MODELLING PLATFORM WITH MULTIPLE REPRESENTATION FORMALISMS.

Ir. Francis LORENZ
LORENZ CONSULTING sprl
Clémodeau, 196a
B-4550 VILLERS-LE-TEMPLE, BELGIQUE.

ABSTRACT

Because every description formalism has advantages and disadvantages, a modelling platform for ODE/DAE systems allowing several formalisms would be required. MS1 is such a program and is described in this paper. Its development will start in August 1991 and will last three years.

MS1 is based on an internal normalized formalism that is actually a mere topological representation of mathematical equation sets. we call that formalism information networks . computational causality is assigned after subsystem assembly and determines the assignment form to be used in each specific case.

The program structure is designed so that it is easy to add a new input formalism - i.e. a description language - or a new output formalism - i.e. to address another solver.

INTRODUCTION.

We call *platform* a set of successive software layers that form together an incomplete system i.e. with the top layer(s) missing. Now, the paper describes a *modelling platform for ODE/DAE systems* . Its outstanding peculiarity is to allow several "user" representation formalisms in a compatible manner. Moreover, these representation formalisms may be generic (block-diagrams for instance) as well as specific (hydraulic networks for instance).

This platform is obviously not specific to building physics, but general to simulation. Its philosophy could even be extended to many other domains of software design. It is presented within the frame of a building simulation conference because this field of activity is at the leading edge of the science of simulation. It is the author's belief that this is due to the known complexity of the problems building designers have to cope with. It is probably not by accident that many advanced modelling concepts emerged in the building simulation community.

An example is given by the related concepts of "modelling by composition" (Laret 1989) and "modélothèque" (Dubois 1988) developed by CSTB in France. Clearly, one of the most difficult questions these researchers have to deal with concerns the choice of a formalized (i.e. machine usable) representation language (Dubois 1990, pp.III.24-28). It should potentially be able to represent all the problems and acceptable to all the researchers and practitioners. This sounds like an unachievable task. Another track consists in allowing several formalisms but the question arising concerns their compatibility. The platform described in this paper would partially answer these questions.

The development of this platform will start in August 1991 and will last three years. This project is not academic. The characteristics of the planned program are based on several years of research (Lorenz 1986 ; Lorenz and Cornet 1986) and their feasibility is well-established. It is time now to bring all this to the field-experienced people i.e. to market it.

The program has temporarily been named MS1.

OVERVIEW OF MS1.

Outstanding Characteristics.

MS1 heart is basically a *modelling engine* . This view is illustrated by figure 1 and explained in the next sections.

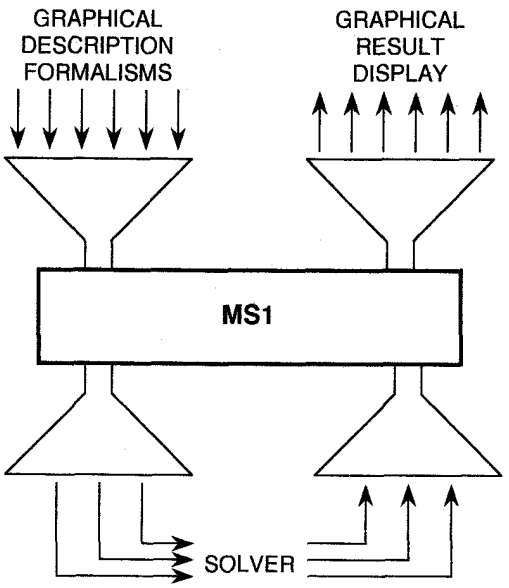

Figure 1. MS1 as modelling engine

The most important and innovative characteristic of MS1 lies in the fact that it is by no way bounded to a single model description formalism (for instance block-diagrams, bond graphs, electronic schematic diagrams, etc...) Actually MS1 can be adapted quite easily to any graphical modelling formalism (topological representation formalisms). How this can be done will be explained later. This feature is called *input multiformalism* .

This characteristic comes from our deep conviction that every formalism has advantages and disadvantages (Lorenz 1987). An endless discussion to convince people of the superiority of one single formalism is at least useless. Individual preferences do exist and do have an influence on the model quality, whose improvement is the ultimate objective of using a modelling program. Moreover we do think such a discussion is also meaningless because the choice of the "best" formalism can never be general but depends on the problem under study.

One should understand not only that several formalisms can be used separately in MS1 but also that they can be used in conjunction with each other to describe a single model i.e. they are *compatible* . In other words, one is allowed to design different subsystems using different formalisms and then to assemble these subsystems together. This opens the door to the highest level of reusability and to the build-up of subsystem libraries.

One should also understand that when we speak of different formalisms, we do not only address generic topological languages (like block-diagrams) but also specific topological languages (like hydraulic schematic diagrams). It is the reason why MS1 must be considered as a modelling engine, potentially adaptable to many applications.

The second most important and innovative characteristic of MS1 is quite similar to the first one. We call it *output multiformalism* . MS1 is not bounded to a single solver but can be adapted to several different solvers. The relevant method will be explained later.

The reasons for inserting this feature are also quite similar. Again, the various available solvers do have their own advantages and disadvantages so that no general choice can be made. The choice of a solver always depends on the characteristics of the problem under study plus local availability, support and maintenance of the product as well as user's preference. It is therefore much preferable that the modeller be able to access several solvers.

The result display module is of course adapted to this situation and is able to take into account the input formalism in its presentation interface. Actually, one mode of presentation will make extensive use of the model drawing that has been entered by the user : it is the symbolic animation module.

Other Features.

In addition to input and output multiformalism, MS1 has also the following important characteristics :

- *Modular representation* . As already mentioned, any system can be made by assembly of subsystems, which may consist of an assembly of sub-subsystems, and so on with no software limit to the level of decomposition. Moreover, the subsystems to assemble may be written using different formalisms (assembly rules do exist and must be observed, of course).

- *Library* . A library mechanism allows to store in a structured way and retrieve frequently used subsystems with special access restrictions. This feature gives all its power to modularity.

- *Mixed DAE* . As far as the modeller is concerned, mixed differential algebraic equations are represented. Their resolution is however possible only if the selected solver allows it.

- *Discontinuities* . Once again, discontinuous systems are considered but can be solved only if the selected solver has this capability.

- *Symbolic animation* . Together with table outputs and conventional diagrams, animation usefully complements the means one has to understand the results, each of these means pointing out different aspects. Of course, a "cartoon" animation cannot be performed on topological graphs. That is why we speak of symbolic animation.

Working Environment.

The choice of the environment is governed together by marketing and technical considerations. *Workstation* hardware has been selected as best choice for such an interactive program. The software environment must satisfy industrial standards i.e. *UNIX* operating system. For the same reasons graphs are under control of *PHIGS* and *X-WINDOW* . Finally, the planned features can be obtained efficiently only with the help of an object oriented language. This will also ensure better maintainability. Industrial standard once more sets the choice on *C++* .

MS1 FUNCTIONAL DESCRIPTION.

Normalized formalism.

The key point of MS1 is a specially designed formalism which entire system is based on. This *normalized formalism* - we call it *information networks* (Lorenz 1989) - is basically a topological representation of mathematical equation sets. This means that the representation does not assume the way the equations will be used i.e. they truly represent the unoriented mathematical relations and not assignment statements. Now, the representation of a specific system can quite easily be extracted from its equation set but also

from its representation in any topological formalism ; MS1 does.

Let us take the trivial example of a linear resistor (Mattsson 1989). Ohm's law states $\Delta V = RI$, where ΔV is the voltage drop across the resistor, I is the current through the resistor and R is the resistance. If we should write the model on assignment form there are two possibilities

$$\Delta V := RI$$

$$I := \Delta V/R$$

Which one must be chosen in a specific case ? This is a decision that cannot be taken without knowing something about the rest of the circuit. If the resistor is simply connected to a current (resp. voltage) source and ground, the first (resp. second) possibility must obviously be taken. The decision is more difficult in most real circuits. This problem is well-known nowadays. It has been studied for many years by linear graph theorists under the name of *normal tree selection* and by bond graph theorists under the name of *computational causality assignment* (Wellstead 1979).

The information network representation of a linear resistor is given on figure 2. It is not very different from a block-diagram representation, except that the arrows do not imply the actual computational causality : the calculation flow is allowed to "pass through" the GAIN element in either direction. In other words, the arrows are there only to indicate whether one must multiply or divide by R for a given calculation flow. The causality assignment procedure will determine which of the two forms will be actually retained in a specific case, after assembly of the whole circuit.

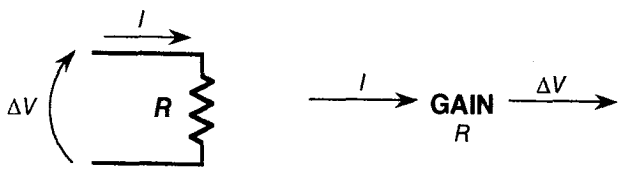

Figure 2. Information network model of a resistor.

Another example is given on figure 3. The SUM element is an algebraic additioner i.e. it implements the relation $\sum Sin = \sum Sout$. It may have any number of input and output bonds, including zero input bond "*XOR* " zero output bond. Which of the signals will be actually computed as signed sum of the others ? This will again be determined by the causality assignment procedure. This contrasts with the block-diagram additioner, which has exactly one output bond that is always calculated as sum of the several input bonds.

The information networks are thus a superset of the classical and well-known

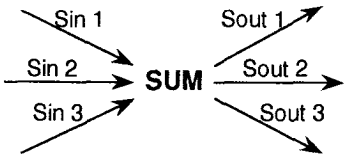

Figure 3. Information network "SUM" element.

block-diagrams. However, they bear much more powerful possibilities.

Model Construction Phases.

MS1 first allows the user to describe the system under study using any of the supported description languages. This is the *input* phase.

It translates upon request all the input descriptions into information networks during the *normalization* phase. This is how different formalisms are equally usable in MS1. Note that the information networks remain internal to MS1 i.e. the user is not even aware of its existence.

As soon as all the subsystems are represented using a common formalism it is easy to check that the connection rules have been observed and eventually to assemble the subsystems. We call this the *assembly* phase. This is how the various input formalisms are made compatible.

Once the system is assembled algorithms are activated during the *check* phase that verify the completeness and coherence of the model. They are also able to detect some potential numerical problems, not all of them. They ultimately determine a possible data flow pattern (not necessarily unique) i.e. they choose a valid ordered assignment form for the model. This last operation is called *computational causality assignment* .

All the variables, parameters, initial computations and equations are then organized in lists ready for simulation code generation. This is the *generation* phase.

Some or all of these lists are finally output in the required order and surrounded by the adequate keywords and other typographic details pertaining to each specific solver. This is the *output* phase and it is how MS1 can address various solvers.

The results produced by the solver in a file are read during the *retrieval* phase and used together with the rest of the information during the *result display* phase.

MS1 ORGANIC DESCRIPTION.

Program Structure.

A program module actually corresponds to each phase of the work. Refer to figure 4 to see how these program modules are organized.

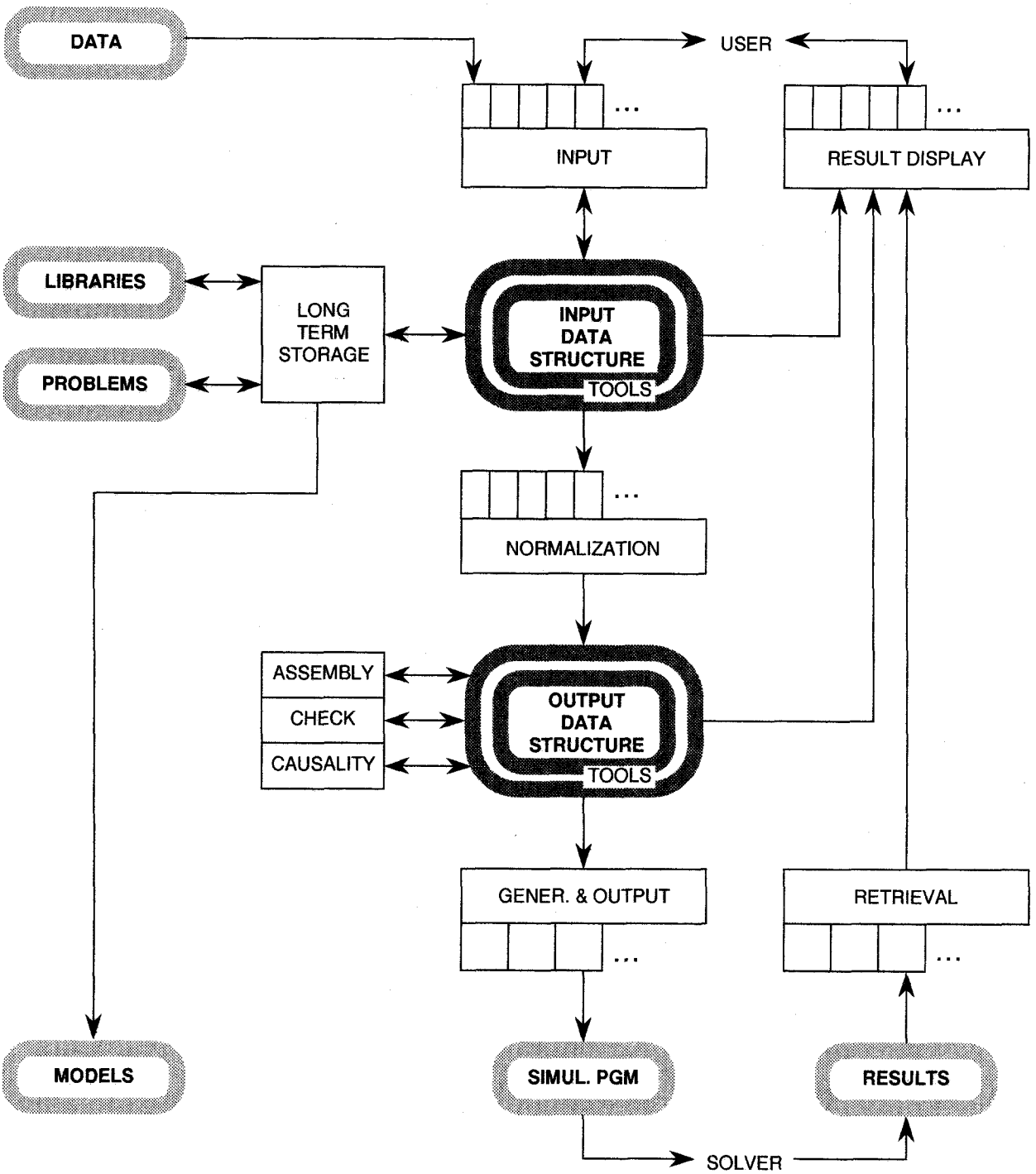

Figure 4. MS1 structure.

One new module also appears in this figure : it is the *long term storage* module. It deals with *subsystem libraries* (private or public sets of commonly used subsystems) but also with *problems* (user's rough input data, generally including several unchecked systems) and *models* (checked and truly workable systems, for which simulation programs have been generated, extracted from the problem data base).

The dual data structure can also be seen in figure 4. The *input data structure* contains unnormalized descriptions and is used for communication with the user i.e. input and

result display or with another program i.e. file input. Long term storage is also performed from this input data structure. The *output data structure* is produced from the input data structure by the normalization module. It contains normalized descriptions and is used for assembly, check, computational causality assignment, generation and output.

Adding an Input Formalism.

Several generic input formalisms are included in MS1 from the very beginning and are the following : block-diagrams, signal flow graphs, linear graphs, bond graphs and compartmental models.

Adding a new generic or specific formalism is quite simple.

Each of the formalism elements, i.e. *vertices* and *edges* as we speak of topological formalisms, is represented as a *class* (object oriented version of a data type). The *graphical representation* of each of these elements is represented by *methods* (object oriented version of subroutine) to be used by the input and result display modules. An *alphanumeric representation* is also represented by methods for file input. This defines the *syntax* of the new formalism.

The *translation rule* of each element is represented by *methods* to be used by the normalization module. This defines the *semantics* of the new formalism.

Finally, *methods* are also designed to specify the *result representation* . They have to do with both syntax and semantics. They actually determine a kind of *extension* to the formalism.

Adding a formalism can be made entirely by the MS1 development team but it can equally be made by the user himself supported by the MS1 team. This could be the case for instance when a research team wants to concentrate their efforts on designing a new modelling formalism and its interface and not on more conventional tasks like library management, subsystem assembly and simulation code generation. In this sense, MS1 is (nothing more than) a high level programming environment.

Adding an Output Formalism.

The output formalisms that will be initially included in MS1 are not yet definitively chosen. One formalism issued from the CSSL family or its recent extensions is probably worthwhile. We will also select another solver, not based on the CSSL norm. Finally a parallel language like OCCAM, although it is not a simulation language, could open the way to parallel architectures for large-sized problems.

Adding an output formalism is still more simple than adding a new input formalism.

One has to just define *methods* for *list selection and ordering* to be used by the output module (we expect all the computation lists to be used, but conversely some formalisms do not require parameter and variable declarations). *Methods* will also deal with *keywords* and *typographic details* .

These extensions can still be made by the MS1 development team or by the user himself, supported by the MS1 team. For instance a very specific solver may be imposed to him for quality assurance reasons.

CONCLUSIONS.

Program MS1 will be developed based on 14 men-years of research and development. Most of its features have already been made, at least as mock-up modules, so that their feasibility is proven. The challenge is now to market these advanced features. This requires severe methods of development to ensure the highest standard of user-friendliness, transportability and maintainability. The working environment has been chosen with these objectives in mind. The rest is only a question of time.

Although already interesting with the default options that have been described, MS1 will take all its true power from its various applications i.e. it will really be enriched by the variety of utilization enforced by the users. Cooperation is therefore encouraged. It is expected to take place in the areas of specific input and/or output formalisms and interfaces and subsystem libraries, but any other suggestion is welcome.

REFERENCES.

Dubois, A.M. 1988. "A Draft of a Component Model Library Manager, the Modélothèque : Analysis of an Approach and First Results." In *Proceedings of the 3rd International Symposium on System Analysis and Simulation* (Berlin, September 1988).

Dubois, A.M. 1990. "Eléments de Spécification d'un Environnement Avancé de Modélisation et Simulation. Application à la Thermique du Bâtiment." Thèse de Doctorat es Sciences, Université de Nice - Sophia-Antipolis, septembre 1990.

Laret, L. 1989. "Building and HVAC Simulation: The Need of Well-Suited Models." In *Proceedings of the IBPSA International Conference : Building Simulation '89* (Vancouver, June 1989).

Lorenz F. 1986. "Bond Graphs Revisited : A Systemic View." In *Complex and Distributed Systems : Analysis, Simulation and Control* , S. Tzafestas and P. Borne, eds. North Holland, Amsterdam, 55-59.

Lorenz, F. and A. Cornet. 1986. "Pluridisciplinary Systems Modelling." In *Proceedings of the 2nd European Simulation Congress* (Antwerp, September 1986) SCS, Ghent, 52-56.

Lorenz, F. 1987. "Reflections about Representation Methods." In *Proceedings of the Ispra Workshop on Future Building Energy Modelling* (Varese, November 1987) Ispra JRC, Varese.

Lorenz, F. 1989. "Acausal Information Bonds in Bond Graph Models." In *Proceedings of the Symposium AIPAC'89* (Nancy, July 1989) IFAC, 302-306.

Mattsson, S.E. 1989. "On Modelling and Differential/Algebraic Systems." *Simulation* , n°. 1 (January), 24-32.

Wellstead, P.E. 1979. *Introduction to Physical System Modelling* . Academic Press, London.

BIOGRAPHY.

Francis Lorenz is currently managing Lorenz Consulting, a company specialized in technical software. Prior to founding this company in 1990, he managed during 6 years a Simulation Cell for CSC sa, the Belgian subsidiary of CSC Consulting, where he was mainly involved in the development of continuous system modelling tools as well as a variety of consulting missions. Before this, he also spent 6 years at the University of Liège (Belgium) as research engineer. Work included simulation of building and HVAC equipments and simulation of robot actuators. He is still lecturer at this University.

Mr. Lorenz received a B.S. in Engineering in 1975, an M.S. in Mechanical and Electrical Engineering in 1978 and an M.S. in Computer Science Engineering in 1983, all from the University of Liège.

He is member of SCS and IMACS and a founder member of IMACS TC 16 Technical Committee on Bond Graph Modelling and Simulation. He is also very active in the attempt to found a French Speaking Simulation Society, which will join EUROSIM (Federation of European Simulation Societies).