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, HS1). 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. hyste resi s.

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 discrete 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.

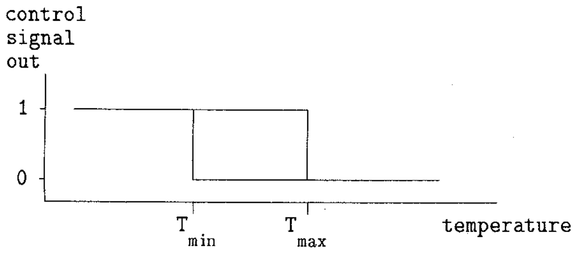
4. DISCONTINUITY HANDLING

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.

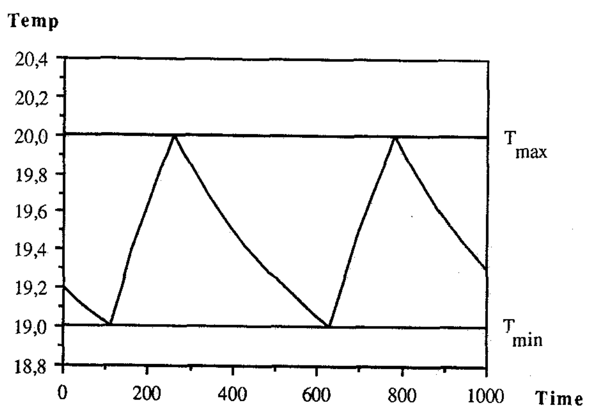
Equations in a wholly continuous model do not have to be written in any particular order, the system is solved simultaneously. The situation changes, however, when assignments are included in the model. Assignment statements are ordered sequentially with respect to each other and the order is normally 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 reportoire 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:



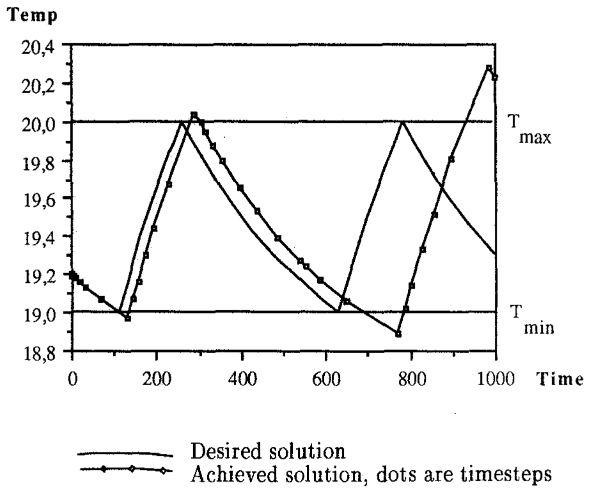
Look at the situation when the rising temperature T approaches T_{max}. Any timestep long enough to let T reach the limit will leave T undefined: If we try with $T > T_{max}$ the heating will be off and T will not reach T_{max}; if we try with $T < T_{max}$ the heating will be on and T should exceed 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.

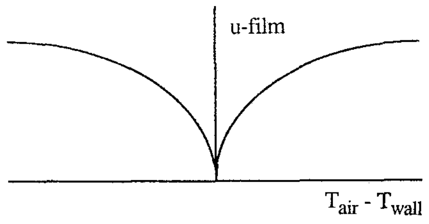
Choosing a suitable step for a solver with fixed time-step is mostly done by guesswork since a proper control can only be achieved by varying the step systematically until a picture emerges of how accuracy depends on timestep.

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 time-steps.

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.