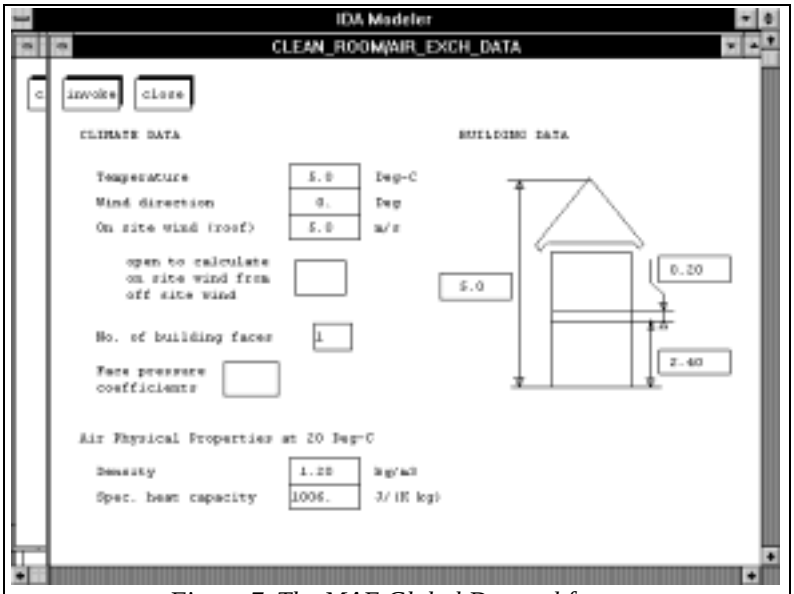


Figure 7. The MAE Global Data subform.

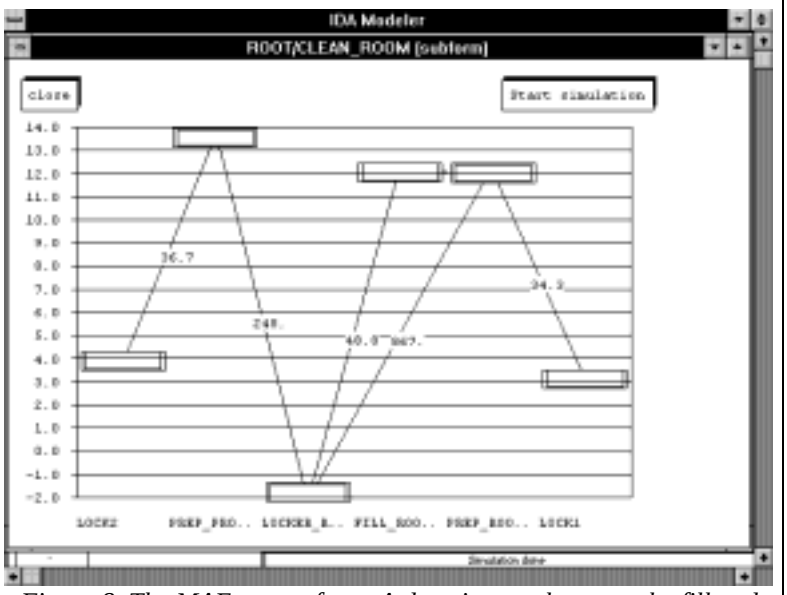


Figure 8. The MAE output form. A door is open between the fill and prep rooms, hence the slight difference in pressure. Zone models can be opened and parameter runs carried out directly from this form.

Generally, application tools can either, like the MAE tools, be available for use as needed, or impose themselves and direct the user rigidly through every step - depending on user sophistication.

In the MAE framework a user can, after having completed the model building phase, open a macro subform 'Global Data' (fig. 7) and enter parameters like outside temperature, wind properties, and building pressure coefficients (how the pressures at the faces of the building vary with wind loading).

After completion of the 'Global Data' forms, the 'invoke' button is pressed. The 'invoke' algorithm will check model consistency (as far as possible), propagate global data, and set up a standard solvable simulation problem.

After a steady state simulation has been carried out, results can be studied either by inspection of individual variables or by pressing the MAE 'output' button, which sets up a special display (fig. 8)where zones are oriented vertically according to relative pressure and where labelled lines between zones indicate flows and flow paths.

Generally, for transient simulations, a user may view the development of individual variable values as the simulation progresses. In model-lab usage of IDA, the operator may also stop the simulation at any time and inspect system state and then resume simulation or terminate.

4. CURRENT STATE

Development of IDA Modeller commenced in 1987. Since then, some five man years have been devoted to development and experimentation. Until recently, the development platform has been Lucid Common Lisp on Apollo workstations. During the fall of 1992, IDA Modeller was ported to the Allegro Common Lisp environment under MS-Windows on the PC. Present minimum hardware requirement is a 386 PC with 8 Mb core memory; more enjoyable is a 486 system with 16 Mb. A full IDA development environment requires access to Common Lisp and FORTRAN compilers, but end-user versions may be shipped as stand alone programs without need for licences for supporting software.

Due to lacking standards in 1987, proprietary object and window systems were developed in the workstation setting. Since then, the Common Lisp Object System (CLOS) has been standardised and several GUI systems have gained wider acceptance. The present Windows port is rather rough, with only a minimum of standard functionality, i.e. the Modeller does not yet have Windows "look and feel." The IDA object system has recently been replaced with CLOS, leaving original object constructor and manipulator functions intact.

Some of the model-lab functionality for transient simulation that was present on Apollo has at this point (March '93) not yet been implemented in the PC-setting. The first priority has been to get the necessary (steady state) MAE features in operation. Similarly, component modelling and application writing still require a significant amount of handicraft.

However, despite the mentioned shortcomings, the full IDA system with the MAE application has proved useful to novel industrial users (system makers) with very limited experience of similar software.

5. CONCLUSIONS

The primary objective of the IDA-project as a whole has been to investigate the viability of a model-lab simulation environment as a framework for building simulation and design tool production. Anticipated problems included insufficient efficiency, robustness, and compactness of end-user tools. Other concerns were the applicability of general numerical methods to building simulation problems ranging from traditional thermal modelling to equipment control and fluid flow problems.

Fortunately, none of the potential obstacles have proved to be insurmountable. The cost of a general approach is entirely acceptable, especially when the improved performance of affordable hardware is taken into consideration. Traditional thermal simulation problems require roughly two to five times more processing power with the suggested approach. This should be compared to desktop computer improvements over the last few years.

The size of a simulation environment like IDA may seem to be a problem; a naked (no models) environment occupies on the order of five megabytes of memory, with a few applications and examples perhaps eight. Of course the same price/performance argument as before applies here as well. However, since most users will host more than one application and several models, one should really look at the marginal size cost of each additional model and application in the environment, rather than the size of a naked environment. Although no systematic study of this has been done, we can safely say that this cost is of the same order as that of tailored stand alone programs.

The most pleasing results relate to the range of problem types that may be treated. Since the power of variable timestep and order implicit DAE solvers are available to non-experts in IDA, users are generally delighted with being able to solve problems that previously were out of reach. The joy of writing down equations and getting results automatically or improving building performance by playing with new system solutions is most likely what will make the new simulation environments popular.

In conclusion, the IDA project has resulted in a feasible new approach to design tool development. The software is sufficiently mature to serve as safe raw material for commercial development.

REFERENCES

Bonneau, D; Covalet, D; Gautier, D; Rongere F-X. 1989. *Manuel de Prise en Main - CLIM 2000.* HE 12 W 2867. EDF - Direction des Etudes et Recherches, Moret-sur-Loing, France.

Bring, A; Herrlin, M. 1991. Bris Data AB, Calscand International. *User's Manual, MOVECOMP-PC, An Air Infiltration and Ventilation System Program.*

Bring, A; Sahlin, P. 1993. "Modelling Air Flows and Buildings with NMF and IDA." To appear in *Proceedings of the IBPSA Building Simulation '93* (Adelaide, Aug.).

Charlesworth, P., Clarke, J.A., Hammond, G., Irving, A., James, K., Lee, B., Lockley, S., Mac Randal, D., Tang, D., Wiltshire, T.J., Wright, A.J., "The Energy Kernel System" In *Proceedings of the IBPSA Building Simulation '91* (Nice, Aug.)

CISI INGENIERIE 1990. *Le Logiciel ALLAN - ALLAN Simulation - Présentation Générale.* Rungis, France.

Lorenz, F. 1991. "Modelling Platform with Multiple Representation Formalisms." In *Proceedings of the IBPSA Building Simulation '91* (Nice, Aug.).

Mattson, S.E. 1986. "On Differential/Algebraic Systems." Report CODEN: LUTFD2/(TFRT-7327)/1-026. Dept. of Automatic Control, Lund Institute of Technology, Sweden.

Mattson, S.E; Andersson, M. 1993. "Omola - An Object-Oriented Modeling Language." In *Recent Advances in Computer Aided Control Systems Engineering*, M. Jamshidi and C.J. Herget, Elsevier Science Publishers, Holland.

Ohlsson, B. 1991. "SANDYS." Research Report RES/KEB/RM-91/001. ABB Corporate Research, Västerås, Sweden.

PRESIM, Working Group. 1988. *Manual for PRESIM - A Preprocessor for Producing TRNSYS Input Data.* Solar Energy Research Center, Univ. of Borlänge, Sweden.

Sahlin, P; Bring, A. 1991. "IDA Solver - a Tool for Building and Energy Systems Simulation." In *Proceedings of the IBPSA Building Simulation '91* (Nice, Aug.).

Sahlin, P; Bring, A; Sowell, E. 1992. "The Neutral Model Format for Building Simulation." Bulletin 24. Dept. of

Building Services Engineering, Royal Institute of Technology, Stockholm, Sweden.

Sowell, E. F; Winkelmann, F.C; Buhl, W.F; Erdem A.E. 1993. "Recent Improvements in SPARK: Strong Component Decomposition, Multi-valued Objects, and Graphical Interface." To appear in *Proceedings of the IBPSA Building Simulation '93* (Adelaide, Aug.).

Paper 5, presented at the IBPSA Building Simulation '93 conference, Adelaide, Australia, August, 1993

Modelling Air Flows and Buildings with NMF and IDA

Axel Bring **Per Sahlin**

Building Engineering Services
Royal Institute of Technology

ABSTRACT

New object oriented simulation environments offer dramatically improved possibilities for simulation of coupled systems. In contrast to traditional building simulation, where separate, stand-alone tools are used for each simulation task, the new environments will offer a unified framework for all simulation problems. For design tool users this leads to two major advantages: (1) model coupling and comparison of results between different simulation applications will be practical, and (2) usage and input data will be standardised across applications. Thereby tool usability will increase considerably. Furthermore, man-time spent on model development and maintenance is expected to decrease, making it feasible to tailor appropriate models to each task, and thus improve the quality of simulation results. In this paper, we report our experiences implementing a set of models for multi-zone air-exchange (MAE) in the general simulation environment IDA. Although MAE analysis is yet far from a standard industrial practice, several tailored tools have been developed. For comparison between general programs and specialised, the MAE models are extremely challenging, since the model structure can be utilised to an unusual degree in the specialised tools. In spite of this, the results obtained are encouraging in terms of processing time as well as numerical reliability (robustness). The paper presents a selection of the models, including the full Neutral Model Format (NMF) code, a discussion of numerical methods, and some general conclusions with bearing on simulation tool development.

1. INTRODUCTION

Studies of inter-zonal air flows in multi zone buildings have attracted interest for considerable time. Several tools for handling of these problems have emerged during the last decade, e.g., AIRNET (Walton 1989), Movecomp (Bring and Herrlin 1991), and COMIS (Feustel and Rayner-Hooson 1990). These tools are tailored to handle air flows only, leaving it to the user to choose thermal boundary conditions.

The interaction between mass transports (air or water) and heat transfer in buildings is frequently significant. Thus, simulations of coupled systems are of interest, but, so far, fairly little work has been done in this field. For instance, (Hensen 1991) reports on such work using the ESP environment, but overall there has been a lack of adequate software for this type of studies.

Dept. of Building Services Engineering,
Royal Institute of Technology, 100 44 STOCKHOLM
phone: +46-8-11 32 38, fax: +46-8-11 84 32,
e-mail: abring or plurre@engserv.kth.se

Since the mid eighties, general, object oriented environments for building simulation are under development and in various stages of completion, e.g. CLIM 2000, EKS, SPARK, ZOOM. One important aim of these environments is to facilitate studies of coupled buildings and systems. The new tools will offer a unified handling of all component models, and thus allow simultaneous simulation of subsystems that hitherto have been handled by different programs. The general tools will normally be computationally less efficient than the specialised tools replaced; almost universally this drawback will be outweighed by the gain in man time efficiency.

Using the modern simulation environment IDA, a model family has been developed for studies of

NMF-BASED ASPECT MODELS IN STEP/EXPRESS FOR BUILDING AND PROCESS PLANT SIMULATION

Per Sahlin
Building Services Engineering
Royal Institute of Technology
100 44 STOCKHOLM, SWEDEN
e-mail: plurre@kth.se

Curt Johansson
Construction Management
Royal Institute of Technology
100 44 STOCKHOLM, SWEDEN
e-mail: curt@ce.kth.se

Abstract

Automated design performance assessment through simulation will be an important aspect of future product model technology. The research in this area has so far been focused on traditional simulation tools. However, the rapid development of new structurally different tools calls for a shift of attention. New object-oriented methods of describing simulation models can and should be integrated with the product model itself. In this paper we will briefly review a current development trend in continuous simulation and present a new language for model description, Neutral Model Format (NMF), which in recent years has gained considerable attention in the field of building simulation. The possibility of joining the continued NMF development with the STEP domain is discussed and some examples of NMF based EXPRESS models are presented.

1. INTRODUCTION

One driving factor behind product model research is that it will give designers direct access to easy and repeated design evaluation. Obviously, cost estimates, bills of materials, and various drawings should be easily generated from product model data, but of equal importance are measures of the dynamical performance of the design at hand. In the AEC field the EEC COMBINE project (phase 1) has demonstrated feasibility of data mapping from an EXPRESS-based data model of a building to a range of established building performance evaluation (BPE) tools [Augenbroe 1993]. Phase 2 of this project seeks to put this technology to use among practitioners in the field. Another industrial sector with considerable activity in both product modelling and simulation is the process industry.

The authors of this paper have for some time worked with new simulation techniques and languages for continuous modular systems. These techniques are applicable to a

large class of static and dynamical simulation problems in, e.g., the building, energy and process industries. One important aspect of this work has been involvement in the definition of a standard format, Neutral Model Format (NMF), for expression of component level simulation models. The purpose of this paper is to investigate the applicability of STEP technology in the continuation of this work.

Currently, *component models* (primitive models) are automatically translated from NMF to the proprietary format of the target simulation environment. For example, an NMF model of an axial fan is used to generate an axial fan class in, e.g. IDA [Sahlin 1991]. The class is then instantiated in the target environment. The instances are furnished with suitable parameters, and incorporated into a system model. The next natural step in the NMF development is to formulate an environment independent way of expressing and communicating instantiated *system models* as well. Several authors have already suggested and even implemented such NMF extensions [Kolsaker 1994a, Lorenz 1994]. Since object oriented simulation is a highly relevant topic for product modelling efforts [Augenbroe 1991], we will analyze the implications of using EXPRESS for data modelling of NMF instantiated system models.

In the next two sections a brief overview is given of current work on so called object oriented simulation methods, mainly in the context of building simulation, and of the Neutral Model Format.

2. OBJECT ORIENTED SIMULATION ENVIRONMENTS

The term object oriented is perhaps not the best descriptor for these tools but it has nevertheless become widely used and we will use it here as well. The object orientation concerns mainly the modularity of the physical systems that are being modelled and not so much software techniques. Naturally, most recent developments also use object oriented programming to varying degrees.

2.1 PHYSICAL SYSTEMS AND MATHEMATICAL MODELS

Physical systems that we aim to simulate are modular in nature, i.e. they naturally decompose into subsystems. Frequently, identical subsystems are repeated a number of times in a model, a fact that is taken advantage of in many tools. Furthermore, the systems should have a basically continuous behavior, meaning that equations used to describe them, as well as forcing functions, will have a limited number of discontinuities. Purely event driven systems are excluded.

Models may be expressed in several ways. Bond graphs, linear graphs, block diagrams, electrical analogies, and mathematical equations are frequently used modes of expression. Also used, for mainly historical reasons, are subroutines in some programming language. A discussion of pros and cons of various methods of description can be found in [Lorenz 1987].

If characterized by equations, the physical systems under consideration will require both algebraic and differential equations. Differential equations can be either ordinary (ODE) or partial (PDE), although current tools require that PDEs are explicitly discretized in space and thus turned into ODEs. Note that in contrast to many widely used commercial tools the simulation environments we are concerned with here are

not limited to ODEs only. They allow a free mixture of algebraic and ordinary differential equations generally referred to as differential-algebraic systems of equations (DAE).

Furthermore, the simulation tools under discussion are rarely used for applications where a strict formalism for generating governing equations exists. In, e.g., electrical circuit analysis, multibody mechanics, or structural analysis special purpose systems may be more advantageous.

Examples of physical systems that fit this description can be found in many fields. Chemical process plant simulation is a significant area of application. Energy distribution networks and plants is another. The authors of this paper have mainly worked with building related systems and important applications within this field are: thermal processes in walls and spaces; air and water based distribution systems and plants; and automatic control.

2.2 SEPARATION OF MODELLING AND SOLVING ACTIVITIES

In contrast to many established design tools, e.g. in building simulation, OOSEs separate strictly between the modelling and subsequent system solution activities. A modelling tool is often used for model formulation. This tool generates a system model, generally expressed in a *modelling language*. The model is then treated by a solver. An important benefit of a separate solver is that it may be altered or even exchanged with minimal interference with the modelling environment.

Key characteristics of the modelling language, such as expressiveness and level of standardization, are critical to the usefulness and development potential of the overall OOSE. The Neutral Model Format is part of such a modelling language. This paper describes one way towards a complete modelling language that may be standardized.

2.3 TARGET USERS AND SOFTWARE STRUCTURE

Most of the simulation tools under discussion are intended for quite sophisticated users, who are well versed in mathematical modelling, numerical methods and advanced use of computers. These tools are not directly suited for designers, without special simulation expertise, that use simulation as one of several methods for design evaluation. However, for the expert, they generally provide an efficient graphical environment for model building, simulation and analysis.

Other tools, e.g. EKS and IDA, are primarily intended for efficient design tool production, and the normal end user will rarely interact directly with the underlying OOSE techniques.

2.4 AVAILABLE AND EMERGING OOSEs

A few tools and environments with the discussed main characteristics are already matured and available and others are under development. Among the available ones are e.g.:

TRNSYS was developed during the seventies at the Solar Energy Lab at the University of Wisconsin. It was one of the first modular simulation solvers for DAEs and it is

distributed as a Public Domain product. Several compatible modelling tools have been developed, e.g. PRESIM.

HVACSIM+ is a solver with similar characteristics as TRNSYS in terms of model format and structure, but more recent numerical techniques are utilized. It was developed by NIST in Maryland and released in the mid eighties on a Public Domain basis.

SANDYS is a general DAE solver and textual modelling environment developed by ASEA, Sweden, in the early eighties. It is commercially available from ABB Corporate Research.

ALLAN-NEPTUNIX is a graphical modeller and solver combination developed by Gaz de France and CISI Engineering. It is since a few years commercially available from the developers.

ESACAP is a recently developed DAE solver by the European Space Agency. It is commercially available from STANSIM, Denmark.

DYMOLA is a text based commercial modelling tool with symbolic algebra capabilities and interfaces to several solvers. A GUI is under development. Available from DYNASIM, Lund, Sweden.

Some tools under development are:

CLIM 2000, a graphical modelling tool for building applications, is developed by Electricite de France.

MS1 is a graphical multi input language modeller with interfaces to several solvers by Lorenz Consulting, Liege, Belgium in cooperation with Electricite de France.

IDA, a graphical modelling environment and solver, is under development at the Swedish Institute of Applied Mathematics.

SPARK is a solver and graphical model editor under development at LBL, Berkeley, California.

OMSIM is a graphical modelling tool under development at the Dept. of Automatic Control at the Lund Institute of Technology, Sweden.

EKS is a C++ toolkit for development of energy related simulation design tools, by among others the Univ. of Strathclyde, Scotland.

3. THE NEUTRAL MODEL FORMAT

Without a comprehensive, validated library of ready made component models in a relevant application area most simulation environments are rather useless. To develop all necessary models from scratch is, in most projects, quite unrealistic. And since the cost of developing a substantial library easily exceeds the development cost of the simulation tool itself, it is important to be able to reuse what other people already have done. This was the basic motivation for proposing a text based neutral model

format to the building simulation community in 1989 [Sahlin and Sowell 1989]. Since then the proposal has attracted a great deal of interest from environment developers and users in several application fields. Prototype translators have been developed for IDA [Kolsaker 1994a], SPARK [Nataf 1994] and ESACAP [Pelletret 1994a]. Translator development projects have been funded for TRNSYS, HVACSIM+ [ASHRAE 1994], and MS1 [Lorenz 1994]. Export and import capabilities are planned and partly implemented for ALLAN-NEPTUNIX [Jeandel 1994].

Pending formal standardization, ASHRAE (American Society of Heating, Refrigerating, and Air-conditioning Engineers) has formed an ad hoc committee that approves changes to the present format.

NMF has two main objectives: (1) models can be automatically translated into the local representation of several simulation environments, i.e. the format is program *neutral* and machine readable; and (2) models should be easy to understand and express for non-experts. The first objective enables development of common model libraries, which can be accessed from a number of simulation environments.

3.1 BASIC NMF FEATURES

Internal component model behavior is described by a combination of algebraic and ordinary differential equations. Equations may be written in any order and in the form

<expression> = <expression>;

NMF only *states* equation models, while *solution* of equations is, in some cases, left to the target environment (e.g. IDA, or SPARK), or the NMF translator in others (e.g. TRNSYS, or HVACSIM+).

NMF supports model encapsulation through a link concept, i.e. models may only interact via variables appearing in LINK statements. To enhance and encourage model plug compatibility, links and variables are globally typed. The idea is that basic list of such types should be included in each revision of the standard, but that users may add to the list as need arise. A selection of such global types is:

```
QUANTITY_TYPES

/* type name      unit              kind */

   Area           "m2"              CROSS
   Control        "dimless"         CROSS
   Density        "kg/m3"           CROSS
   Factor         "dimless"         CROSS
   HeatCap        "J/(K)"           CROSS
   HeatCapA       "J/(K m2)"        CROSS
   HeatCapM       "J/(kg K)"        CROSS
   HeatCond       "W/(K)"           THRU
   HeatFlux       "W"               THRU
   HeatFlux_k     "kW"              THRU
   Temp           "Deg-C"           CROSS

LINK_TYPES

/* type name       variable types... */

/* generic        (arbitrary, arbitrary,...) implicitly
defined */
   F              (Force)
   FL             (Force,Length)
   Q              (HeatFlux)
   T              (Temp)
   PMT            (Pressure, MassFlow, Temp)
   PMTQ           (Pressure, MassFlow, Temp, HeatFlux)

   MoistAir       (Pressure, MassFlow, Temp, HumRatio)
   BidirFlow      (Pressure, MassFlow, Enthalpy, HeatFlux)
```

A quantity type includes a physical unit and information about potential (across) or flow (through) type. A link type is simply an ordered list of quantity types. Let us now look at an example of a rather simple NMF model using the heat equation in one dimension.

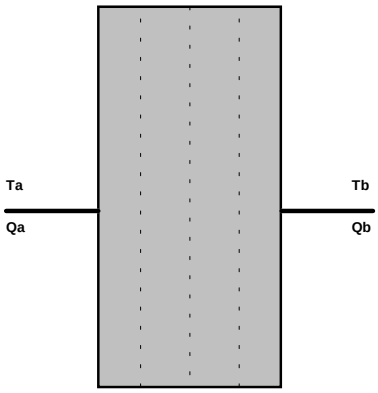


Figure 1. A finite difference model of a wall with one homogeneous layer. Temperature and heatflux on each terminal.

```
CONTINUOUS_MODEL tq_hom_wall

ABSTRACT
"A 1D finite difference wall model. One homogeneous layer.
TQ interfaces on both sides."

EQUATIONS
/* space discretized heat equation */
c_coeff * T'[1] = Taa - 2.*T[1] + T[2] ;
c_coeff * T'[n] = T[n - 1] - 2. * T[n] + Tbb ;

FOR i = 2, (n -1)
c_coeff * T'[i] = T[i - 1] - 2. * T[i] + T[i + 1];
END_FOR ;
/* boundary equations */
0 = -Ta + .5 * (Taa + T[1]) ;
0 = -Tb + .5 * (T[n] + Tbb) ;
0 = -Qa + d_coeff * (Taa - T[1]) ;
0 = -Qb + d_coeff * (Tbb - T[n]) ;

LINKS
/*  type           name      variables ....  */
    TQ             a_side    Ta, POS_IN Qa ;
    TQ             b_side    Tb, POS_IN Qb ;

VARIABLES
/* type    name        role def     min    max      description*/
  Temp     T[n]        OUT   20. abs_zero  BIG  "temperature profile"
  Temp     Ta          OUT   20. abs_zero  BIG  "a-side surface temp"
  Temp     Tb          OUT   20. abs_zero  BIG  "b-side surface temp"
  Temp     Taa         OUT   20. abs_zero  BIG  "a-side virtual temp"
  Temp     Tbb         OUT   20. abs_zero  BIG  "b-side virtual temp"
  HeatFlux Qa          IN     0.    -BIG   BIG  "a-side entering heat"
  HeatFlux Qb          IN     0.    -BIG   BIG  "b-side entering heat"

MODEL_PARAMETERS
/* type    name       role  def mi  max        description  */
  INT      n          SMP    3   3  BIGINT "number of temp layers"

PARAMETERS
/*   type    name       role   [def   [min   max]]   description*/

/* supplied parameters */
  Area      a            S_P    10.  SMALL BIG  "wall area"
  Length    thick        S_P   .2    SMALL BIG  "wall total thickness"
  HeatCondL lambda       S_P   0.5   SMALL BIG  "heat transfer coeff"
  Density   rho          S_P   2000  SMALL BIG  "wall density"
  HeatCapM  cp           S_P   900.  SMALL BIG  "wall heat capacity"

/* computed parameters */
  generic   d_coeff      C_P              "lambda*a/dx"
  Length    dx           C_P              "layer thickness"
  generic   c_coeff      C_P              "rho*cp*dx*dx/(lambda*3600.)"


PARAMETER_PROCESSING
dx := thick / n ;
c_coeff := rho * cp * dx * dx / (lambda * 3600.) ;
d_coeff := lambda * a * dx ;

END_MODEL
```

To enable direct model translation to input-output oriented environments (e.g. TRNSYS, or HVACSIM+), variable declarations have a role attribute indicating IN for given variables and OUT for calculated ones.

Variables and parameters may be vectors or matrices. A parameter is anything that must remain constant throughout every simulation. Links may also be vectors, thus allowing models with variable number of ports. Vector and matrix dimensions are governed by a special type of parameter, model parameters. Regular and model

parameters are divided into two categories, user supplied and computed, algorithmic computation of which is described in the parameter processing section.

Arbitrary foreign functions in Fortran 77 or C may be defined, either globally or locally within a model.

Special functions are defined to handle discontinuities, hysteresis, linearization, and errors. A more complete account of NMF is given in the reference report [Sahlin, Bring, and Sowell 1994].

3.2 NMF DEVELOPMENT DIRECTIONS

Currently, a reasonable agreement about the NMF grammar has been reached. Developers can count on stability of the present format and backward compatibility. This enables us to get on with the work of defining NMF-based component model libraries and to develop further NMF translators. Several substantial model libraries have already been developed and many more are underway.

Regarding the format itself, several extensions have been suggested. In the discussion of these it is important to bear in mind that, at the time of the original proposal, NMF was not primarily intended as a replacement of existing proprietary model languages, but as a complement, enabling component model exchange and library building.

Planned extensions and supporting tools that fall within the scope of the original NMF intentions are:

1. An NMF handbook with style guidelines for model architecture. The current NMF manual is completely insufficient as a pedagogical tool. (Encompassed by funded project [ASHRAE 1994].)

2. Model documentation guidelines and templates, storage and retrieval mechanisms. This area is addressed by Pelletret in a recent (draft) proposal [Pelletret 1994]. The ESPRIT OLMECO project - development of a large mechatronics library - is another source of inspiration.

3. Investigation regarding adaptable models, through property inheritance and/or through hierarchical modelling. Property inheritance between models may result in better model reuse but it will on the other hand also have negative effects on model portability, since inheritance trees must be passed when shipping a model. This leads to reconciliation problems if a similar, but not identical, tree exists on the receiving side.

4. Model library structure and management tools, including mechanisms for model browsing and retrieval.

5. Discrete time (sampling) models. This is necessary to study sampling control circuits.

There are several additional items that belong in this list - such as formal rules for permitted model connections and a language or keyword system for expression of model assumptions - that are omitted here due to space.

In the context of a complete modelling language the present format lacks the ability to express:

1. Component model instances, with parameter values, initial values of all variables, and information about boundary variables..

2. Hierarchical systems of such instances.

3. Numerical simulation parameters, such as tolerances, stepsize limits, algorithm selection commands, that can be generalized for a large class of solvers.

4. Graphical schemata for user presentation of simulation models. Large models are much easier to comprehend if they are described graphically.

The drive for development of a complete NMF-based modelling language comes primarily from developers of new modelling tools, who see little reason to develop proprietary formats. Two such developers have made concrete proposals and implementations are well underway [Lorenz 1990], [Kolsaker 1994].

4. WHY STEP/EXPRESS?

STEP (STandard for the Exchange of Product model data) is an international standard for product descriptions [ISO TC 184 1993]. The data for these descriptions are modelled in a special language called EXPRESS, which is in itself part of the STEP standard. EXPRESS is an object oriented language that is particularly well suited for information modelling. A subset of EXPRESS is EXPRESS-G, a fully graphical language for data modelling. EXPRESS-G schemata can automatically be translated into textual EXPRESS code, which in turn can be translated into, e.g., C++ class definitions. A number of tools and related standards are (and will be) available for STEP/EXPRESS. A (default) textual representation of any EXPRESS schema is for example implicitly defined (STEP physical file).

Since the first proposal in 1989 the discussion about various NMF-constructs has focused on the grammar. The textual appearance of selected models has been the main object. This is of course quite appropriate for the equation core of component models, but for instantiated system models and related data it may be more fruitful to regard data models directly, and to treat textual representation as one of several possible views. EXPRESS seems to be an appropriate vehicle for the future NMF discussion. Further reasons for the employment of STEP technology include:

- Simulation models will most likely be an important aspect of many product model applications, and they should therefore be encompassed by STEP, either as pure aspect models or as parts of global models

- Many existing STEP/EXPRESS resources will be useful for development of NMF-oriented application tools

- The fact that STEP physical files most likely will be more difficult to read (for humans) than a tailored high level language is of little consequence for realistic-size simulation models, which generally are of such magnitude that they rarely are

printed and studied in their raw form

4.1 PRESENT NMF IN STEP

In this our initial work we have chosen to focus directly on the imminent problem of defining conceptual models of NMF instances, and of hierarchical systems of such instances. This means that nothing is said about the internal behavior, e.g. equations, of a model. Only its state is encompassed, and it is assumed that the underlying NMF model is known to all parties.

Another interesting issue is of course the conceptual models of internal behavior as well, i.e. to model the present NMF in EXPRESS, with entities such as equation, if_then_else_clause, etc. Such models are necessary for development of NMF parsers and translators.

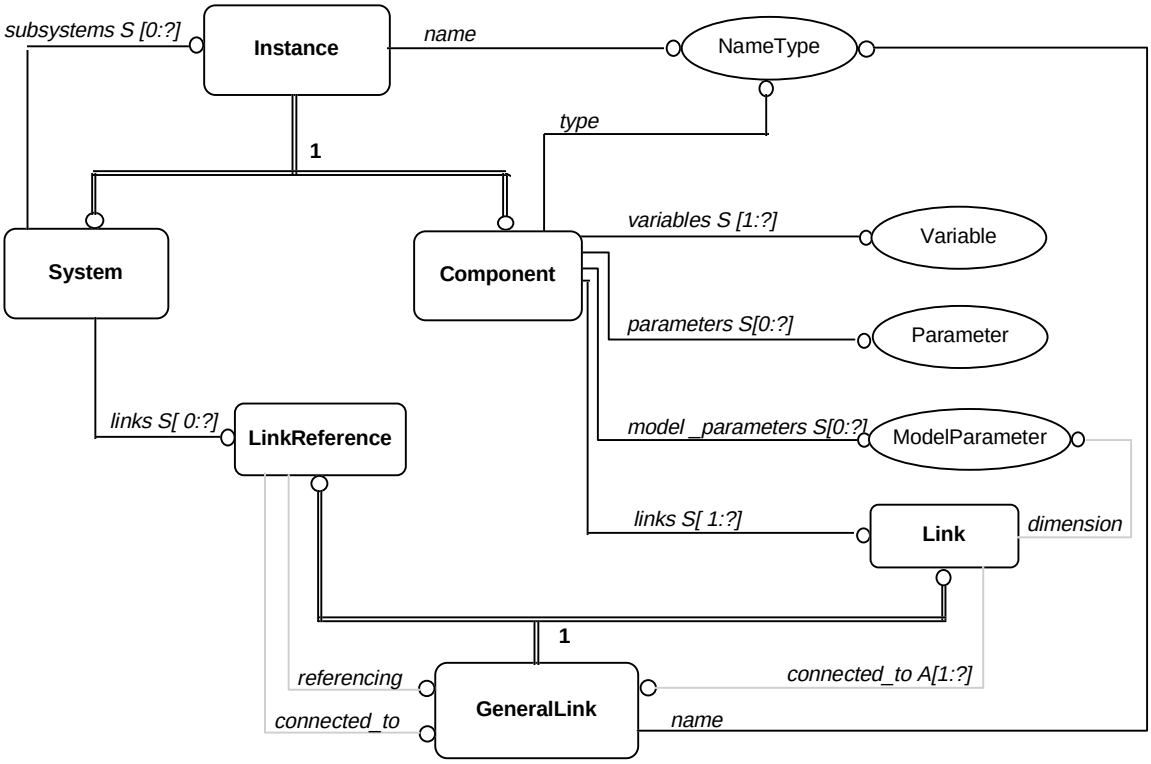
The main motivation for remodelling the present NMF in EXPRESS is completeness. New component models could be communicated with the same tools and protocols. A potential EXPRESS-based STEP standard would not have to rely on an additional non-EXPRESS standard.

Additional benefits can be expected for design and implementation of NMF component model databases and management tools.

The present conclusion is that it would be worthwhile to model the present NMF in EXPRESS. However, since the discussion of instances and systems can be carried out separately, we have chosen to focus on this in our initial work.

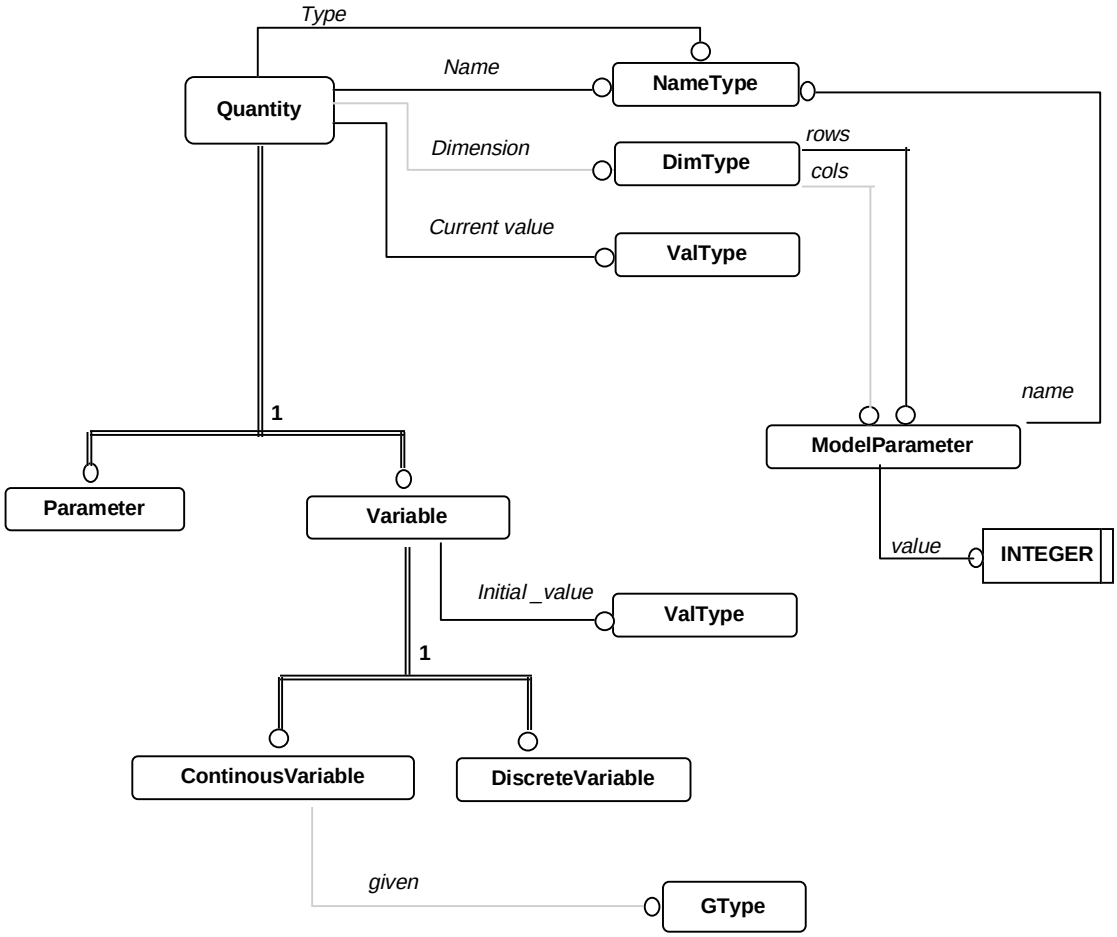
5. NMF MODEL INSTANCES IN EXPRESS-G

In the following an EXPRESS-G representation of NMF component and system model instances is presented. An instance is a specific occurrence of a model expressing the full state, in terms of its parameter values, variable values, and associated data. Schemata 1 through 4 shows the EXPRESS-G representation of this data.



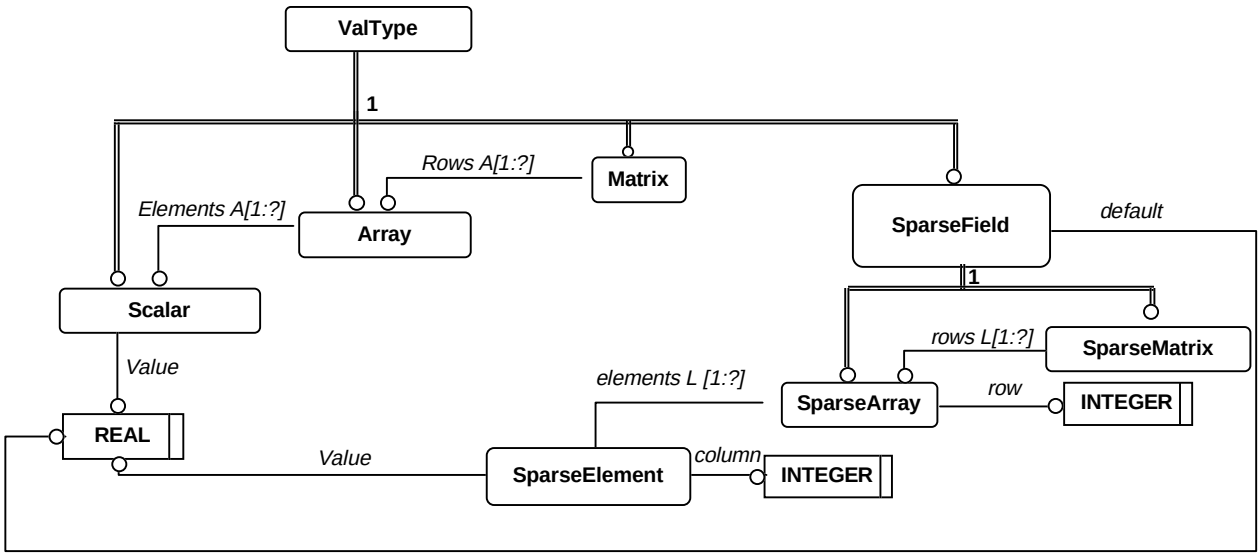
Schema 1

Schema 1 shows the structure of an NMF model instance, which may appear as either a System, with references to underlying subsystems, or as a Component with object bags for variables, parameters, model parameters, and links, each of which is specified more closely in the following schemata . Model parameters are named integers that are used for dimensioning of arrays and matrices. Links are the connection ports of Components. The ports of a System are called LinkReferences. They provide reference chains to underlying Links. The distinction between the quantity subclasses parameters and variables is that parameters always remain fixed at a given value throughout a simulation, while variables, naturally, vary.



Schema 2

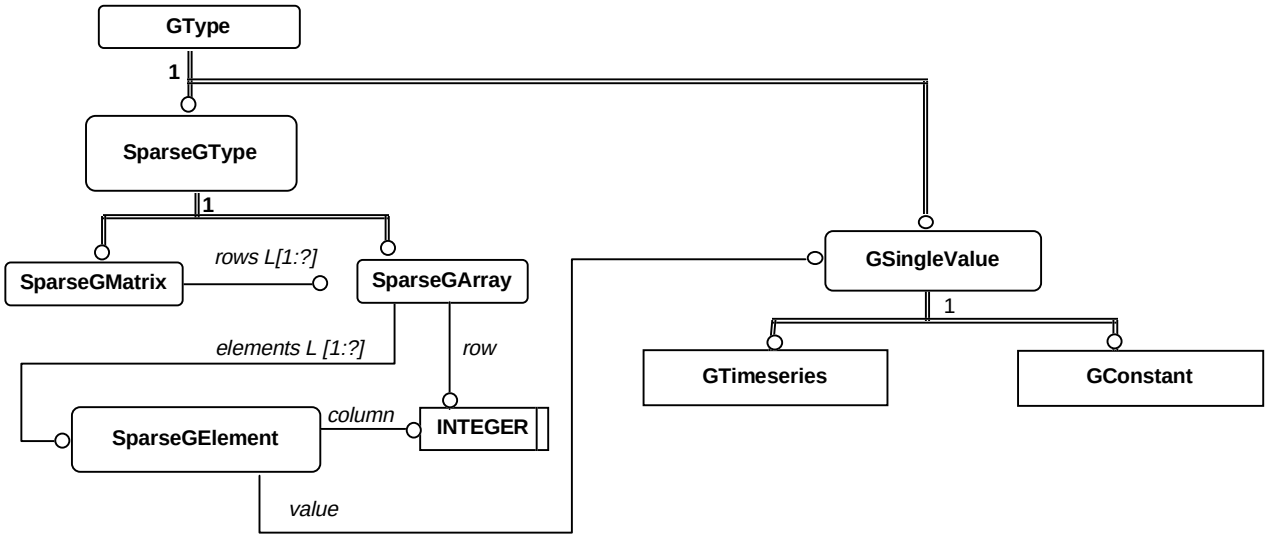
A quantity has a type (the NMF QUANTITY_TYPE referred to), a name, a dimension (if non-scalar), and a current value. Variables also have an initial value, which holds the state at the beginning of a simulation for dynamic (or state) variables and an initial value guess for algebraic variables. Discrete variables is a provision for future development of discrete time NMF models and is not currently used.



Schema 3

Continuous variables also have an optional flag given which, if present, indicates that a variable, or selected parts of a field variable, are to be kept at a given value throughout the simulation.

Schemata 3 and 4 specify storage structures and stored elements for current and initial values (schema 3) and for given flags (schema 4).



Schema 4

Values may be stored in either a sparse matrix storage structure or in full matrices (or ditto arrays). Initial values are generally stored in a sparse structure where only exceptions from the default value are listed.

Since the great majority of variables are calculated, the given flag is stored in a sparse structure as well. The flags themselves are either a reference to a time series of values (not specified in detail) or a GConstant symbol, indicating that the variable is to be kept at its initial value throughout the simulation.

6. CONCLUSIONS AND FUTURE WORK

Our present work suggests that EXPRESS is suitable for modelling of many of the data structures that are relevant for continuous simulation of modular systems. If not incorporated into the STEP effort, a continuous simulation language standardization project could certainly operate in a similar fashion and use many of the same methods and tools.

Next on the agenda will be to test the functionality of the suggested data structures by writing a parser for, initially, IDA system model descriptions.

References

Augenbroe, G, and F. Winkelmann, 1991, Integration of Simulation into the Building Design Process, proc. of Building Simulation ´91 , Nice, France, International Building Performance Simulation Association

Augenbroe, G. (ed), 1993, COMBINE Final Report, CEC-DG XII-JOULE

ASHRAE 1993. Invitation to Submit a Research Proposal on an ASHRAE Research Project: 839-TRP Development of a Component Model Translator for the Neutral Model Format, American Society for Heating, Refrigerating and Air-Conditioning Engineers, Atlanta, GA

Buhl, W.F., E.F. Sowell, and J-M Nataf, 1989. Object-oriented Programming Equation-Based Submodels, and System Reduction in SPANK, proc. of Building Simulation ´89 , Vancouver, BC, International Building Performance Simulation Association

ISO TC 184 1993. The STEP Standard, draft international standard DIS 10303, continuously since 1992 published in several different parts

Jeandel A. 1994, Personal communication

Kolsaker, K. 1994a. NEUTRAN-supported NMF Enhancements, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Kolsaker, K. 1994b. Simpler NMF Description of Advanced Models Using Hierarchical Modelling and Data Abstraction, to be presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994

Lorenz F. 1987, Reflections about Representation Methods, proc. workshop on the future of building energy modelling, Ispra, Italy, Nov. 1987, CEC EUR 11603 EN PREPRINT, May 1988

Lorenz F. 1990. Brief Description of the MS1 (Modelling System 1) Project, private communication

Lorenz F., 1994, Comments on the Neutral model Format, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Nataf J.-M. 1994, Translator from Neutral Model Format to SPARK, draft paper presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Pelletret, R. 1994, Personal communication

Pelletret, R., S. Soubra 1994b. Standardizing Model Documentation - The PROFORMA Experience, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Sahlin, P, E.F. Sowell 1989, A Neutral Format for Building Simulation Models, proc. of Building Simulation ´89 , Vancouver, BC, International Building Performance Simulation Association

Sahlin, P. 1991, IDA - a Modelling and Simulation Environment for Building Applications, Swedish Institute of Applied Mathematics, ITM Report no. 1991:2

Sahlin, P., A. Bring, and E. F. Sowell, 1994. The Neutral Format for Building Simulation, Version 3.01, Swedish Institute of Applied Mathematics, ITM Report no. 1994:2

Sowell, E.F. 1994. A Proposal for Hierarchical Submodels in NMF, to be presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994

coupled air flows and heat transfers in multi-zone buildings. In addition to the coupling issues, this development has a special interest in that it offers an opportunity to weigh the pros and cons of general simulation tools versus application oriented ones, particularly with respect to: efficiency, robustness and development man-time. In this case, the application area is extremely well suited for tailored methods, so the odds against the general tools are high.

IDA has been developed at the Swedish Institute of Applied Mathematics in cooperation with the department of Building Services Engineering at KTH. At the heart of the system is a general solver for differential-algebraic (DAE) systems of equations, IDA Solver, which among other things features input-output free, precompiled component models. A discussion of solver design issues can be found in (Sahlin and Bring 1991) and (Eriksson, Söderlind, and Bring 1992) treats the numerical methods of IDA in some more detail. IDA Modeller is an interactive front end to the solver. IDA component models are primarily described with the Neutral Model Format, NMF (Sahlin, Bring, and Sowell 1992).

The present project has involved use of the entire IDA environment. The programming tool kit of IDA Modeller has been used to complement the general model-lab facilities of IDA. The result is a tailored user interface, focusing on the editing of the air flow networks and on specialised result presentations. These features of the current application, and the Modeller in general, are treated in an accompanying paper (Sahlin 1993). In this paper we will concentrate on NMF modelling of the components and on solver techniques.

The current application has been developed with support from the Swedish ventilation manufacturer, ABB Indoor Climate. Their primary interest is to use it for clean room design. ABB had evaluated the Movecomp program (developed by one of the present authors in cooperation with M. Herrlin), and found the functionality adequate but the user interface slightly dated. The idea to modernise Movecomp was discussed, but, since the resulting product would still be a monolithic program with uncertain development potential, an alternative was sought. The alternative, to implement the Movecomp functionality in IDA and develop a tailored tool, was accepted.

Section 2 covers the basic mathematical modelling, which is concretised further in Section 3 with sample NMF models. Application performance and numerical features are discussed in Sections 4 and 5. Due to space constraints, the account is focused on the multizone air exchange (MAE) models alone. These models have been successfully tested in conjunction with building thermal models, but the main difficulties are all MAE model related. Hence, our focus here.

2. FEATURES MODELLED

The implemented air-exchange models follow closely the approximations made in the Movecomp program (Herrlin 1992). Macroscopic models are used, assuming complete mixing within each zone (node). Obviously, for clean room design, it would be advantageous to study flow patterns within zones, especially in cases when doors are temporarily opened. However, this would require coupling of CFD models to lumped parameter models for ventilation and infiltration components. Such a task, although technically feasible, is outside the scope of our current design tool effort.

Besides the basic pressure - mass balance, we model transports of enthalpy plus one contaminant. Generally, the temperature distribution has a weak influence on the mass transport, whereas the contaminant is considered to lack such coupling. The current set of models treats only simple molecular transport of the contaminant. Sorption phenomena and molecular reactions are not included. It would be straightforward to extend the models to transports of more than one contaminant.

Excluding heat transfer models for the building envelope, we deal with two groups of air flow components; we will call them *nodes* and *connecting elements*.

The node components are characterised by their potentials: pressure (P), temperature (T), contaminant concentration (X). Nodes can be zones, or junctions in the ventilation system. Their model equations represent conservation of mass, energy, and contaminant:

$$\begin{cases} 0 = \sum \dot{m}_i \\ 0 = \sum q_i + q_{zone} \\ 0 = \sum xf_i + xf_{source} \end{cases} \qquad (1)$$

where q_{zone} is heat from the envelope and xf_{source} is a contaminant source term.

The connecting elements can be leaks, or any double-ended pieces of the ductwork (ducts, grilles, fans, etc.). Among their model equations we always find the mass flow modelled as a function of the pressure difference across the element,

$$\dot{m}_{1-2} = f(P_1 - P_2). \qquad (2)$$

We have chosen to use power law equations for the pressure - flow relation, rather than quadratic relations. Both choices have advantages; for us the main factor has been compatibility with Movecomp to simplify comparisons. For a leak, with linearization around zero, we then have,

$$f(\Delta p) = \begin{cases} c_0 \Delta p, & Abs(\Delta p) < \Delta p_0 \\ c(\Delta p)^n, & \Delta p > 0 \\ -c(-\Delta p)^n, & \Delta p < 0 \end{cases} \qquad (3)$$

where *n* is a coefficient between 0.5 and 1.0.

Heat and contaminant transport through a connecting element are typically just convected by the mass flow,

$$q_{1-2} = \begin{cases} c_p T_1 \dot{m}_{1-2}, & \dot{m}_{1-2} > 0 \\ c_p T_2 \dot{m}_{1-2}, & \dot{m}_{1-2} < 0 \end{cases} \text{ and} \qquad (4)$$

$$xf_{1-2} = \begin{cases} X_1 \dot{m}_{1-2}, & \dot{m}_{1-2} > 0 \\ X_2 \dot{m}_{1-2}, & \dot{m}_{1-2} < 0 \end{cases}, \qquad (5)$$

where *T* and *X* are the temperatures and contamination concentrations of the connected nodes on each side of the leak.

Heating or cooling coils will also model a local heat balance; a filter will model a reduction in contaminant concentration.

One of the major advantages with the modular approach is that replacement of individual component models is easy as long as the interface variables between models remain the same. This makes it possible to replace these simple models by more detailed ones. Thus, we could for instance introduce heat transports between ducts and surroundings, or detailed models of sorption phenomena. So far no such models have been implemented.

3. NMF AND EXAMPLES

In this section we will explain some basic features of the Neutral Model Format (NMF), and present NMF code for two representative models.

3.1. The Neutral Model Format (NMF)

NMF is a suggested standard for model expression. It has two main objectives: (1) models can be *automatically translated* into several simulation environments, i.e. the format is program *neutral* and machine readable; and (2) models should be easy to understand and express for non-experts. The first objective enables development of common model libraries, which can be accessed from a number of simulation environments.

Internal component model behaviour is described by a combination of algebraic and ordinary differential equations. (The given examples have only algebraic equations.) Equations may be written in any order and in the form

<expression> = <expression>;

NMF only *states* equation models, while *solution* of equations is, in some cases, left to the target environment (e.g. IDA, or SPARK), or the NMF translator in others (e.g. TRNSYS, or HVACSIM+).

NMF supports model encapsulation through a link concept, i.e. models may only interact via variables appearing in LINK statements. To enhance and encourage model plug compatibility, links and variables are globally typed.

To enable direct model translation to input-output oriented environments (e.g. TRNSYS, or HVACSIM+), variable declarations have a role attribute indicating IN for given variables and OUT for calculated ones. In addition, pure help variables, with role LOC, may be *assigned* to.

A complete account of NMF is given in (Sahlin, Bring, and Sowell 1992).

3.2. Zone Model Example

```
CONTINUOUS_MODEL BdZone
ABSTRACT
 "A static zone model for air-exchange modelling. Bidirectional
transports of energy plus a mass fraction are modelled."
EQUATIONS
/* mass conservation (eqn. 1a)*/
  0 = M_0 + SUM i=1, n M[i] END_SUM;
/* energy conservation (eqn 1b) */
.  0 = Q_zone + Q_0 + SUM i2=1, n Q[i2] END_SUM;
/* fraction conservation (eqn. 1c)*/
  0 = xf_source + Xf_0 + SUM i3=1, n Xf[i3] END_SUM;
LINKS
/* type    name       variables... */
BidirX terminal_0 P, POS_IN M_0, T, POS_IN Q_0, X, POS_IN Xf_0;
FOR i = 1, n
  BidirX terminal[i]   P, POS_IN M[i], T, POS_IN Q[i], X, POS_IN Xf[i];
Tq      air_temp      T, POS_IN Q_zone;
VARIABLES
  /* type          name    role [def   min  max]  description */
  MassFlow_u  M_0     OUT          "terminal 0 mass flow"
  MassFlow_u  M[n]    IN           "terminal i mass flow"
  Pressure    P       IN           "zone floor level pressure"
  HeatFlux    Q_0     OUT          "terminal 0 heatflux"
  HeatFlux    Q[n]    IN           "terminal i heat flux"
  Temp        T       IN           "zone temperature"
  FractFlow_u  Xf_0   OUT          "terminal 0 transport"
  FractFlow_u  Xf[n]  IN           "terminal i transport"
  Fraction_y  X       IN           "zone fraction"
  HeatFlux    Q_zone IN            "heat gain/loss in zone"
MODEL_PARAMETERS
/* type   name     min  max          description */
  INT       n         1   BIGINT    "Number of links minus one"
PARAMETERS
/*type       name  [def  min  max]   description */
Length         za                   "zone floor height relative to ground"
Length         h   2.4  SMALL  BIG      "zone height"
Area           a   10  SMALL  BIG       "zone floor area"
FractFlow_u xf_source               "Mass fraction source (or sink)"
END_MODEL
```

NMF code for the air-exchange zone model

The link type for bidirectional flow between models BidirX has six variables: pressure and mass flow, temperature and heat flow, contaminant concentration and contaminant flow. It might seem that either temperature or heat flow could be eliminated, and corresponding for the contaminant pair of variables. It turns out, however, that modelling of bidirectional flow in a modular framework precludes that simplification. A discussion of this topic is outside the scope of this presentation, but it centers around the need to have a fixed number of equations for any component, independent of flow directions (contemplate the change when a T-piece swaps from converging to diverging state).

Besides the BidirX links, which handle air-exchange interaction with neighbouring models, the zone model has a Tq (temperature, heat flux) interface. This interface is the bridge between the air-exchange family of models and the thermal models for the building envelope. Due to space constraints we will refrain from a presentation of these models here.

A special case of scaling is used for time related variables, i.e. flows and time derivatives. These occur related to different time units, chosen to fit the current simulation focus. E.g. thermal building models are typically used with hour as time unit but may also be combined with flow models and control models in order to study faster phenomena at seconds scale. To avoid trivial model duplications, the scaling differences can be expressed in the models by using a time unit specified by a parameter t_scale. This scale gives the length in seconds of the time unit, t_scale = 3600 for hours, etc.

The zone and leak models have the BidirX interface which allows bi-directional flow. However, in the interest of calculation time we have chosen to model only uni-directional flow in ventilation components. This means that ventilation models need only four link variables, rather than six for the BidirX links and, generally, only one equation (eqn. 2), instead of three. The four-variable link type is called VentX.

3.3. Zone Supply Terminal Model

The supply terminal model, VxSupT, acts as interface between the VentX and BidirX models. Consequently, it has one link of each kind. This model still needs three equations - although only uni-directional flow is allowed - since there is a BidirX interface present.

```
CONTINUOUS_MODEL VxSupT
ABSTRACT "Supply Terminal. Linear flow below limit 'dp0', and if LIN."
EQUATIONS
/*two help assignments, for local density and pressure drop */
  Rho := rho_20 * (20. - ABS_ZERO) / (T2 - ABS_ZERO) ;
  Dp := P1 - P2 + zr2 * G * Rho ;
/* power law mass flow equation,  eqn. (2) in paper */
/* LINEARIZE function explained in Section 5.1 of paper */
  0 = -M / t_scale +
          IF LINEARIZE (1) THEN c_turb * Dp
          ELSE_IF Dp < dp0 THEN c_lin * Dp
          ELSE c_turb * sqrt (Dp)
          END_IF ;
/* convected heat through terminal, eqn. (4) in paper */
  Q = IF LINEARIZE (1) THEN T1
       ELSE cp * T1 * M / t_scale
       END_IF ;
/* fraction transported through terminal, eqn. (5) in paper */
  Xf = IF LINEARIZE (1) THEN X1
      ELSE X1 * M
      END_IF ;
LINKS
  /* type     name       variables... */
  VentX  inlet    P1, POS_IN M, T1, X1 ;
  BidirX  zone   P2, POS_OUT M, T2, POS_OUT Q, X2, POS_OUT Xf;
VARIABLES
/* type        name   role  [def   min  max]      description */
  MassFlow_u  M     OUT   0.  -BIG   BIG      "mass flow "
  Pressure      P1    IN     2.   SMALL  BIG   "pressure in"
  Pressure      P2    IN     1.   SMALL  BIG   "pressure out"
  temp          T1    IN     15. ABS_ZERO BIG "temperature in"
  temp          T2    IN     15. ABS_ZERO BIG "temperature zone"
  HeatFlux      Q     OUT   0.  -BIG   BIG  "heat convected by massflow"
  fraction_y    X1     IN    .1   0.       BIG   "pollutant fraction in"
  fraction_y    X2     IN    .1   0.       BIG   "pollutant fraction zone"
  FractFlow_u Xf      OUT   0.  -BIG   BIG   "pollution transport"
  Pressure      Dp    LOC                  "eff pressure diff"
  density       Rho   LOC                  "air density"
PARAMETERS
  /* type       name  [def    min    max]  description */
/* easy access parameters */
/* priority order: c_t, xi */
  generic       c_t      0.   0          BIG   "power law coefficient"
  length        d        .25  SMALL  BIG   "inner diameter"
  generic       xi       10.   0         BIG   "loss coefficient"
  length        zr2       0.   0          BIG   "leak height from floor"
/* globally given */
  HeatCapM   cp    1006   500  3000      "air cp"
  Pressure    dp0    .1     SMALL  BIG    "limit for linear flow"
  Density     rho_20 1.2  SMALL  BIG     "reference density"
```

```
  Factor       t_scale 1.   SMALL  BIG     "size of time unit [s]"
/* derived parameters */
  area          a                          "cross section area"
  generic       c_lin                       "laminar coefficient"
  generic       c_turb                       "flow characteristic"
PARAMETER_PROCESSING
/*par processing is executed once, prior to simulation */
  a := PI * d * d / 4. ;
/* Check alternative definitions of C_turb */
  c_turb := IF c_t != 0. THEN
              c_t
            ELSE_IF xi == 0. THEN
              0.
            ELSE
              a * (2. * rho_20 / xi)**0.5
            END_IF ;
  IF c_turb == 0 THEN
  CALL nmf_error ("wrong parameters (c_t or xi) for supply terminal");
  END_IF ;
  c_lin := c_turb / sqrt (dp0) ;
END_MODEL
```

The use of the LINEARIZE function enhances model robustness. This issue is treated at some depth in Section 5.1.

4. PROBLEM SIZE

A crucial issue in our case of general versus specialised programs is obviously the question of overall efficiency.

In tailored programs for multi-zone air exchange, it is natural to reduce the central pressure - mass flow equation system (eqns 1 and 2) to contain just the flow balances in the nodes. The equations for the connecting elements relating pressure and flow (eqn. 2) are used to calculate the Jacobian matrix of the system with regard to the node pressures. The resulting linear system is positive definite and can therefore be solved without pivoting. Newton iteration, often damped, is used to solve the nonlinear system. The weak coupling from temperature to pressure, due to stack effect, can be handled in the same iteration loop without untoward effect on the convergence properties. The contaminant distribution, which normally has no feedback on pressure, can be calculated separately once the mass flows are known.

In a general simulation environment, such as IDA, the above type of simplification is not readily available. One of the aims of the current study has also been to demonstrate the applicability of a general tool on this type of problem. Thus, no effort has been made to utilise the special structure of the problem. The general sparsity techniques implemented in IDA are of course used. These are, on the other hand, most effective for problems with large components (many equations) but relatively few connections between components, and are thus not particularly well suited for the current model family.

In IDA, the equation system that is simultaneously solved will contain, both the conservation equations from the nodes, the pressure - flow relations from the connecting elements, and all transport equations. The system matrix will thus often be more than an order of magnitude greater than in the tailored program, especially when a ventilation system with many components is included.

At first sight this growth in system size seems, of course, fatal. However, for an overall appraisal of the approach, one should weigh the time spent on modelling, i.e. connecting models and giving parameters, against the raw simulation (solution) time. When IDA Modeller is used to set up a simulation problem, a reasonably sized problem may take about an hour to assemble, provided all parameters are known in advance. Let's say a problem with a three story building, five zones on each floor, and a balanced mechanical ventilation system. The time spent on actual number crunching on such a problem, with the general approach, is about half a minute on a 486 system. Obviously, the calculation time of a tailored program is only a few seconds, but this is, in our view, of little practical consequence, since the total turnaround time will be virtually the same. If a large number of simulations have to be done with the same model - for, say, automatic optimisation - the performance difference naturally becomes more important.

5. NUMERICAL APPROACH

In the previous section we have discussed the efficiency of the general tools in comparison with specialised. Naturally, the other crucial performance factor is robustness. If a general approach is significantly less reliable, it will clearly not be useful as a base for end user tools. Again, the multizone air-exchange set of models is extremely demanding.

Looking again at the tailored programs, we find that the restricted problem type can be exploited to enhance efficiency. Movecomp, e.g., uses a two step strategy: First, the flow equations are linearised. The exponents in the power law equations are set to one, and this linear system is solved. Secondly, starting from the linear solution, a modified newton method is used; in each step a Jacobian is calculated and a line search is made in the direction defined by a newton step. A formal proof has been found for the universal convergence of this method (Lindberg 1985) for the case of power law leak models. This is, indeed, quite remarkable since few such proofs exist for severely nonlinear systems.

For general DAE simulation, a separate initial value calculation is required to find start values satisfying the algebraic equations. For nonlinear systems, this task is a crucial problem, and will remain to be so,

since no general solution is possible. Independent of problem type, the task can be solved if the user supplies good enough first guesses. Obviously, this can be very difficult for complicated models. Several general methods are available in IDA to deal with this problem:

- explicit linearization of NMF models to get a user independent starting point for nonlinear equation solving;

- two Newton homotopy techniques plus a line search in the Newton direction;

- a gradient solution method has been implemented and is under testing for cases when user guesses result in a singular Jacobian matrix.

The present version of the air-exchange library of models is completely static (algebraic), i.e. the initial value problem is the *whole* problem. However, for coupled models, e.g. when thermal models are included, we have a true DAE problem, but the initial value difficulties of this problem come from the air-exchange models alone. Hence, it is sufficient to look at them alone for experimentation with initial value techniques.

Two of the initial value tools of IDA have been developed in conjunction with the present project: explicit linearization and line search techniques.

5.1. Linearised NMF Models

Linearization of nonlinear models can be a useful means to support initial value calculation, quite independent of application field. We have thus chosen to make the linearization an explicit feature of the NMF models. This has been a planned extension of the NMF syntax for some time, but the present models are the first, where such a feature is indispensable. NMF-translators for various simulation environments may implement different interpretations, the simplest being to ignore the construction by implementing a dummy LINEARIZE function. This type of extension is however of general interest and we report the experience of our experiments so far.

The IDA solver is prepared to pass through a sequence of one or more preparatory stages at the beginning of each initial value calculation. Typically, there is only one extra stage, during which some components may choose to linearise their models. The stage information is requested by the component via a call of a Boolean system function

LINEARIZE (n),

where n is an integer constant. The function definition is:

LINEARIZE is true if the stage number of the solver is less than or equal to n.

The solver will let the stage number vary 1, 2, ... until no call of LINEARIZE gets an answer true. At that stage, all linearizations have been removed, and, if the solving has converged so far, one set of initial values has been found.

In the current application, all power law equations governing mass flow have been equipped with a linearised alternative, e.g. the leak mass balance (eqn. 2 and 3) is:

```
/*Leak with bidirectional flow */

/* power law mass flow equation */
  M / t_scale =
        IF LINEARIZE(1) THEN c * Dp
        ELSE_IF abs (Dp) < dp0 THEN c0 * Dp
        ELSE_IF Dp > 0 THEN c * Dp**n
        ELSE -c * (-Dp)**n
        END_IF ;
```

The equations governing heat flow and contaminant flow, being without major influence on the mass transport, can be manhandled even more. In a converging T-piece we might define the heat balance through:

```
/*Converging T-piece*/

/* energy balance equation */
0 = IF LINEARIZE (1) THEN
                    - T3 + T1 + T2
                ELSE
                    - M3 * T3 + M1 * T1 + M2 * T2
                END_IF ;
```

It is conceivable that a two stage relaxation of linearizations might be useful for this type of components. It could for instance help to keep temperature and contaminant equations linearised while the nonlinear mass flow equations are introduced. So far, this has not been called for in the current application.

For the air-exchange models the linearization technique relieves the user completely of guessing initial values. This is, in our view, an essential factor in the case of general vs. special.

5.2. Line Search Technique

Ordinary Newton-Raphson iteration is not a reliable tool for initial value calculation in nonlinear applications. Damped Newton-Raphson, is a better alternative, especially if the dampening factor is dynamically selected, e.g. by a line search. One such method has been implemented in conjunction with the current application.

A search is made in the direction defined by a newton step. The location of a minimum for a residual function is estimated, using quadratic interpolation between three suitable points. The minimised function is a weighted Euclidean norm of the residuals in the model equations:

$$\min_{\lambda} \sum_{i} (w_i * r(x_0 + \lambda * dx)_i)^2 \quad , \qquad (6)$$

where **r** is the residual vector and **dx** defines the newton direction. **w** is a weight vector introduced to compensate for variations in scaling between the equations 0 = **F**(**x**):

$$w_i = 1 / \max_k \left(\left| \frac{\partial F_i}{\partial x_k} \right| \right) \qquad (7)$$

The mentioned explicit linearization and line search techniques have resulted in sufficient convergence properties for all practical purposes. Occasionally, for cases without any driving forces, the linearized solution leads to a singular Jacobian. However, these problems are likely to be solved with the recently implemented gradient methods.

5.3. Utilising Sparsity

A general problem formulation often leads to sparsely populated equation systems. The MAE models are, as we have seen, no exception in this respect. Consequently, a lot of the work on solvers for general simulation environments deals with various methods to utilise this sparsity with little or no loss of generality. IDA Solver provides several methods for various problem types, but many more could be developed. The numerical methods of IDA are presented at some depth in (Eriksson et al 1992). Here we will be content with a very superficial discussion.

The most sophisticated IDA methods, the *modular methods*, starts with a very large system of equations, which not only includes all component equations but also equations for every connected pair of variables, e.g.,

$$zone7.m_5 = -leak1.m_{1-2}. \qquad (6)$$

This class of methods is best suited for models with few component interface variables in comparison to internal variables. Another IDA method, the *compact method*, starts with eliminating the connection equations and then proceeds with solving the system without further utilisation of sparsity.

On the mentioned three-story test case, the best modular method outperforms the compact method slightly, in spite of the large number of interface variables.

The most obvious improvement would therefore be to implement some staple sparse techniques within the compact method, e.g. band and skyline sorting and solution. However, we believe a lot more can be done, and that, in fact, the MAE models represent a quite important problem category, i.e. a central mass flow - pressure system (eqns. 1 and 2) and several loosely coupled transport equations (eqns. 4 and 5). Topologically, all processes are spread across the network. Transports in the present models are only a single contaminant fraction and thermal energy, but they could be many more. Hitherto, we have not had the means to work on sparse methods for this class of problems but it seems evident that much could be done.

6. CONCLUSION

An application tool for inter-zone air exchange has been developed, using NMF and the general simulation tool IDA. NMF has proven a very effective means to describe component models, for human readers as well as for automatic translation into a simulation environment. IDA has been used to generate a tailored user interface for the application, making system building and manipulation simple tasks. The IDA Solver has proved to be an adequate tool for solving the nonlinear algebraic systems generated by the current application.

The application tool is presently under evaluation in an industrial design setting at ABB Indoor Climate.

A comparison with an existing simulation program, developed explicitly for inter-zonal air flow studies only (Movecomp), shows that the general approach is competitive:

- The tailored program is faster, but calculation times with the general program are quite acceptable also for fairly large systems, in spite of the facts that the problem type is extremely well suited for specialised solution methods and that the speed-up potential of the general tools is far from exhausted.

- The robustness of the general method is sufficient for practical purposes.

- Development time in the general environment is several times shorter than the specialised ditto.

- Model coupling with ,e.g., thermal models is straightforward in the general case while hardly practical in the specialised.

- The general models have significantly better maintenance and development potential. Adding a new model to the library only involves *formulating* the NMF model; the implementation time is negligible.

In summary, considering the continuous improvement of computer hardware, the weakness of the general system in calculation time is far outweighed by other factors. The same argumentation should be valid for most other building simulation applications, since the MAE models are rather demanding.

REFERENCES

Bring, A; Herrlin, M. 1991. Bris Data AB, Calscand International. *User's Manual, MOVECOMP-PC, An Air Infiltration and Ventilation System Program.*

Eriksson, L; Söderlind, G; Bring, A. 1992. "Numerical Methods for the Simulation of Modular Dynamical Systems." Bulletin 21. Dept. of Building Services Engineering, Royal Institute of Technology, Stockholm.

Feustel, H. E; Rayner-Hooson, A. 1990. *COMIS Fundamentals.* Lawrence Berkeley Laboratory, CA, USA.

Hensen, J.L.M; Clarke, J.A. 1991. "A Simulation Approach to the Evaluation of Coupled Heat and Mass Transfer in Buildings." In *Proceedings of the IBPSA Building Simulation '91* (Nice, Aug.).

Herrlin, M. 1992. "Air-Flow Studies in Multizone Buildings." Bulletin 23. Dept. of Building Services Engineering, Royal Institute of Technology, Stockholm.

Lindberg, B. 1985. "An Algorithm for Simulation of the Pressure Distribution in a Building." Research Report TRITA-NA-8503. Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm.

Sahlin, P; Bring, A. 1991. "IDA Solver - a Tool for Building and Energy Systems Simulation." In *Proceedings of the IBPSA Building Simulation '91* (Nice, Aug.).

Sahlin, P; Bring, A; Sowell, E. 1992. "The Neutral Model Format for Building Simulation." Bulletin 24. Dept. of Building Services Engineering, Royal Institute of Technology, Stockholm, Sweden.

Sahlin, P. 1993. "IDA Modeller - a Man-Model Interface for Building Simulation." To appear in *Proceedings of the IBPSA Building Simulation '93* (Adelaide, Aug.).

Walton, G. N. 1989. *AIRNET - a Computer Program for Building Airflow Network Modelling.* U.S. Department of Commerce, National Institute of Standard and Technology, Gaithersburg, MD, USA

NMF-BASED ASPECT MODELS IN STEP/EXPRESS FOR BUILDING AND PROCESS PLANT SIMULATION

Per Sahlin
Building Services Engineering
Royal Institute of Technology
100 44 STOCKHOLM, SWEDEN
e-mail: plurre@kth.se

Curt Johansson
Construction Management
Royal Institute of Technology
100 44 STOCKHOLM, SWEDEN
e-mail: curt@ce.kth.se

Abstract

Automated design performance assessment through simulation will be an important aspect of future product model technology. The research in this area has so far been focused on traditional simulation tools. However, the rapid development of new structurally different tools calls for a shift of attention. New object-oriented methods of describing simulation models can and should be integrated with the product model itself. In this paper we will briefly review a current development trend in continuous simulation and present a new language for model description, Neutral Model Format (NMF), which in recent years has gained considerable attention in the field of building simulation. The possibility of joining the continued NMF development with the STEP domain is discussed and some examples of NMF based EXPRESS models are presented.

1. INTRODUCTION

One driving factor behind product model research is that it will give designers direct access to easy and repeated design evaluation. Obviously, cost estimates, bills of materials, and various drawings should be easily generated from product model data, but of equal importance are measures of the dynamical performance of the design at hand. In the AEC field the EEC COMBINE project (phase 1) has demonstrated feasibility of data mapping from an EXPRESS-based data model of a building to a range of established building performance evaluation (BPE) tools [Augenbroe 1993]. Phase 2 of this project seeks to put this technology to use among practitioners in the field. Another industrial sector with considerable activity in both product modelling and simulation is the process industry.

The authors of this paper have for some time worked with new simulation techniques and languages for continuous modular systems. These techniques are applicable to a

large class of static and dynamical simulation problems in, e.g., the building, energy and process industries. One important aspect of this work has been involvement in the definition of a standard format, Neutral Model Format (NMF), for expression of component level simulation models. The purpose of this paper is to investigate the applicability of STEP technology in the continuation of this work.

Currently, *component models* (primitive models) are automatically translated from NMF to the proprietary format of the target simulation environment. For example, an NMF model of an axial fan is used to generate an axial fan class in, e.g. IDA [Sahlin 1991]. The class is then instantiated in the target environment. The instances are furnished with suitable parameters, and incorporated into a system model. The next natural step in the NMF development is to formulate an environment independent way of expressing and communicating instantiated *system models* as well. Several authors have already suggested and even implemented such NMF extensions [Kolsaker 1994a, Lorenz 1994]. Since object oriented simulation is a highly relevant topic for product modelling efforts [Augenbroe 1991], we will analyze the implications of using EXPRESS for data modelling of NMF instantiated system models.

In the next two sections a brief overview is given of current work on so called object oriented simulation methods, mainly in the context of building simulation, and of the Neutral Model Format.

2. OBJECT ORIENTED SIMULATION ENVIRONMENTS

The term object oriented is perhaps not the best descriptor for these tools but it has nevertheless become widely used and we will use it here as well. The object orientation concerns mainly the modularity of the physical systems that are being modelled and not so much software techniques. Naturally, most recent developments also use object oriented programming to varying degrees.

2.1 PHYSICAL SYSTEMS AND MATHEMATICAL MODELS

Physical systems that we aim to simulate are modular in nature, i.e. they naturally decompose into subsystems. Frequently, identical subsystems are repeated a number of times in a model, a fact that is taken advantage of in many tools. Furthermore, the systems should have a basically continuous behavior, meaning that equations used to describe them, as well as forcing functions, will have a limited number of discontinuities. Purely event driven systems are excluded.

Models may be expressed in several ways. Bond graphs, linear graphs, block diagrams, electrical analogies, and mathematical equations are frequently used modes of expression. Also used, for mainly historical reasons, are subroutines in some programming language. A discussion of pros and cons of various methods of description can be found in [Lorenz 1987].

If characterized by equations, the physical systems under consideration will require both algebraic and differential equations. Differential equations can be either ordinary (ODE) or partial (PDE), although current tools require that PDEs are explicitly discretized in space and thus turned into ODEs. Note that in contrast to many widely used commercial tools the simulation environments we are concerned with here are

not limited to ODEs only. They allow a free mixture of algebraic and ordinary differential equations generally referred to as differential-algebraic systems of equations (DAE).

Furthermore, the simulation tools under discussion are rarely used for applications where a strict formalism for generating governing equations exists. In, e.g., electrical circuit analysis, multibody mechanics, or structural analysis special purpose systems may be more advantageous.

Examples of physical systems that fit this description can be found in many fields. Chemical process plant simulation is a significant area of application. Energy distribution networks and plants is another. The authors of this paper have mainly worked with building related systems and important applications within this field are: thermal processes in walls and spaces; air and water based distribution systems and plants; and automatic control.

2.2 SEPARATION OF MODELLING AND SOLVING ACTIVITIES

In contrast to many established design tools, e.g. in building simulation, OOSEs separate strictly between the modelling and subsequent system solution activities. A modelling tool is often used for model formulation. This tool generates a system model, generally expressed in a *modelling language*. The model is then treated by a solver. An important benefit of a separate solver is that it may be altered or even exchanged with minimal interference with the modelling environment.

Key characteristics of the modelling language, such as expressiveness and level of standardization, are critical to the usefulness and development potential of the overall OOSE. The Neutral Model Format is part of such a modelling language. This paper describes one way towards a complete modelling language that may be standardized.

2.3 TARGET USERS AND SOFTWARE STRUCTURE

Most of the simulation tools under discussion are intended for quite sophisticated users, who are well versed in mathematical modelling, numerical methods and advanced use of computers. These tools are not directly suited for designers, without special simulation expertise, that use simulation as one of several methods for design evaluation. However, for the expert, they generally provide an efficient graphical environment for model building, simulation and analysis.

Other tools, e.g. EKS and IDA, are primarily intended for efficient design tool production, and the normal end user will rarely interact directly with the underlying OOSE techniques.

2.4 AVAILABLE AND EMERGING OOSEs

A few tools and environments with the discussed main characteristics are already matured and available and others are under development. Among the available ones are e.g.:

TRNSYS was developed during the seventies at the Solar Energy Lab at the University of Wisconsin. It was one of the first modular simulation solvers for DAEs and it is

distributed as a Public Domain product. Several compatible modelling tools have been developed, e.g. PRESIM.

HVACSIM+ is a solver with similar characteristics as TRNSYS in terms of model format and structure, but more recent numerical techniques are utilized. It was developed by NIST in Maryland and released in the mid eighties on a Public Domain basis.

SANDYS is a general DAE solver and textual modelling environment developed by ASEA, Sweden, in the early eighties. It is commercially available from ABB Corporate Research.

ALLAN-NEPTUNIX is a graphical modeller and solver combination developed by Gaz de France and CISI Engineering. It is since a few years commercially available from the developers.

ESACAP is a recently developed DAE solver by the European Space Agency. It is commercially available from STANSIM, Denmark.

DYMOLA is a text based commercial modelling tool with symbolic algebra capabilities and interfaces to several solvers. A GUI is under development. Available from DYNASIM, Lund, Sweden.

Some tools under development are:

CLIM 2000, a graphical modelling tool for building applications, is developed by Electricite de France.

MS1 is a graphical multi input language modeller with interfaces to several solvers by Lorenz Consulting, Liege, Belgium in cooperation with Electricite de France.

IDA, a graphical modelling environment and solver, is under development at the Swedish Institute of Applied Mathematics.

SPARK is a solver and graphical model editor under development at LBL, Berkeley, California.

OMSIM is a graphical modelling tool under development at the Dept. of Automatic Control at the Lund Institute of Technology, Sweden.

EKS is a C++ toolkit for development of energy related simulation design tools, by among others the Univ. of Strathclyde, Scotland.

3. THE NEUTRAL MODEL FORMAT

Without a comprehensive, validated library of ready made component models in a relevant application area most simulation environments are rather useless. To develop all necessary models from scratch is, in most projects, quite unrealistic. And since the cost of developing a substantial library easily exceeds the development cost of the simulation tool itself, it is important to be able to reuse what other people already have done. This was the basic motivation for proposing a text based neutral model

format to the building simulation community in 1989 [Sahlin and Sowell 1989]. Since then the proposal has attracted a great deal of interest from environment developers and users in several application fields. Prototype translators have been developed for IDA [Kolsaker 1994a], SPARK [Nataf 1994] and ESACAP [Pelletret 1994a]. Translator development projects have been funded for TRNSYS, HVACSIM+ [ASHRAE 1994], and MS1 [Lorenz 1994]. Export and import capabilities are planned and partly implemented for ALLAN-NEPTUNIX [Jeandel 1994].

Pending formal standardization, ASHRAE (American Society of Heating, Refrigerating, and Air-conditioning Engineers) has formed an ad hoc committee that approves changes to the present format.

NMF has two main objectives: (1) models can be automatically translated into the local representation of several simulation environments, i.e. the format is program *neutral* and machine readable; and (2) models should be easy to understand and express for non-experts. The first objective enables development of common model libraries, which can be accessed from a number of simulation environments.

3.1 BASIC NMF FEATURES

Internal component model behavior is described by a combination of algebraic and ordinary differential equations. Equations may be written in any order and in the form

<expression> = <expression>;

NMF only *states* equation models, while *solution* of equations is, in some cases, left to the target environment (e.g. IDA, or SPARK), or the NMF translator in others (e.g. TRNSYS, or HVACSIM+).

NMF supports model encapsulation through a link concept, i.e. models may only interact via variables appearing in LINK statements. To enhance and encourage model plug compatibility, links and variables are globally typed. The idea is that basic list of such types should be included in each revision of the standard, but that users may add to the list as need arise. A selection of such global types is:

```
QUANTITY_TYPES

/* type name      unit              kind */

   Area           "m2"              CROSS
   Control        "dimless"         CROSS
   Density        "kg/m3"           CROSS
   Factor         "dimless"         CROSS
   HeatCap        "J/(K)"           CROSS
   HeatCapA       "J/(K m2)"        CROSS
   HeatCapM       "J/(kg K)"        CROSS
   HeatCond       "W/(K)"           THRU
   HeatFlux       "W"               THRU
   HeatFlux_k     "kW"              THRU
   Temp           "Deg-C"           CROSS

LINK_TYPES

/* type name       variable types... */

/* generic        (arbitrary, arbitrary,...) implicitly
defined */
   F              (Force)
   FL             (Force,Length)
   Q              (HeatFlux)
   T              (Temp)
   PMT            (Pressure, MassFlow, Temp)
   PMTQ           (Pressure, MassFlow, Temp, HeatFlux)

   MoistAir       (Pressure, MassFlow, Temp, HumRatio)
   BidirFlow      (Pressure, MassFlow, Enthalpy, HeatFlux)
```

A quantity type includes a physical unit and information about potential (across) or flow (through) type. A link type is simply an ordered list of quantity types. Let us now look at an example of a rather simple NMF model using the heat equation in one dimension.

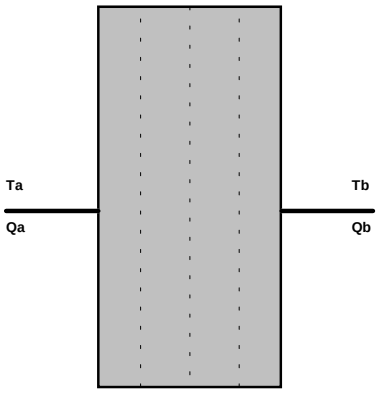


Figure 1. A finite difference model of a wall with one homogeneous layer. Temperature and heatflux on each terminal.

```
CONTINUOUS_MODEL tq_hom_wall

ABSTRACT
"A 1D finite difference wall model. One homogeneous layer.
TQ interfaces on both sides."

EQUATIONS
/* space discretized heat equation */
c_coeff * T'[1] = Taa - 2.*T[1] + T[2] ;
c_coeff * T'[n] = T[n - 1] - 2. * T[n] + Tbb ;

FOR i = 2, (n -1)
c_coeff * T'[i] = T[i - 1] - 2. * T[i] + T[i + 1];
END_FOR ;
/* boundary equations */
0 = -Ta + .5 * (Taa + T[1]) ;
0 = -Tb + .5 * (T[n] + Tbb) ;
0 = -Qa + d_coeff * (Taa - T[1]) ;
0 = -Qb + d_coeff * (Tbb - T[n]) ;

LINKS
/*  type           name      variables ....  */
    TQ             a_side    Ta, POS_IN Qa ;
    TQ             b_side    Tb, POS_IN Qb ;

VARIABLES
/* type    name        role def     min    max      description*/
  Temp     T[n]        OUT   20. abs_zero  BIG  "temperature profile"
  Temp     Ta          OUT   20. abs_zero  BIG  "a-side surface temp"
  Temp     Tb          OUT   20. abs_zero  BIG  "b-side surface temp"
  Temp     Taa         OUT   20. abs_zero  BIG  "a-side virtual temp"
  Temp     Tbb         OUT   20. abs_zero  BIG  "b-side virtual temp"
  HeatFlux Qa          IN     0.    -BIG   BIG  "a-side entering heat"
  HeatFlux Qb          IN     0.    -BIG   BIG  "b-side entering heat"

MODEL_PARAMETERS
/* type    name       role  def mi  max        description  */
  INT      n          SMP    3   3  BIGINT "number of temp layers"

PARAMETERS
/*   type    name       role   [def   [min   max]]   description*/

/* supplied parameters */
  Area      a            S_P    10.  SMALL BIG  "wall area"
  Length    thick        S_P   .2    SMALL BIG  "wall total thickness"
  HeatCondL lambda       S_P   0.5   SMALL BIG  "heat transfer coeff"
  Density   rho          S_P   2000  SMALL BIG  "wall density"
  HeatCapM  cp           S_P   900.  SMALL BIG  "wall heat capacity"

/* computed parameters */
  generic   d_coeff      C_P              "lambda*a/dx"
  Length    dx           C_P              "layer thickness"
  generic   c_coeff      C_P              "rho*cp*dx*dx/(lambda*3600.)"


PARAMETER_PROCESSING
dx := thick / n ;
c_coeff := rho * cp * dx * dx / (lambda * 3600.) ;
d_coeff := lambda * a * dx ;

END_MODEL
```

To enable direct model translation to input-output oriented environments (e.g. TRNSYS, or HVACSIM+), variable declarations have a role attribute indicating IN for given variables and OUT for calculated ones.

Variables and parameters may be vectors or matrices. A parameter is anything that must remain constant throughout every simulation. Links may also be vectors, thus allowing models with variable number of ports. Vector and matrix dimensions are governed by a special type of parameter, model parameters. Regular and model

parameters are divided into two categories, user supplied and computed, algorithmic computation of which is described in the parameter processing section.

Arbitrary foreign functions in Fortran 77 or C may be defined, either globally or locally within a model.

Special functions are defined to handle discontinuities, hysteresis, linearization, and errors. A more complete account of NMF is given in the reference report [Sahlin, Bring, and Sowell 1994].

3.2 NMF DEVELOPMENT DIRECTIONS

Currently, a reasonable agreement about the NMF grammar has been reached. Developers can count on stability of the present format and backward compatibility. This enables us to get on with the work of defining NMF-based component model libraries and to develop further NMF translators. Several substantial model libraries have already been developed and many more are underway.

Regarding the format itself, several extensions have been suggested. In the discussion of these it is important to bear in mind that, at the time of the original proposal, NMF was not primarily intended as a replacement of existing proprietary model languages, but as a complement, enabling component model exchange and library building.

Planned extensions and supporting tools that fall within the scope of the original NMF intentions are:

1. An NMF handbook with style guidelines for model architecture. The current NMF manual is completely insufficient as a pedagogical tool. (Encompassed by funded project [ASHRAE 1994].)

2. Model documentation guidelines and templates, storage and retrieval mechanisms. This area is addressed by Pelletret in a recent (draft) proposal [Pelletret 1994]. The ESPRIT OLMECO project - development of a large mechatronics library - is another source of inspiration.

3. Investigation regarding adaptable models, through property inheritance and/or through hierarchical modelling. Property inheritance between models may result in better model reuse but it will on the other hand also have negative effects on model portability, since inheritance trees must be passed when shipping a model. This leads to reconciliation problems if a similar, but not identical, tree exists on the receiving side.

4. Model library structure and management tools, including mechanisms for model browsing and retrieval.

5. Discrete time (sampling) models. This is necessary to study sampling control circuits.

There are several additional items that belong in this list - such as formal rules for permitted model connections and a language or keyword system for expression of model assumptions - that are omitted here due to space.

In the context of a complete modelling language the present format lacks the ability to express:

1. Component model instances, with parameter values, initial values of all variables, and information about boundary variables..

2. Hierarchical systems of such instances.

3. Numerical simulation parameters, such as tolerances, stepsize limits, algorithm selection commands, that can be generalized for a large class of solvers.

4. Graphical schemata for user presentation of simulation models. Large models are much easier to comprehend if they are described graphically.

The drive for development of a complete NMF-based modelling language comes primarily from developers of new modelling tools, who see little reason to develop proprietary formats. Two such developers have made concrete proposals and implementations are well underway [Lorenz 1990], [Kolsaker 1994].

4. WHY STEP/EXPRESS?

STEP (STandard for the Exchange of Product model data) is an international standard for product descriptions [ISO TC 184 1993]. The data for these descriptions are modelled in a special language called EXPRESS, which is in itself part of the STEP standard. EXPRESS is an object oriented language that is particularly well suited for information modelling. A subset of EXPRESS is EXPRESS-G, a fully graphical language for data modelling. EXPRESS-G schemata can automatically be translated into textual EXPRESS code, which in turn can be translated into, e.g., C++ class definitions. A number of tools and related standards are (and will be) available for STEP/EXPRESS. A (default) textual representation of any EXPRESS schema is for example implicitly defined (STEP physical file).

Since the first proposal in 1989 the discussion about various NMF-constructs has focused on the grammar. The textual appearance of selected models has been the main object. This is of course quite appropriate for the equation core of component models, but for instantiated system models and related data it may be more fruitful to regard data models directly, and to treat textual representation as one of several possible views. EXPRESS seems to be an appropriate vehicle for the future NMF discussion. Further reasons for the employment of STEP technology include:

- Simulation models will most likely be an important aspect of many product model applications, and they should therefore be encompassed by STEP, either as pure aspect models or as parts of global models

- Many existing STEP/EXPRESS resources will be useful for development of NMF-oriented application tools

- The fact that STEP physical files most likely will be more difficult to read (for humans) than a tailored high level language is of little consequence for realistic-size simulation models, which generally are of such magnitude that they rarely are

printed and studied in their raw form

4.1 PRESENT NMF IN STEP

In this our initial work we have chosen to focus directly on the imminent problem of defining conceptual models of NMF instances, and of hierarchical systems of such instances. This means that nothing is said about the internal behavior, e.g. equations, of a model. Only its state is encompassed, and it is assumed that the underlying NMF model is known to all parties.

Another interesting issue is of course the conceptual models of internal behavior as well, i.e. to model the present NMF in EXPRESS, with entities such as equation, if_then_else_clause, etc. Such models are necessary for development of NMF parsers and translators.

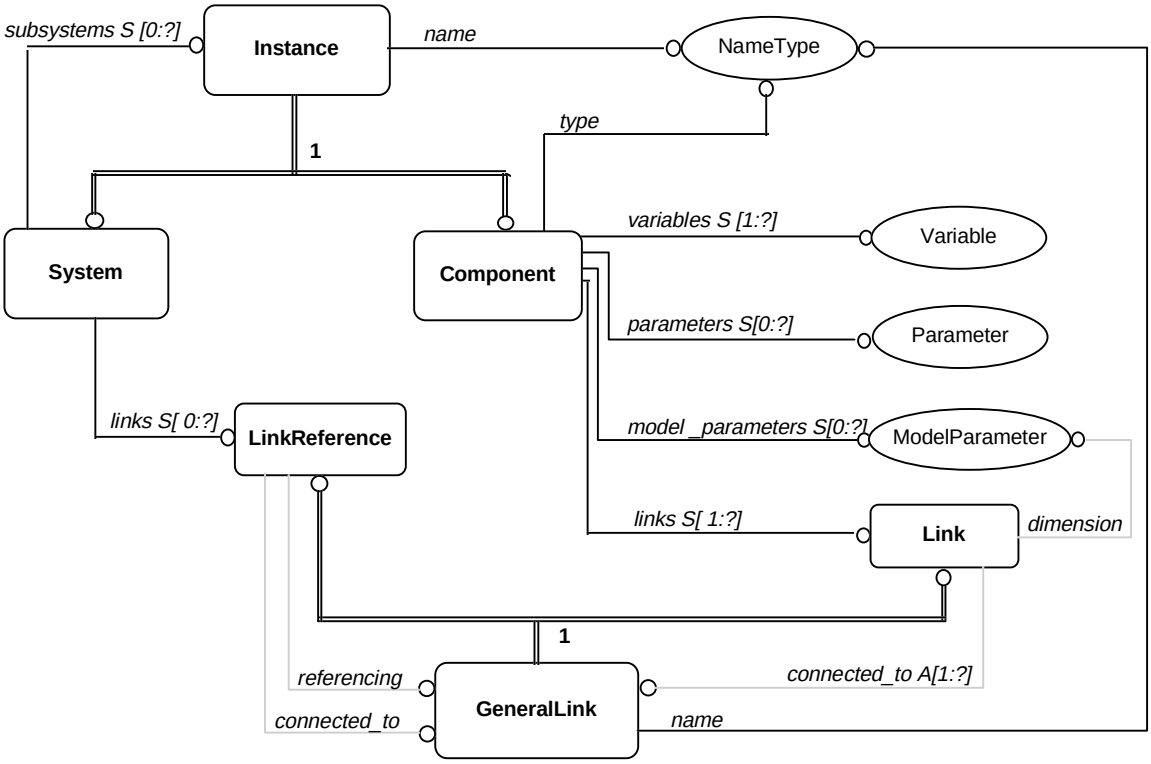
The main motivation for remodelling the present NMF in EXPRESS is completeness. New component models could be communicated with the same tools and protocols. A potential EXPRESS-based STEP standard would not have to rely on an additional non-EXPRESS standard.

Additional benefits can be expected for design and implementation of NMF component model databases and management tools.

The present conclusion is that it would be worthwhile to model the present NMF in EXPRESS. However, since the discussion of instances and systems can be carried out separately, we have chosen to focus on this in our initial work.

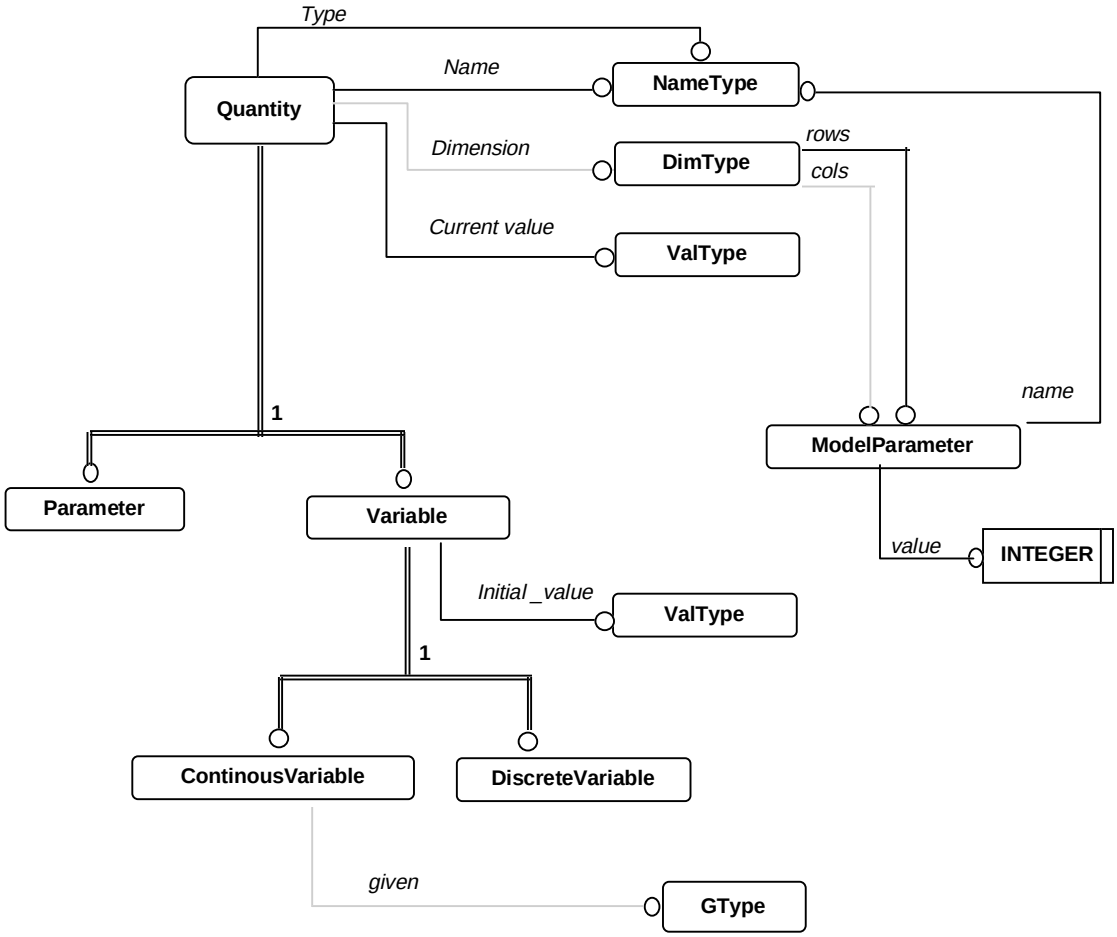
5. NMF MODEL INSTANCES IN EXPRESS-G

In the following an EXPRESS-G representation of NMF component and system model instances is presented. An instance is a specific occurrence of a model expressing the full state, in terms of its parameter values, variable values, and associated data. Schemata 1 through 4 shows the EXPRESS-G representation of this data.



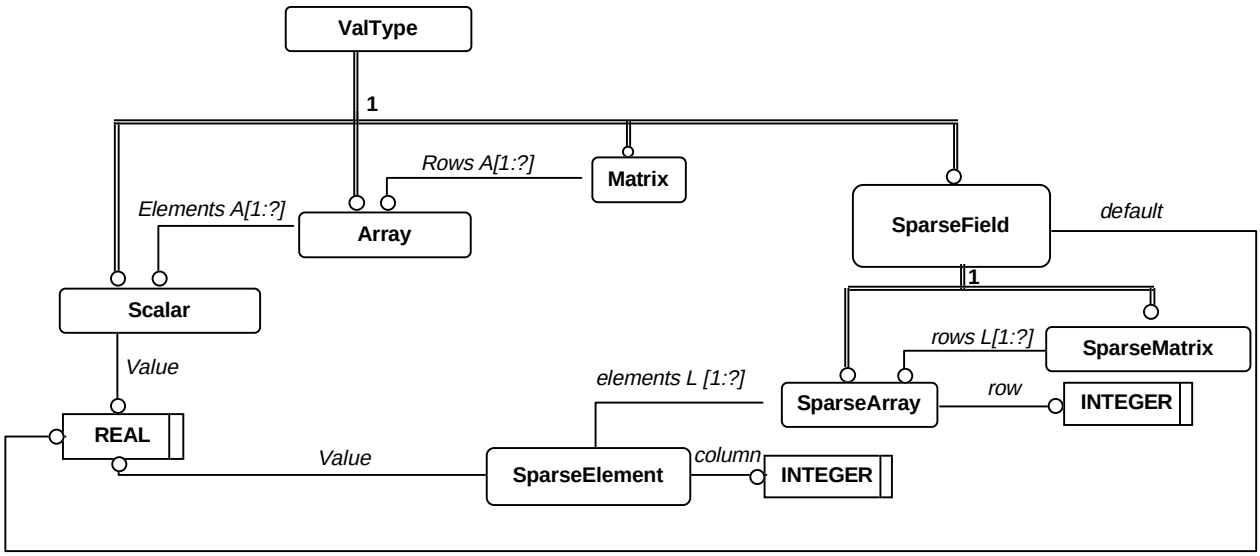
Schema 1

Schema 1 shows the structure of an NMF model instance, which may appear as either a System, with references to underlying subsystems, or as a Component with object bags for variables, parameters, model parameters, and links, each of which is specified more closely in the following schemata . Model parameters are named integers that are used for dimensioning of arrays and matrices. Links are the connection ports of Components. The ports of a System are called LinkReferences. They provide reference chains to underlying Links. The distinction between the quantity subclasses parameters and variables is that parameters always remain fixed at a given value throughout a simulation, while variables, naturally, vary.



Schema 2

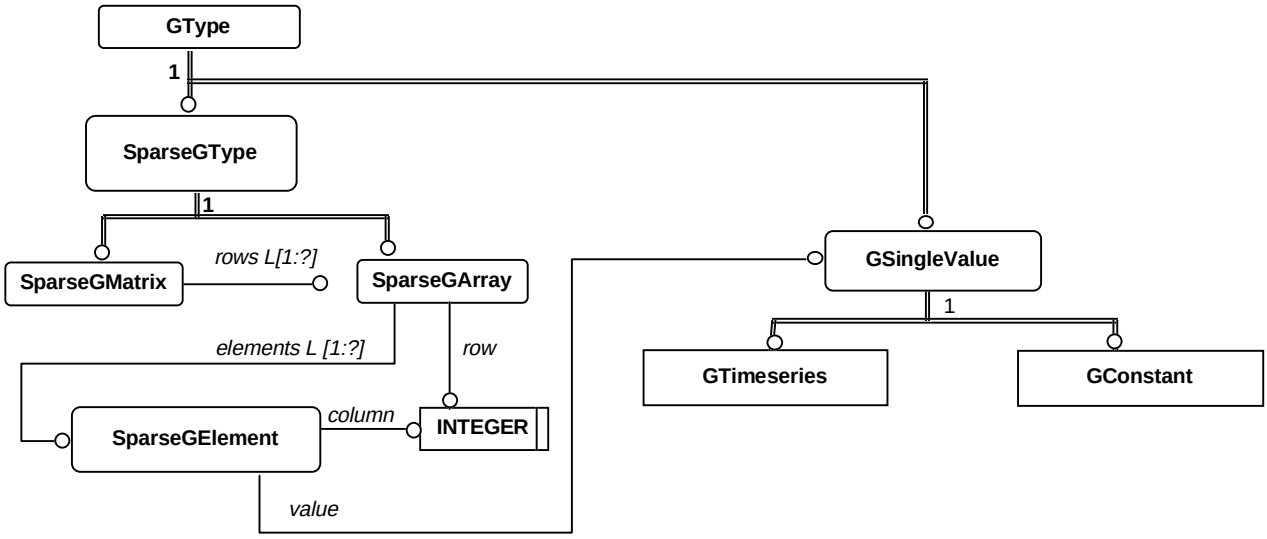
A quantity has a type (the NMF QUANTITY_TYPE referred to), a name, a dimension (if non-scalar), and a current value. Variables also have an initial value, which holds the state at the beginning of a simulation for dynamic (or state) variables and an initial value guess for algebraic variables. Discrete variables is a provision for future development of discrete time NMF models and is not currently used.



Schema 3

Continuous variables also have an optional flag given which, if present, indicates that a variable, or selected parts of a field variable, are to be kept at a given value throughout the simulation.

Schemata 3 and 4 specify storage structures and stored elements for current and initial values (schema 3) and for given flags (schema 4).



Schema 4

Values may be stored in either a sparse matrix storage structure or in full matrices (or ditto arrays). Initial values are generally stored in a sparse structure where only exceptions from the default value are listed.

Since the great majority of variables are calculated, the given flag is stored in a sparse structure as well. The flags themselves are either a reference to a time series of values (not specified in detail) or a GConstant symbol, indicating that the variable is to be kept at its initial value throughout the simulation.

6. CONCLUSIONS AND FUTURE WORK

Our present work suggests that EXPRESS is suitable for modelling of many of the data structures that are relevant for continuous simulation of modular systems. If not incorporated into the STEP effort, a continuous simulation language standardization project could certainly operate in a similar fashion and use many of the same methods and tools.

Next on the agenda will be to test the functionality of the suggested data structures by writing a parser for, initially, IDA system model descriptions.

References

Augenbroe, G, and F. Winkelmann, 1991, Integration of Simulation into the Building Design Process, proc. of Building Simulation ´91 , Nice, France, International Building Performance Simulation Association

Augenbroe, G. (ed), 1993, COMBINE Final Report, CEC-DG XII-JOULE

ASHRAE 1993. Invitation to Submit a Research Proposal on an ASHRAE Research Project: 839-TRP Development of a Component Model Translator for the Neutral Model Format, American Society for Heating, Refrigerating and Air-Conditioning Engineers, Atlanta, GA

Buhl, W.F., E.F. Sowell, and J-M Nataf, 1989. Object-oriented Programming Equation-Based Submodels, and System Reduction in SPANK, proc. of Building Simulation ´89 , Vancouver, BC, International Building Performance Simulation Association

ISO TC 184 1993. The STEP Standard, draft international standard DIS 10303, continuously since 1992 published in several different parts

Jeandel A. 1994, Personal communication

Kolsaker, K. 1994a. NEUTRAN-supported NMF Enhancements, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Kolsaker, K. 1994b. Simpler NMF Description of Advanced Models Using Hierarchical Modelling and Data Abstraction, to be presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994

Lorenz F. 1987, Reflections about Representation Methods, proc. workshop on the future of building energy modelling, Ispra, Italy, Nov. 1987, CEC EUR 11603 EN PREPRINT, May 1988

Lorenz F. 1990. Brief Description of the MS1 (Modelling System 1) Project, private communication

Lorenz F., 1994, Comments on the Neutral model Format, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Nataf J.-M. 1994, Translator from Neutral Model Format to SPARK, draft paper presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Pelletret, R. 1994, Personal communication

Pelletret, R., S. Soubra 1994b. Standardizing Model Documentation - The PROFORMA Experience, presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Sahlin, P, E.F. Sowell 1989, A Neutral Format for Building Simulation Models, proc. of Building Simulation ´89 , Vancouver, BC, International Building Performance Simulation Association

Sahlin, P. 1991, IDA - a Modelling and Simulation Environment for Building Applications, Swedish Institute of Applied Mathematics, ITM Report no. 1991:2

Sahlin, P., A. Bring, and E. F. Sowell, 1994. The Neutral Format for Building Simulation, Version 3.01, Swedish Institute of Applied Mathematics, ITM Report no. 1994:2

Sowell, E.F. 1994. A Proposal for Hierarchical Submodels in NMF, to be presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994

FUTURE TRENDS OF THE NEUTRAL MODEL FORMAT (NMF)

Per Sahlin[‡], Axel Bring[‡], and Kjell Kolsaker[†]

KTH, Stockholm, Sweden, and NTH, Trondheim, Norway

ABSTRACT

The Neutral Model Format for building simulation was proposed in 1989 as a means for documentation and exchange of models. It has attracted much interest and an acceptance as a potential standard, maintained by ASHRAE's TC 4.7 technical committee. So far, the format has only been directed towards component (leaf) models, but many suggestions have been made to extend it to also cater for systems of component models. A brief review of NMF is given. This paper makes detailed proposals for NMF extensions covering hierarchical modeling, inheritance, equation-based function definitions, link types with associated equations, and a general method to handle non-standard extensions. A future development of NMF towards STEP/EXPRESS is also discussed.

1. INTRODUCTION

Most researchers in building simulation agree that the present technology - with a large number of stand-alone monolithic Building Performance Evaluation (BPE) tools - is unlikely to ever satisfy constantly changing industrial demands. An entirely new approach is called for. One solution is provided by the so called object oriented simulation environments (OOSE), several of which are available or under development. These new modular systems offer radically increased levels of adaptability, while customizable user interfaces ensure high quality end-user tools. For developers, the production time of new BPE tools is drastically reduced and the resulting tools can be efficiently maintained. For end-users, the main advantage is realistic access to a range of related BPE tools, that have uniform interaction principles and that are data compatible with each other.

A common characteristic between OOSE's is that physical models are regarded as data; they are not regarded as such in most present tools, but are generally inextricably bound to the program code. In OOSE's, submodels may easily be added, changed, removed and combined arbitrarily. This is where the real power of the new systems lie. However, it also creates a common need for well validated and documented models, preferably expressed in a standardized machine-readable way. In fact, without a comprehensive, validated library of ready made component models in a relevant application area, most simulation environments are of limited usefulness. To develop all necessary models from scratch is, in most projects, quite unrealistic. And since the cost of developing a substantial library easily exceeds the development cost of the simulation tool itself, it is important to be able to reuse what other people have already done. This was the basic motivation for proposing a text based neutral model format to the building simulation community in 1989 [Sahlin and Sowell 1989]. Since then the proposal has attracted a great deal of interest from environment developers and users in several application fields. Translators have been developed for IDA [Kolsaker 1994a], SPARK [Nataf 1994], ESACAP (prototype) [Pelletret 1994a], TRNSYS, HVACSIM+ [Sahlin 1995], and MS1 [Lorenz 1994]. Export and import capabilities are planned for ULM (ALLAN) [Jeandel 1994].

Pending formal standardization, ASHRAE (American Society of Heating, Refrigerating, and Air-conditioning Engineers) has formed an ad hoc committee that approves changes to the present format.

NMF has two main objectives: (1) models can be automatically translated into the local representation of several simulation environments, i.e. the format is program *neutral* and machine readable; and (2) models should be easy to understand and express for non-experts. The first objective enables development of common model libraries, which can be accessed from a number of simulation environments.

Section 2 gives a brief introduction to NMF. The proposed extensions are presented in section 3 and illustrated with numerous examples. Section 4 suggests advantages attainable by a STEP [ISO TC 184 1993] orientation of future NMF development.

2. NMF REVIEW

Internal component model behavior is described by a combination of algebraic and ordinary differential equations. Equations may be written in any order and in the form

<expression> = <expression>;

NMF only *states* equation models, while *solution* of equations is, in some cases, left to the target environment (e.g. IDA, or SPARK), or the NMF translator in others (e.g. TRNSYS, or HVACSIM+).

NMF supports model encapsulation through a link concept, i.e. models may only interact via variables

[‡] Dept. of Building Services Engineering,
Royal Institute of Technology,
100 44 STOCKHOLM, SWEDEN
phone: +46-8-411 32 38, fax: +46-8-411 84 32,
e-mail: abring or plurre@engserv.kth.se

[†] NTH/ VVS-teknikk
7034 TRONDHEIM, NORWAY
phone: +47-7-59 25 09, fax: +47-7-59 38 59
e-mail: Kjell.Kolsaker@termo.unit.no

appearing in LINK statements. To enhance and encourage model plug compatibility, links and variables are globally typed. The idea is that a basic list of such types should be included in each revision of the standard, but that users may add to the list as need arises. A selection of such global types is located in Section 3.6.1.

A quantity type includes a physical unit and information about potential (across) or flow (through) type. A link type is simply an ordered list of quantity types. An example of an NMF model using the heat equation in one dimension can also be found in Section 3.6.1.

To enable direct model translation to input-output oriented environments (e.g. TRNSYS, or HVACSIM+), variable declarations have a role attribute indicating IN for given variables and OUT for calculated ones. Variables and parameters may be scalars, vectors, and matrices. A parameter is anything that must remain constant throughout every simulation. Links may also be vectors, thus allowing models with variable number of ports. Vector and matrix dimensions are governed by a special type of parameter, model parameters. Regular and model parameters are divided into two categories, user supplied and computed, algorithmic computation of which is described in the parameter processing section.

Arbitrary foreign functions in Fortran 77 or C may be defined, either globally or locally within a model. Special functions are defined to handle discontinuities, hysteresis, linearization, and errors. A more complete account of NMF is given in the reference report [Sahlin, Bring, and Sowell 1994[1]]

3. PROPOSED EXTENSIONS

Ever since the original NMF paper there has been a healthy flow of proposals for new features. The bulk of these have concerned additional support for good model and code structure. Some have suggested syntax for extensions that were already implicitly defined in terms of existing constructs; others take up entirely new threads of thought. It seems that we now have enough material in terms of modeling experience and proposal "raw material" to decide on a well balanced set of extensions, well in tune with the original NMF ideas. This section will outline our view of a package of such extensions: hierarchical modeling on the link level; function definitions that may contain implicit equations and that can be symbolically processed; property links - link types with associated equations; model inheritance; and a standardized gateway for solver specific extensions.

3.1 Hierarchical Modeling

Since the most imminent need for model exchange between environments is on the component rather than on the system level, no standardized syntax for system models was included in the original NMF proposal. System models would, for the time being, be handled separately in each target environment. This decision has been criticized by many and there has consequently been a number of proposed system modeling constructs. Developers that are looking for a complete modeling language clearly need a standardized way of expressing system models. It might seem less obvious that a language, mostly intended as a vehicle for construction of component model libraries, would need a system modeling capability. However, the division between what is a component and what is a system is generally so vague that it becomes unreasonable to base any language limitations on it. The arguments in favor of inclusion of system modeling features seem quite convincing. The next issue is to determine a good range of hierarchical modeling constructs.

Most obvious is a system model construct, with references to underlying submodels, which of course might be system models themselves, and the possibility to interconnect submodels at the link rather than the individual variable level. This type of model is clearly implied already in the existing link construct. We will use the standard NMF two component system - a "Heating Collector" (fig. 1) for mixing and heating two air streams - to illustrate the suggested syntax.

[1] An ASCII version of the reference report and other NMF material is located at: `ftp://mailbase.ac.uk/pub/lists-f-j/ibpsa-nmf`. Connect as user `anonymous` and give your e-mail address as password.

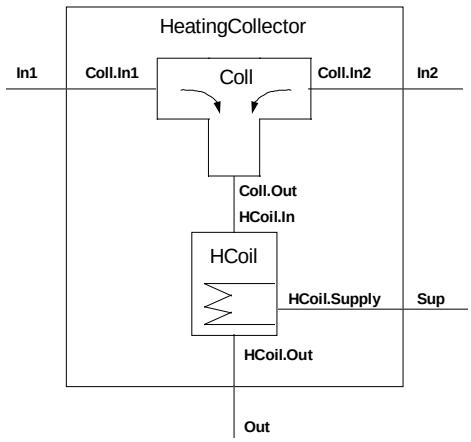


Fig. 1. System model with a collector and heating coil

```
SYSTEM_MODEL Heating_Collector

PARAMETERS
    HeatFlux    my_flux     S_P     100     "a parameter";

SUBMODELS       // Declare submodel instances
//  class       instance
//      data
    Collector   Coll;
    Heat_Coil   Hcoil,
        rise_time := 30;
        // rise_time is a parameter of Heat_Coil

CONNECTIONS     //Link level internal connections
//  inst.link   =   inst.link;
    Coll.Out    =   Hcoil.In;

LINKS           // Lift (and rename) submodel links to
                // become Heating_Collector links
//  name        =   inst.link;
    In1         =   Coll.In1;
    In2         =   Coll.In2;
    Out         =   Hcoil Out;
    Sup         =   Hcoil.Supply;

DOCUMENTATION
END_DOCUMENTATION

END_MODEL
```

Submodels are declared (instantiated) in the SUBMODELS Section; we will in this text use the word instantiated for a declared submodel, although the submodel is not physically instantiated until the surrounding system model is. Parameter values and variable initial values may be supplied in order to override defaults in the instantiated model. Internal connections between submodels are made in the CONNECTIONS Section, where variables of joined models are connected collectively at the link level, with appropriate signs for THRU variables.

The LINKS Section defines those submodel links that should be visible on the boundary of the defined system, and thus be available for connection at the next level up in the hierarchy.

A PARAMETER section may be declared for systems. Parameter expressions may be used in the SUBMODELS section.

The present NMF continuous_model maps in a natural way to TRNSYS or HVACSIM+ type routines. The system_model construct that we are suggesting here would enable NMF expression of TRNSYS decks or HVACSIM+ work files as well.

3.2 Hybrid System/Continuous Models

In this extension proposal, equations are not allowed in system_ models, nor are submodel references allowed in present continuous_models. It is in our opinion vital to have a pure system model that only contains submodel references and never deals with individual variables and internal submodel behavior. This allows simplified modeling software, without equation manipulation capability, for interconnection of existing submodels. Examples of such tools are the present front-end programs for TRNSYS and HVACSIM+. It also makes possible top-down modeling, where the precise structure of submodels and connected links remain undecided. So far, NMF only handles equation based submodels, but the intention is to extend this to other model categories, perhaps most imminently discrete time models for sampling controllers etc. The (pure) system_ model, as proposed here, will be able to accommodate submodels of several types.

For simplicity, our initial extension idea was to avoid models that mix submodel references with individual equations - hybrid models - in NMF. Hybrid models are difficult to handle for a TRNSYS/HVACSIM+ translator. Since they may contain any type of equation, they have to be mapped to type subroutines rather than to decks/work files. This means that the hierarchical structure of the system model part of the hybrid model will have to be flattened out. Since the hierarchy might be arbitrarily deep, generated variable and parameter names in the flattened model will be either

very long, if you want to keep original names, or else quite cryptic.

In our view, the modeling flexibility gained by allowing hybrid models, does not by itself motivate the necessary increase in translator complexity. However, the required capability to flatten (or expand) hierarchical system models can also be useful in other ways. Most importantly, it would enable a user to optionally expand a system model into a single, flat, continuous model, which in turn could be translated directly into a type subroutine. It would also facilitate translation of NMF system models to differential-algebraic equation (DAE) solvers without any submodel support, such as the popular DASSL solver [Brenan 1989]. Finally, it would simplify general symbolic processing of system models, to e.g. eliminate uninteresting variables, or application of index (of nilpotency) reduction algorithms. For these reasons, we have decided to propose a hybrid model type, a continuous_system_model, that mixes equations with references to submodels, which have to be equation based. The heating collector with a slight variation will illustrate the proposed syntax.

```
CONTINUOUS_SYSTEM_MODEL  Heating_Collector_w_eqn

VARIABLES
    HeatFlux        Heat_sup    OUT         "supply power" ;

PARAMETERS
    HeatFlux        my_fluxS_P      100 "a pram" ;

SUBMODELS       //Declare submodel instances
//  class           instance
        // data
    Collector       Coll;
    Heat_Coil   Hcoil,
        rise_time := 30;
        // rise_time is a parameter of Heat_Coil

EQUATIONS
    // Regular equations and NMF assignments
    0 = - Heat_sup + 10*Sin(Time*2*PI/(24*t_scale));
    // may be mixed with link level internal connections
    //  inst.link   =   inst.link;
    Coll.Out        =   Hcoil.In; // (as for pure system models)
    // and arbitrary equations containing submodel
    // variables with syntax (inst . link . var)
    Hcoil.Supply.Q_sup/11.4 = Heat_sup/11.4;

LINKS           // Lifted submodel links or
                // locally defined links may be mixed
    In_1    =   Coll.In1; // a lifted link, as for pure systems
    MTG         In_2    POS_IN Coll.In2.M2, Coll.In2.T2,
                        Coll.In2.W2;
                // the parallel link, not lifted but
                // redeclared. Locally defined variables
                // (such as Heat_sup) may also appear here
    Out =   Hcoil.Out;      //also lifted

DOCUMENTATION
END_DOCUMENTATION

END_MODEL
```

This hybrid model concept is very similar to that proposed by [Sowell 1994]. Similarly to Sowell's proposal, submodels may only interact with each other via variables appearing on the links of the joined models.

The fundamental difference between our proposal and Sowell's original concept is that it enables link level operations as well as variable level operations, i.e. submodel links may be connected or lifted on the link level rather than always on the variable level. In the example, all the variables of the collector Out link are joined to the heater In variables in one operation. Similarly the collector In1 link is lifted and renamed In_1 in one operation. To illustrate the variable level alternative, the In_2 link of the collector has been decomposed into its individual variables, and then (trivially) assembled again. The latter construction would of course allow a redefinition of the link, by for example adding an extra variable.

Connecting and using vector valued links requires a more detailed specification, which is left out here.

3.3 New Function Definitions

In the present syntax (version 3), substructuring is achieved mainly through the definition of functions. The main purpose of external function definitions was originally to provide a gateway to foreign code, and all functions and subroutines had to be coded in a foreign language. In version 3, NMF assignment modeling is allowed in function definitions as an alternative to C and Fortran and this is appreciated by most modellers, who find it convenient to write everything in a single language. Another advantage is that these NMF based function definitions are as accessible as equations for symbolic processing. The limited statement repertoire of NMF could easily be processed both to generate derivatives and to generate alternative inverses of functions. Since function references are very natural in an equation based context, we see this as a superior alternative in many of the examples that have been presented as motivation for hybrid modeling. Compare for example the collector model with a call to a function in the equations (not through help variables) to the suggested syntax with local links.

Our preferred syntax for equation and link sections, referring to a *function* EnthalpyF(W,T) (not a continuous model):

```
CONTINUOUS_MODEL  CollMoistAir

// documentation and declarations omitted

LINKS
//  type        name        variables
    MTG         In1_air     POS_IN M1, T1, W1;
    MTG         In2_air     POS_IN M2, T2, W2;
    MTG         Ivg_air     POS_OUT Mo, To, Wo;

EQUATIONS
    Mo = M1+ M2;
```

```
Mo * EnthalpyF(Wo,To)
    = M1 * EnthalpyF(W1,T1)
    + M2 * EnthalpyF(W2, T2);
Mo * Wo = M1*W2 + M2*W2;
```

The corresponding sections in our proposed hybrid syntax with reference to a continuous_model EnthalpyM:

```
CONTINUOUS_SYSTEM_MODEL  CollMoistAir

// documentation and declarations omitted

LINKS
//  type        name        variables
    MTG         In1_air     POS_IN M1, Ent1.Int.T ,
                            Ent1.Int.W;
    MTG         In2_air     POS_IN M2, Ent2.Int.T ,
                            Ent2.Int.W;
    MTG         Lvg_air     POS_IN Mo, Ento.Int.T ,
                            Ento.Int.W;

SUBMODELS
    EnthalpyM       Ent1;   // Instantiation of
                            // CONTINUOUS_MODEL
EnthalpyM,
    EnthalpyM       Ent2;   // with a single LINK:
                            // HTG Int H, T, W;
    EnthalpyM       Ento;

EQUATIONS
    Mo = M1 + M2 ;
    Mo * Ento.Int.H =   M1 * Ent1.Int.H
                      + M2 * Ent2.Int.H;
    Mo * Ento.Int.W =   M1 * Ent1.Int.W
                      + M2 * Ent2.Int.W;
```

It seems obvious that the hybrid model alternative is considerably more awkward than the first version, where Enthalpy is treated as a function rather than as a submodel. The fundamental difference between a submodel and a function call is that the latter does not have an internal state, i.e. no internal parameters or state variables are associated with each function call. The restriction to only algebraic equations eliminates state variables. Thus, we can only speak of instantiation with respect to submodels and never to functions.

Kolsaker [Kolsaker 1994c] has suggested improved declaration of INPUT and OUTPUT quantities of functions, making them look more like the corresponding declarations in continuous models and to enable unit checking. This seems to be a very sound alternative, although not backwards compatible. Perhaps the old version, which has its merits in terms of simplicity, could coexist.

Going one step further regarding function definitions, we propose to allow algebraic equations among function assignments.

```
VOID FUNCTION AirPsych (P, M, T, W, PSat, H)

DOCUMENTATION
Definition of psychrometric relations for moist air
END_DOCUMENTATION

QUANTITIES
//  type        name    role    description
    Pressure    P       IN      "air pressure" ;
    MassFlow    M       IN      "not used in eq" ;
    Temp        T       IN      "dry bulb temp" ;
    HumRatio    W       IN      "humidity ratio" ;
    Pressure    PSat    OUT     "saturation pressure";
    Enthalpy    H       OUT     "enthalpy" ;

CODE
    // equations and assignments may both occur.
    // Call function to calculate saturation pressure
    PSat := ASHRAE_SaturationP (T) ;
    0 = -H + CP_Air * T + W * (CP_VAP * T + HF_Vap) ;

END_CODE

END_FUNCTION
```

Quantities acting both as inputs and outputs (A_S variables in the calling model), receive role I_O. Local quantities are declared LOC. Variables of any role may appear in equations, and all except IN may be assigned to once, i.e. several assignments of the same variable may occur, if in the same conditional statement and excluding each other. Functions that return a value are defined similarly, but with an output quantity type instead of VOID. Such functions may not in general have I_O variables.

These rules are not in complete harmony with current role definitions in continuous models. Space will not allow a thorough digression on these matters here. However, it is possible (and preferable) to change, in a backwards compatible fashion, the current role definitions in continuous models to be more adequate and at the same time to be in harmony with the definitions above.

The new features we have so far discussed, the system models and the extended function declaration, form a natural set of changes that should satisfy the most imminent needs for hierarchical modeling. However, we would like to go further and address two additional structural problems.

3.4 Property Links

NMF link types are ordered sets of typed variables and normally correspond to physical connections having well defined properties. Examples are fluid flows, electrical connections, control signals (physical or logical). When defining link types, it is desirable to select 'minimal' sets of variables, sufficient to define the media properties relevant for the modeling level of the connection. Dry air flows could e.g. be described, either by (massflow, pressure, temperature) or (massflow, pressure, enthalpy) but preferably not by the redundant set (massflow, pressure, temperature, enthalpy). Component models using these link types will, however, often make use of quantities that have to be derived from the minimal set in the link. An example is a collector with temperatures in the links but expressing enthalpy conservation.

Much of the discussion on NMF extensions has centered on how these (missing) property equations can be handled in component models. We propose that *link property models* be introduced to handle this.

Extended link types will then contain a (minimal) set of variable types, plus a link model defining further variables and their relations (to the minimal set). Component models can optionally use the extra variables by referring to the link model, but without repeating the link model equations. Duplication of equations is thus avoided in the description of component models with common property links. NMF translators will automatically add property equations and variables as necessary in generated component models.

We think that the basic information about such extended link types should be presented in a very compact way among the present global link type definitions, in order to retain the index character of this section. The actual models, which in our opinion should be limited to assignments and algebraic equations, could very well be declared in terms of a new-style subroutine (VOID FUNCTION) definition. The basic declaration among the global link types looks like this:

```
FatMoistAir (   Pressure "air pressure",
                MassFlow,
                Temp,
                HumRatio )
    DERIVED_FROM air_psych
            (   Pressure "saturation pressure",
                Enthalpy ) ;
```

The subroutine air_psych would take the four first variables as input and calculate the remaining two as output. The order of the variables in the formal argument list of air_psych would be the same as in the link type declaration.

The equation and link sections of a moist air collector would then be:

```
EQUATIONS
    Mo = M1 + M2 ;
    Mo * Ho = M1 * H1 + M2 * H2 ;
    Mo * Wo = M1 * W1 + M2 * W2 ;

LINKS
//  type            name        variables
    FatMoistAir     In1_air     P, POS_IN  M1, T1, W1,
                                VOID, H1;
    FatMoistAir     In2_air     P, POS_IN  M2, T2, W2,
                                VOID, H2;
    FatMoistAir     Lvg_air P, POS_OUT Mo, To,
                                Wo, VOID, Ho;
```

3.5 Model Inheritance

Another feature that is missing in NMF is a possibility to exploit similarities between related models. An obvious way to handle this is via inheritance, in the manner of OOP. The most important reasons for an NMF class structure are that we may express information about a compatible family of models in a single place, and that trivial specializations of a model may be expressed separately, e.g. different parameter processing for different component manufacturers etc.

The single inheritance scheme we are suggesting obeys the following rules:

I. New declarations under an NMF heading are added after previous (inherited) declarations

II. Uniquely identifiable declarations may be overridden, except for assignments, which are never overridden.

This means that, for the case of continuous models, equations and assignments may be added to previously defined ones. Overriding previously made assignments is in some cases technically possible, but would result in cryptic code. Redeclaring a previously declared link, variable, or parameter, cancels the previous declaration completely. Overriding will be used mostly for link declarations, to provide increasingly detailed link types.

A new keyword CLASS is introduced. A CLASS (in our definition) may *not* be instantiated. It is used only for structuring purposes, as a repository for class-common definitions.

Multiple inheritance is excluded. Its merits in this context are questionable, and it may result in declaration collisions. These could be resolved by allowing e.g. aliasing of inherited properties or by letting the order of the inheritance list carry meaning. However, since we are striving for simple solutions, our suggestion is that single inheritance is sufficient for the moment.

As in illustration of the technique, we can choose collector models.

```
CLASS CollTemplate

DOCUMENTATION
END_DOCUMENTATION

LINKS
//  type        name        variables
    MT          InAir1      POS_IN M1, T1 ;
    MT          InAir2      POS_IN M2, T2 ;
    MT          OutAir      POS_OUT Mo, To ;

EQUATIONS
    // mass balance
    Mo = M1 + M2 ;

VARIABLES
    MassFlow    M1      IN      "mass flow 1 in"
    MassFlow    M2      IN      "mass flow 2 in"
    MassFlow    Mo      OUT     "mass flow out"
    Temp        T1      IN      "temp of flow 1"
    Temp        T2      IN      "temp of flow 2"
    Temp        To      OUT     "temp of flow out"
```

```
END_MODEL
```

The template is not a complete model by itself; at least enthalpy balance has to be added.

```
CONTINUOUS_MODEL (CollTemplate) DryColl

EQUATIONS
    // add enthalpy balance
    Mo * CP_AIR * To = M1 * CP_AIR * T1
                     + M2 * CP_AIR * T2 ;

END_MODEL
```

The template can also be used to define a collector for moist air.

```
CONTINUOUS_MODEL (CollTemplate) MoistColl

LINKS                  // redefine all three links
//  type      name      variables
    MTG       InAir1    POS_IN M1, T1, W1 ;
    MTG       InAir2    POS_IN M2, T2, W2 ;
    MTG       OutAir    POS_OUT Mo, To, Wo ;

EQUATIONS
    // add enthalpy and humidity balances
    Mo * EnthalpyF(To,Wo)
       = M1 * EnthalpyF(T1,W1)
       + M2 * EnthalpyF(T2,W2) ;
    Mo * Wo = M1 * W1 + M2 * W2 ;

VARIABLES
    // add declarations for humidities
    HumRatio  W1 IN     "hum ratio of flow 1"
    HumRatio  W2 IN     "hum ratio of flow 2"
    HumRatio  Wo OUT    "hum ratio of flow out"

END_MODEL
```

Now, let us try to decompose the Heating_Collector using inheritance.

```
CLASS HC_template

DOCUMENTATION
Common doc's go here
END_DOCUMENTATION

SUBMODELS              // this section is (optionally)
                       // declared here for clarity only
//  GENERIC     instance
    GENERIC     Coll;
    GENERIC     Hcoil;      // no data allowed for
                            // GENERIC

CONNECTIONS
//  inst.link   =  inst.link ;
    Coll.Out    =  Hcoil.In ;

LINKS
//  name  =  inst.link ;
    In1   =  Coll.In1 ;
    In2   =  Coll.In2 ;
    Out   =  Hcoil.Out ;

END_MODEL
```

One possible use of this resource is of course a new edition of the Heating_Collector:

```
SYSTEM_MODEL (HC_template) Heating_Collector

DOCUMENTATION
Suplementary, class specific, doc's are added here
END_DOCUMENTATION

SUBMODELS
//  class              instance
//      data
    Collector          Coll ;
    Heating_Coil       Hcoil
        Q_max := 1000,     rise_time := 30;

END_MODEL
```

A nice feature is that we easily may populate the HC_template with new submodels, which may have entirely different link structures.

```
SYSTEM_MODEL (HC_template)
                   Moist_Heating_Collector

DOCUMENTATION
END_DOCUMENTATION

SUBMODELS
//  class                  instance
//      data
    Moist_Collector        Coll ;
    Moist_Heating_Coil         Hcoil
        Q_max := 1000,     Other_data := 4017;

END_MODEL
```

3.6 NMF Extension keyword

Each implementor of NMF based tools has added extensions as necessary for the targeted environments. This is completely in accordance with the original ideas, provided that the NMF part of a full model description remains meaningful to others, and that extensions can be easily removed, leaving standard NMF only. It has become obvious that it would be an impossible endeavor to strictly define what is truly neutral about a model. The only feasible solution is to construct a working process for the successive inclusion of features of sufficient common interest. In principle, the ASHRAE NMF committee provides a forum for additions to the language but individual users and groups of users need access to quicker, more informal, means.

Kolsaker [Kolsaker 1994a] has to this effect suggested a special pair of keywords EXTENSION and END_EXTENSION on the continuous model level, to delimit whatever information, in a free format, that might be useful in a particular context. The only requirement on the delimited information is that the second token should identify the purpose of the following text and be readable by any NMF translator. Aside from being a good pragmatic solution for individual users, the suggested mechanism would work as

a source of inspiration for formal extension proposals - a gateway for organic growth.

We propose that extensions may occur in a number of well defined places in the NMF code: among global declarations, in models, and tagged on to individual statements. Any front-end NMF translator should then be prepared to store the unprocessed extension code for subsequent parsing in the appropriate back-ends. Since extensions may appear frequently in target code for a particular solver, we suggest an abbreviated keyword pair: EXT and END_EXT.

3.6.1 NMF Samples with possible extension locations

Extension keyword pairs are marked with ⇨; only places of principal importance have been selected.

```
QUANTITY_TYPES
//  Quantity types are used for both variables and
//  parameters. For parameters the kind is irrelevant.
//  CROSS = Potential, non-directional
//  THRU = Flow, directional
//  type name          unit            kind */

    Enthalpy         "J/kg"          CROSS ⇨;
    Factor           "dimless"       CROSS ;
    HeatCapM         "J/(kg K)"      CROSS ;
    HeatCondA        "W/(m2 K)"      THRU ;
    HeatFlux         "W"             THRU ;
    HumRatio         "kg/kg"         CROSS ;

⇨  //Extensions may occur once  at the global level

LINK_TYPES
//typename          (variable types)

TQ                  (Temp, HeatFlux) ⇨;

MT                  (MassFlow, Temp) ⇨;
PMT                 (Pressure, MassFlow, Temp) ;
MoistAir            (Pressure, MassFlow, Temp, HumRatio) ;
```

A sample thermal model of a wall:

```
CONTINUOUS_MODEL tq_hom_wall

DOCUMENTATION
A 1-D finite difference wall model. One homogeneous layer. TQ
interfaces on both sides.
END_DOCUMENTATION

⇨ //Once at the model level

EQUATIONS
    //NMF assignments can always be interpreted as if
    //they were equations

    Taa := 2 * Ta - T[1] ⇨;
    Tbb := 2 * Tb - T[n] ;

    // space discretized heat equation, T' indicates dT/dTime
    FOR  i = 2, (n-1)

        c_coeff * T'[i] = T[i - 1] - 2. * T[i] + T[i + 1] ⇨;
    END_FOR ;

    c_coeff * T'[1] = Taa - 2.* T[1] + T[2] ⇨;
    c_coeff * T'[n] = T[n - 1] - 2. * T[n] + Tbb ;
    // boundary equations
    0 = -Qa + d_coeff * (Taa - T[1]) ;
    0 = -Qb + d_coeff * (Tbb - T[n]) ;

LINKS

//  type        name        variables

    TQ      a_side      Ta, POS_IN Qa ⇨;
    TQ      b_side      Tb, POS_IN Qb ;

VARIABLES
//  type        name    role    description

    Temp        T[n]    OUT     "temperature profile" ⇨ ;

    Temp        Ta      OUT     "a-side surface temp" ⇨ ;
    Temp        Tb      OUT     "b-side surface temp" ;
    Temp        Taa     LOC     "a-side virtual temp" ;
    Temp        Tbb     LOC     "b-side virtual temp" ;
    HeatFlux    Qa      IN      "a-side entering heat" ;
    HeatFlux    Qb      IN      "b-side entering heat" ;

MODEL_PARAMETERS
//  type    name    role    [def min max]   description
    INT     n       SMP     5   3   BIGINT  "number of temp

layers" ⇨ ;

PARAMETERS
//  type        name        role    description
//  supplied parameters

    Area        a           S_P     "wall area" ⇨ ;
    Length      thick       S_P     "wall total thickness" ;
    HeatCondL   lambda      S_P     "heat transfer coeff" ;
    Density     rho         S_P     "wall density" ;
    HeatCapM    cp          S_P     "wall heat capacity" ;
//  computed parameters
    generic     d_coeff     C_P     "lambda*a/dx" ;
    Length      dx          C_P     "layer thickness" ;
    generic     c_coeff     C_P     "rho*cp*dx*dx/
                                     (lambda*t_scale)" ;

PARAMETER_PROCESSING

    dx := thick / n ⇨ ;
    c_coeff := rho * cp * dx * dx / (lambda * t_scale) ;
    d_coeff := lambda * a / dx ;

END_MODEL
```

4. STEP RECONCILIATION

Since the first proposal in 1989, the discussion about various NMF-constructs has focused on grammar, as indeed this paper does as well. The textual appearance of selected models has been the main object. This is of course quite appropriate for the equation core of component models, but for instantiated system models and related data it may be fruitful to also regard data models directly, and to treat the textual representation as one of several possible views. STEP/EXPRESS [ISO TC 184 1993] seems to be an appropriate vehicle for an extension of the future NMF discussion in this direction.

Further reasons for the employment of STEP technology for NMF include:

- STEP dominates product model applications, which will be an important source for generation of simulation models, and these should therefore be encompassed by STEP, either as pure aspect models or as parts of global models
- Many existing STEP/EXPRESS resources will be useful for development of NMF-oriented application tools

In our initial work in this area [Sahlin and Johansson 1994], we have chosen to focus only on the problem of defining conceptual models of NMF instances, and of hierarchical systems of such instances, i.e. data models corresponding to the system_model construct proposed in this paper. The textual description of the data, found in a STEP physical file, would simply be another completely equivalent representation, less appealing to the human eye, but processable by standard STEP tools. In the first set of conceptual models, only the state of a model is encompassed, represented by its quantity values, whereas the internal component behavior, represented by the equations, is not treated. It is assumed that underlying NMF continuous_models are known to all parties.

Another interesting issue is of course the models of internal behavior as well, i.e. to model the present NMF in EXPRESS, with entities such as equation, if_then_else_clause, etc. Such models are necessary for development of NMF parsers and translators.

The main motivation for remodeling the present NMF in EXPRESS is completeness. New component models could be communicated with the same tools and protocols. A potential EXPRESS-based STEP NMF standard would not have to rely on an additional non-EXPRESS standard. Additional benefits can be expected for design and implementation of NMF component model databases and management tools. The present conclusion is that it would be worthwhile to model the present NMF in EXPRESS.

5. CONCLUSIONS

NMF seems to be well on the way of becoming a commonly accepted format for exchange of simulation models. The ASHRAE committee presently in command provides, for the time being, an adequate neutral forum for the development process. We believe that the existence of a standard will be a salient feature in the evolution of object oriented simulation environments, which in turn will provide the building sector with appropriate and powerful simulation tools.

REFERENCES

Brenan, K.E., S.L. Campbell, and L.R. Petzold, 1989 "Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations", North Holland

ISO TC 184 1993. "The STEP Standard", draft international standard DIS 10303, continuously since 1992 published in several different parts

Jeandel A. 1994, Personal communication

Kolsaker, K. 1994a. "NEUTRAN-supported NMF Enhancements", presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Kolsaker, K. 1994b. "Simpler NMF Description of Advanced Models Using Hierarchical Modelling and Data Abstraction", presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994

Kolsaker, K. 1994c "NEUTRAN - A Translator of Models from NMF into IDA and SPARK", Proceeding to the BEPAC conference, BEP'94, York

Lorenz F. 1990. "Brief Description of the MS1 (Modelling System 1) Project", private communication

Lorenz F., 1994, "Comments on the Neutral model Format", presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Nataf J.-M. 1994, "Translator from Neutral Model Format to SPARK", draft paper presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE winter meeting 1994

Pelletret, R. 1994, Personal communication

Sahlin, P, E.F. Sowell 1989, "A Neutral Format for Building Simulation Models", proc. of Building Simulation ´89 , Vancouver, BC, International Building Performance Simulation Association

Sahlin, P., A.Bring, E.F.Sowell 1994, "The Neutral Model Format for Building Simulation", Version 3.01, ITM report 1994:4.

Sahlin, P., C. Johansson, 1994 "NMF-Based Aspect Models in STEP for Building and Process Plant Simulation", proc. of the CIC W78 workshop on computer integrated construction, Aug. 22-24, 1994, VTT, Helsinki, Finland

Sahlin, P. 1995. Dec 94 - Feb. 95 "Progress Report on TRP-839 Development of a Component Model Translator for the Neutral Model Format (NMF)", *Report submitted to ASHRAE*

Sowell, E.F. 1994. "A Proposal for Hierarchical Submodels in NMF", presented to the TC 4.7 NMF Ad Hoc Subcommittee at the ASHRAE annual meeting 1994