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 ftont end ofIDA, a general environment for building and eneqy 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, such as optimisation algorithms requiring repeated simulation runs.

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

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.

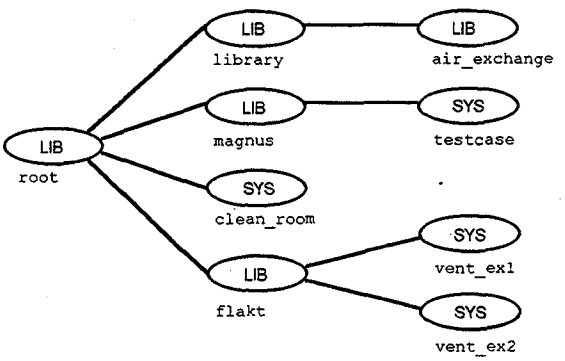

Figure 1. A selection of macros.

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 presentation can also be enhanced with figures, built with polylines and arcs.

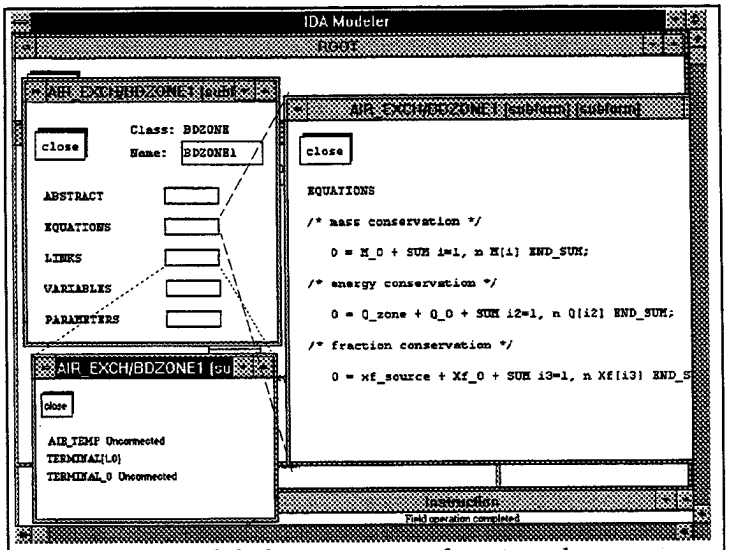

Figure 2. The default presentation of an air exchange zone model, with two open subforms, showing some details of the model

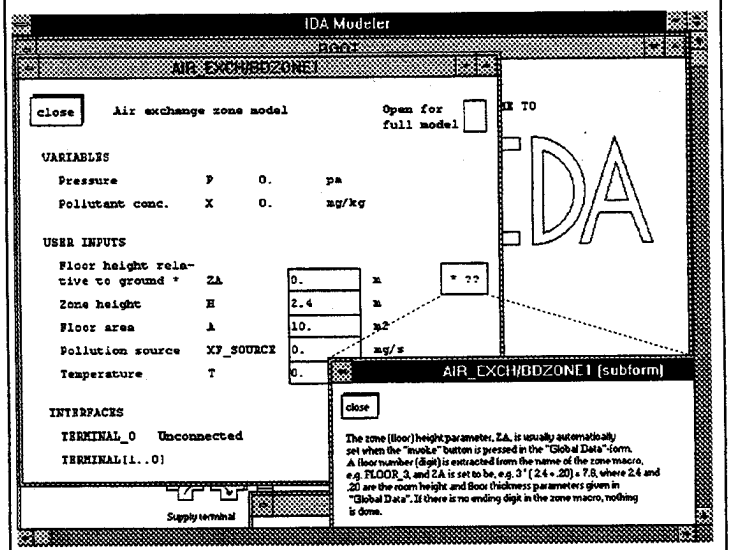

Figure 3. A tailored form for the zone model, with a supporting subform open.

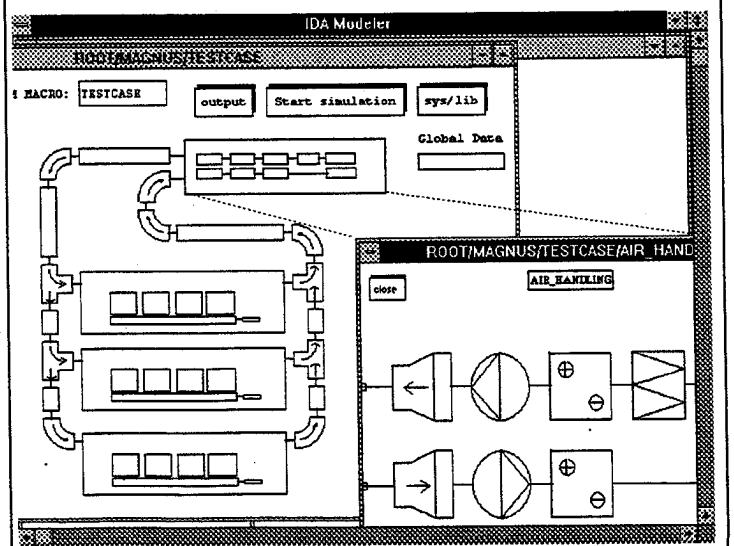

Figure 4. An air exchange building model (a system macro) with one open submacro.

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 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 mod-

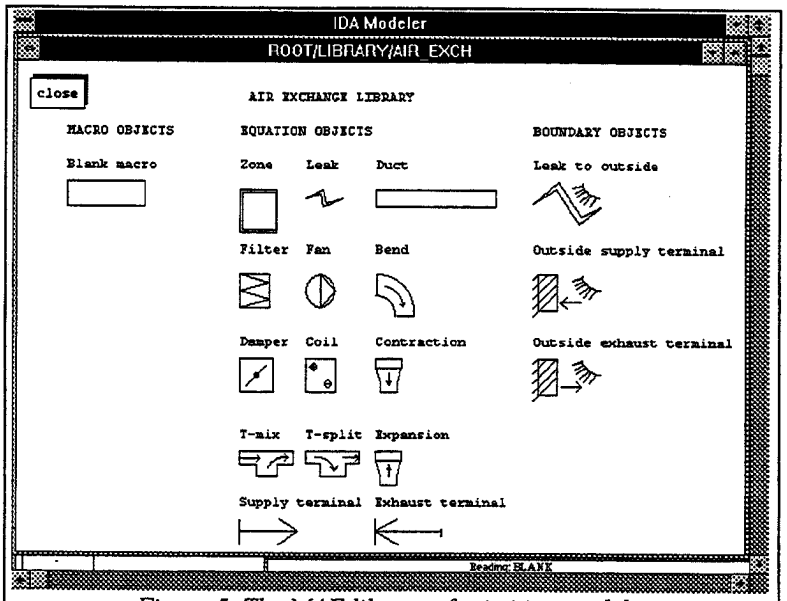

Figure 5. The MAE library of primitive models.

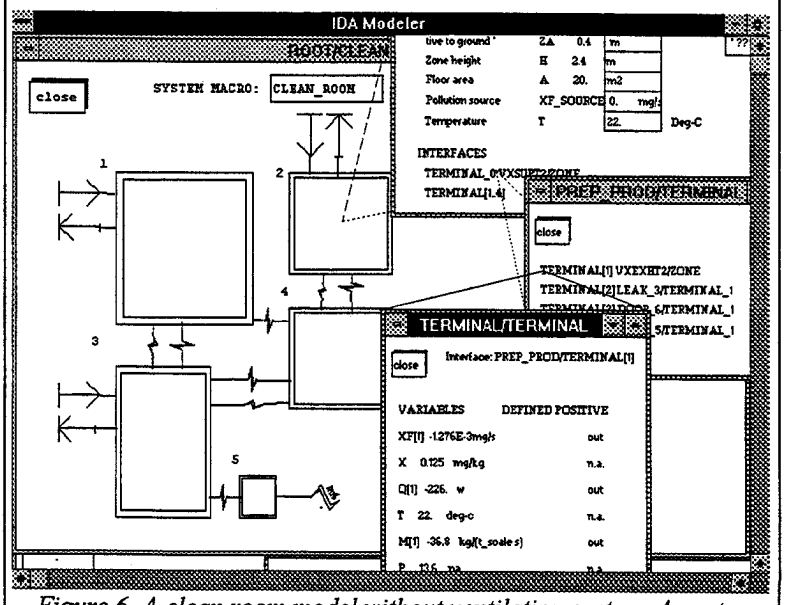

Figure 6. A clean room model without ventilation system. A vector terminal and one of its components has been opened.

els 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.

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.

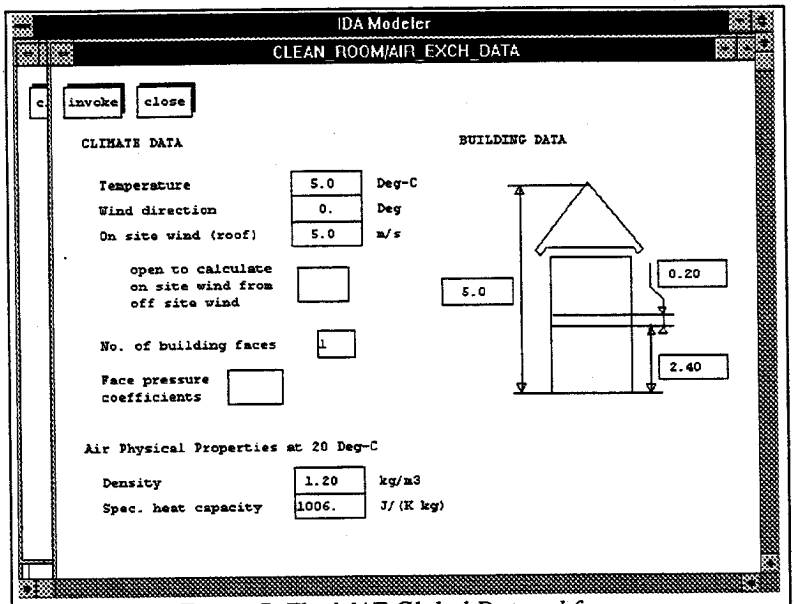

Figure 7. The MAE Global Data subform.

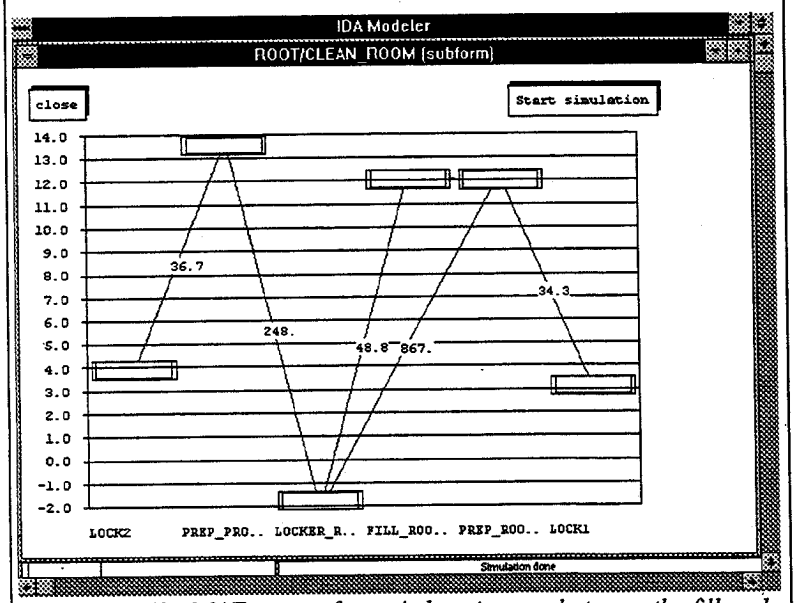

Figure 8. The MAE output form. A door is open between the fill and prep rooms, hence the slight difference in pressure. Zone models can be opened and parameter runs carried out directly from this form.

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.).