An Environment for Model Development and Simulation

Mattsson, Sven Erik; Andersson, Mats

1989

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-3205)/1-030/(1989)

An Environment for Model Development and Simulation

Sven Erik Mattsson
Mats Andersson

STU projects 87-02503, 87-02425

Department of Automatic Control
Lund Institute of Technology
September 1989

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Final Report

Date of issue
1989-09-30

Document Number
CODEN:LUTFD2/(TFRT-3205)/1-030/(1989)

Author(s)
Sven Erik Mattsson
Mats Andersson

Supervisor

Sponsoring organisation
The National Swedish Board of Technical Development (STU contracts 87-2503, 87-2425)

Title and subtitle
An Environment for Model Development and Simulation

Abstract

This is a final report for the project "Tools for model development and simulation" (STU projects 87-02503, 87-02425) carried out in the period July 1987 to June 1989. The project is the last part of the STU supported research program "Datorbaserade hjälpmedel för utveckling av styrsystem (Computer Aided Control Engineering, CACE)", which started in the end of 1984. The main result is a proposal for a kernel for model representation. The kernel may serve as a central model data base in an integrated environment for model development and simulation. The CSSL definition from 1967 has had a profound impact on simulation and has served very well for over 20 years. It is perhaps now time to capitalize on the enormous development of information technology and reconsider the foundations of model representation. The proposal is a modest effort in this direction. If we could agree upon a common set of ideas we may lay the foundation to a new standard. The proposed kernel supports a modularized and object oriented representation of models to allow flexible and safe reuse of model components. The model developer may supply extra information which is used for automatic consistency analysis to check for unintended abuse of models. The kernel can allow any logical and mathematical framework such as differential-algebraic equations, difference equations, etc. to describe behaviour, but a basic idea is that behaviour descriptions should be declarative and equation based. The kernel allows integration of different customized user interfaces. A prototype of the kernel is implemented in Common Lisp and KEE. A new STU-supported project to implement the kernel in C++ has been started. The project has also included an application study focusing on modelling of chemical processes.

Key words
Computer aided control engineering; computer aided system design; modelling; simulation languages; hierarchical systems; system representations; data structures.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
30

Recipient's notes

Security classification

Contents

1. Introduction 4

2. Models Are Essential 5

3. Support of Model Development 7
3.1 Requirements 7
3.2 Basic ideas 7

4. Model Structures 10

5. Object-Oriented Representation 13

6. The User Interface 16

7. An Application Study 19

8. Conclusions 20
8.1 Results 20
8.2 Development and technology transfer 21
8.3 Experiences of software techniques and tools 22

References 25

A. Published Papers and Conference Contributions 27

B. Reports 28

C. Lectures 29

1. Introduction

This is a final report for the project "Tools for model development and simulation" (STU projects 87-02503, 87-02425) carried out in the period July 1987 to June 1989. The project is the last part of the STU supported research program "Datorbaserade hjälpmedel för utveckling av styrsystem (Computer Aided Control Engineering, CACE)", which started in the end of 1984.

Automation and advanced control are of strategic importance for the Swedish industry. There are examples in the whole range from traditional process industry and power generation to aero- and astronautics. To be able to develop and operate high performance systems, computer based tools for model development, simulation, analysis, design, validation, operator training, production planning, operator support, supervision, fault diagnosis etc. are needed.

Today's CACE tools have proved to be useful. However, they were designed 10–20 years ago. The computers had then moderate computing capacity and primitive hardware for graphical input and output. The main function of the tools is to perform extensive numeric calculations and present the results in the form of simple plots. The users want to have tools that better support their needs: user interfaces which support their way of thinking, integrated environments supporting all phases from specification and design to operation and maintenance etc. The enormous development of the information technology (workstations, object-oriented programming, computer graphics, artificial intelligence, expert system techniques, computer algebra etc.) has opened possibilities to improve the CACE tools significantly. The goal of the CACE project has been to

1. investigate how the CACE tools can be improved
2. develop prototype tools
3. establish international contacts

In the first phase of the CACE project a number of pilot projects investigated some ideas and the potential of computer graphics, computer algebra and expert system techniques. Prototype tools, which can demonstrate ideas and principles were also developed. These projects showed that it indeed is possible to improve the tools. A summary can be found in Mattsson (1987).

For the last phase of the CACE project it was decided to focus on tools for model development and simulation (Mattsson, 1987). The motives were:

1. It is of importance for all kinds of engineering.
2. It contains most of the important issues for CACE.
3. It fitted well in the international collaboration.

An important conclusion from the pilot projects was that model representation is a critical issue. The system concept is fundamental in control engineering, but today's tools have only primitive representations, which do not support the users' perception of systems. Furthermore, a common basic representation is needed to make the CACE tools integrated.

The work in last phase of the CACE project has largely followed the research program. A major result is a design proposal for a kernel for model development and simulation. The proposal may be of interest for all users of

models. A basic idea is to support reuse of models so that models can be used for different tasks and so that it easy to modify a model to describe a similar system, since it is difficult and laborious to develop new models. By a kernel we mean the routines to manipulate the internal representation. In our design there is a clear separation between user interface, internal representation of data and models and processing tools. This separation makes the design more flexible and allows customized user interfaces. The kernel can be viewed as a central model data base in an integrated environment for model development, simulation, analysis, design, documentation etc. A prototype implementation of the kernel as well as a user interface has been written in Common Lisp and KEE. The project has also comprised an application study focusing on modelling of chemical processes to get some evaluation of the ideas.

This report is organized as follows. In Chapter 2 motives for supporting model development are given. Chapter 3 outlines basic ideas and our approach to support model development. Chapters 4 and 5 describe the kernel in some detail. Chapter 6 is about user interfaces and Chapter 7 is about the application study. Chapter 8 contains the conclusions. Appendices A – C list published papers, conference contributions, other reports and external lectures given by CACE group members.

2. Models Are Essential

The reason for supporting model development is that

1. models are essential in all kinds of engineering and
2. model development is difficult and time consuming.

It is a well-known fact that it is difficult and time-consuming to develop a new model and we will discuss approaches to support model development in the next section. Let us now motivate why models are needed.

What are the uses of models?

Models are useful in all phases of a systems life from design to operation and maintenance. The designer can use a model to simulate and to analyse the behaviour to learn about the system and to get insight in its behaviour and to validate his design. He can try various system architectures or configurations to make the best choice. He can use optimization tools to tune system parameters. Models are needed in simulators for education and training. Computer based tools for production planning, online optimization, operator support, supervision and failure analysis need models of the system.

Note that modelling and simulation are closely connected to each other. To simulate you need a model. Realistic models are typically non-linear, which implies that it is difficult to analyse a behaviour in other ways than through simulations. However, with a modelling language clearly separated from calculation and simulation issues, models can be used in a more general context for process documentation and to preserve design knowledge.

Why are models needed?

All mathematical methods need some kind of model of the system under consideration. If we do not want to use mathematical methods and models when making a new design we have to make trial-and-error experiments on real equipment. It may be unfeasible to make experiments on real equipment for complexity, performance, safety and economic reasons. First, the system to be designed may have to be so complex that it is impossible to come up with any reasonable design from trial-and-error experiments. Second, to achieve high performance the system must be optimized, but it is in practice impossible to tune more than three coupled parameters by trial-and-error. Third, safety regulations may forbid real experiments, or require validation of the design for extreme and emergency conditions and it may be dangerous or impossible to perform this validation by real experiments. Fourth, real experiments are often expensive and time consuming to perform. Furthermore, when redesigning a plant, it may not be allowed to disturb the operation of the existing plant.

Power generation, aero- and astronautics are typical areas where advanced mathematical methods have been used for a long time to handle complexity, performance and safety issues.

Fierce competition is an important force to use advanced mathematical methods to make better and cheaper designs, and to use computer based tools for production planning, online optimization, operator support, supervision and failure analysis to increase productivity and quality and to decrease production and maintenance costs.

Requirements on saving energy and raw material as well as avoiding environmental pollution make the designs more complex, since the system must contain recirculation loops to win back energy and material. Recirculation loops introduce interactions between various parts of the process implying that it is impossible to design and to operate them independently of of each other.

More specific motives for using advanced mathematical methods for design and in particular control design can be found in Anon (1987) and Fleming (1988). The US Department of Defense has picked simulation and modelling technology as one of 22 critical technologies, since it can reduce design and production costs, improve performance and maintenance, train personnel. Simulators for education and training have been used for a long time in power generation, aero- and astronautics. The interest from other industry areas to use simulators for education, training and operator support is large today. STU has a special research program DUP to investigate how process operators' tasks can be supported by computer based tools. A large part of this program is devoted to simulators.

3. Support of Model Development

In this chapter we will first indicate requirements on concepts and tools for model development and then outline our approach.

3.1 Requirements

Since models are important and since it is difficult to develop new models, a basic question is how computer based tools can support model development?

Reuse

The best way is of course to be able to provide the user with the desired model directly and automatically. This implies model libraries and reuse of models. There are three facets of reuse:

1. *Various purposes or calculations.*
   Models are needed in all mathematical methods and it should be possible to use a model for various purposes without having to recode it manually.
2. *Similar systems.*
   It should be easy to adapt a model to describe a similar system.
3. *Different users.*
   The user interface should preferable be customized and adapted to understand and use the user's concepts and terminology.

New models

A model can be developed using first principles or by analysing measured data. Our project have mainly focused on the first approach.

When developing a new model, decomposition is needed to handle complexity. It should also be possible to extract and reuse parts of existing models. There should be tools that tune model parameters from measured data.

3.2 Basic ideas

Our proposal is based on four main ideas

1. declarative models
2. structured models
3. automatic consistency checking
4. customized user interfaces

Declarative models

Models developed to be used in one package today cannot be used in another package without additional work. Unfortunately, much "model development" work of today consists of manual recoding or implementation of adapters.

An obvious reason is of course that there is no common agreement on the representation of models.

Another maybe less obvious reason is that the representations used in most of today's CACSD and simulation tools are too specialized and of too low a level to allow reuse of models for other tasks than simulation. Today's most used languages for continuous simulation (ACSL, CSMP, CSSL IV, EASY5 etc., for overviews see Kreutzer (1986) and Kheir (1988)) follow the CSSL definition (Strauss, 1967). These tools solve problems of the type $dx/dt = f(t, x)$ if the user defines a Fortran-like procedure which calculates $f(t, x)$.

To allow a model to be used for different purposes, it should be declarative and not procedural. It should describe facts and relations (equations) and not be a calculation procedure. A declarative model is multipurpose, since it can be manipulated automatically to generate efficient code for simulation, code for calculation of stationary points, linear representations, descriptions which are accepted by other existing packages etc. Models of controllers can be used for automatic generation of the control software or to generate layouts for special purpose analog or digital VLSI circuits which implement the controller.

A declarative model is usually also closer to the model developer's perception of the physical reality, and therefore, development of new models is easier. When developing a model from first principles for a physical system one uses fundamental laws as mass balances, energy balances and phenomenological equations. Model development as well as documentation are facilitated if the user can enter these equations as they are without having to transform them into a computational procedure. The risk of introducing errors during manual transformation is reduced. The natural declarative form for continuous time models are Differential-Algebraic Equation (DAE) systems, $g(t, \dot{x}, x) = 0$. An overview of important properties can be found in Mattsson (1989a).

The kernel can allow any logical and mathematical framework such as differential-algebraic equations or difference equations to describe behaviour, but a basic idea is that behaviour descriptions should be declarative and equation based.

Structured models

To understand large models and to be able to reuse parts of models, good structuring facilities must be supported. A powerful modularization concept supports model development by beating complexity as well as it allows reuse of parts and building of models by putting together existing components.

Block diagrams is a common structuring tool. A block represents a submodel. The connections between the blocks show cause-and-effect relationships between inputs and outputs of the submodel. A connection is unidirectional saying that the value of an output should be calculated from the input connected. It means that the model developer must deduce the computational causality to define what are inputs and outputs to a submodel.

When making a model library, it is very inconvenient to define which of the terminals of a submodel that are outputs, because what are inputs and outputs of a submodel is not only a property of the submodel itself, but also of how it is used. As motivated above a model should not be a computational procedure. It should not be a procedure which can calculate the outputs when the values of the inputs and the internal state are given. The model development and simulation tools must be able to handle interactions with unspecified computational causality (non-directional interaction). A con-

nection should only define a relation saying that two terminals A and B are equal, not define a compute statement $A := B$ or $B := A$. Ideas like this have been developed in connection with special purpose simulators. An example is SPICE (Nagel, 1975) for electrical circuits where the basic building blocks are four poles.

In a simulation language like CSSL (Strauss, 1967) model decomposition is handled by macros, which require specification of the causality of the interaction. Another drawback with the macro concept is that the model structure is not preserved at compilation. The macros are expanded at compilation and at simulation the model has no structure. The names of the states and the variables of the submodels are replaced by names like `QQQQ1`, `QQQQ2` etc that are generated automatically resolve potential name conflicts. In the simulation language Simnon (Elmqvist, 1975) blocks are included in the language, but it is necessary to specify causality.

Bond graphs (Karnopp and Rosenberg, 1971) is another way of describing a model. It works well if the components are coupled via energy exchange only.

Our proposal for model structuring is object-oriented and introduces a small, basic, common set of concepts; A collection of basic objects like *models* and *terminals* with specified properties and operations. The proposal is described further in the next chapter.

Automatic consistency checking

It is important to make the use of library models safe and reliable. A model component is an encapsulated entity with well defined interfaces. This prevents to a large extent unintended abuse. It would be nice if the user could get automatic warnings when making improper connections when putting together a model. To allow automatic consistency checks, the model developer must "supply" redundant information. Our concepts allow a model developer to supply such information as described in next chapter. However, it is not our aim to force a user who, for example, is in an exploratory phase, to specify things that the computer itself can deduce from the context. A model developer is hopefully better motivated to supply redundant information when he has tested the model and is going to include it in a public model library. Such information can be seen as a part of the model documentation.

Customized user interfaces

We believe that various users could agree upon the objects proposed, but that they want to have customized user interfaces with various textual and graphical presentations. The proposal focuses on the basic objects and their properties and allows integration of different customized user interfaces.

The concepts proposed are basic and are mainly intended for researchers and modelling and simulation specialists. Other user categories can be supported by building new user interfaces and new layers of tools. Such tools can allow an architect or a chemical engineer to describe his building or chemical plant and the assumptions in his own language. The tools should then generate the desired model in an explicit form as outlined below. It means that the generated model is readable and can be modified by the user. Today's "high-level" tools of this kind are too rigid. They produce canned, black box models which cannot be modified. The user is in trouble if some component is missing, since it is very difficult or even impossible for him to add new components.

4. Model Structures

An important conclusion from computer science is that modules should be encapsulated with well-defined interfaces. The idea is to support abstraction by separating the internal details of a model from its interface. It means also that internal details can be changed without affecting the way the module is used as a component.

The *model* is the kernel's basic structuring unit. It is an abstraction of some dynamic behaviour. A model consists of three parts: terminals, parameters and realizations. The terminals are variables which constitute a well-defined interface to describe interaction with the environment. Parameters are interface variables defined by the model designer to allow the user to adapt the description of behaviour.

Realizations

A realization is a description of model behaviour. A model user can use a model without having to bother about how its behaviour is defined internally and the model designer can and must define its behaviour without any assumptions about the environment.

One reason for treating a realization as a separate part within the model is that we want to have multiple realizations. Different realizations can give more or less refined descriptions of the behaviour or they can define the behaviour for different working conditions or phases of a batch process. The user can choose the appropriate realization for each particular use.

We distinguish between primitive realizations and structured realizations. A structured realization is decomposed into submodels and its behaviour is described by the submodels and their interaction. The submodels can in turn have structured realizations which means that the model concept is hierarchical. A primitive realization is not decomposed into submodels, but its behaviour is described in some mathematical or logical framework as differential equations, difference equations etc.

Parameters

A parameter is a time invariant variable that can be set from outside to modify a realization. The burden of a user to set parameters can be relieved by letting the model developer provide default values. If a good default alternative is provided, the casual user could be left unaware about the flexibility and no extra burden is put on him. To support reparameterizations and alternative parameters, it is possible to define relations between parameters.

Terminals

Terminals can be viewed as variables which are shared by the internal description of the model and its environment.

It is natural to aggregate terminal variables, since the description of an interaction often involves several quantities. We propose two types of composite terminals: record and vector terminals. Their subterminals can be simple, record or vector terminals.

EXAMPLE 4.1—A pipe terminal
A terminal to describe the ends of a pipe or the inlets and outlets of pumps, valves and tanks can be defined as a record terminal

```
PipeTerminal IS A RecordTerminal WITH
  components:
    p IS A PressureTerminal;
    q IS A MassFlowTerminal;
    d IS A DiameterTerminal;
END;
```

having three components, which are simple terminals. The component d defines the diameter of the pipe or hole. □

Connections

Interactions between submodels of a structured realization are described by terminal connections. The term "connection" reflects what we are doing in the block diagram when describing an interaction. We will not discuss user interfaces here, but just point out that a block diagram is a good way of describing model structure. Elmqvist and Mattsson (1989) have developed a prototype simulator, where hierarchical block diagrams with information zooming are used to visualize the model structure. Information zooming means that the amount of information displayed in a block changes dynamically depending on its size on the screen.

A connection between two structured terminals means that their first components are connected to each other and so on recursively down to the level of simple terminals. There are two sorts of simple terminals: *across* and *through*. A connection between two across terminals means that they are equal. Examples of physical quantities are position, pressure, temperature and voltage. Through terminals have an associated direction (in or out) and connected terminals should sum to zero. Examples of through quantities are mass flow, energy flow, force, torque and current.

A simple terminal has an attribute defining the unit of measure with the SI unit as default. It is used for automatic introduction of proper scale factors in the connection equations, thus eliminating the need of user defined adapters.

It is important to note that generally the causality of a terminal (input or output) is not defined by the model designer but is inferred from the use of the model.

The semantics of a connection is kept simple, since we do not want to provide two different ways of describing complex behaviors. It is possible to describe complex interaction by introducing new submodels. It is also desirable to make the means to describe interactions independent of the frameworks used to describe the behaviour of primitive models.

EXAMPLE 4.2—Pipe terminals cont.
Assume that we want to model a system where a tank has a valve at the outlet. We then just connect the outlet terminal of the tank model to the inlet terminal of the valve model. The equations for the interaction saying that the pressures as well as the diameters should be equal and that the mass flows should sum to zero are deduced automatically from the connection. □

Consistency of connections

It is important to make the use of library models safe and reliable. The encapsulation of the models prevents to a large extent unintended abuse, but the terminals are dangerous holes in the wall. To allow automatic checks of connections, the model developer may add extra information, which also is useful for documentation.

Simple terminals have the attributes name of quantity and value range. The name of quantity is used used to check the consistency of connections. There is an international standard (ISO 31) for naming of quantities in different national languages like English or Swedish. Information about ranges of validity is used to test for unintended abuse during simulations.

A terminal component may be declared as time-invariant. Such a terminal is similar to a parameter. This has two complementary uses. First, a connection implies automatic propagation of parameter values from one submodel to another. Second, if the two connected parameter terminals have defined values, they must be equal for the connection to be consistent.

EXAMPLE 4.3—Pipe terminals cont.
Consider `PipeTerminal` in Example 4.1. The pressure component `p` can be defined by

```
PressureTerminal IS A SimpleTerminal
WITH
  attributes:
    value       := UNKNOWN;
    quantity    := pressure;
    unit        := kPa;
    direction   := across;
    variability := time_varying;
    causality   := UNKNOWN;
END;
```

The mass flow component `q` and the diameter component `d` are defined in analogous ways. An important difference is that mass flow is a through variable and the direction attribute should be set to `in` or `out`.

The variability of `d` ought to be set to `time_invariant` if the model does not allow the size of the pipe or hole to vary with time. It also allows automatic check of that two connected pipes are of the same diameter.

The terminal could also have a component indicating medium, which can be used for consistency checking or parameter propagation. For example, we can check that water pipes are connected to water pipes. □

Unspecified terminal attributes

To allow exploratory model development and prototyping, a declaration of a terminal may leave attributes unspecified as long as necessary information can be deduced from the context. Unspecified attributes make it possible to develop generic models. To support consistency checks of generic models, the model developer can specify relations between unspecified attributes. See Mattsson (1989b).

5. Object-Oriented Representation

In this chapter we will outline the conceptual design of a kernel for model representation. The basic entities, relations between entities and operations on them are discussed.

Object-oriented programming has been an increasingly popular methodology for software development. Increased programmer productivity, increased software quality and easier program maintenance are the objectives for this new methodology. Object-oriented programming supports these objectives by facilitating modularization and reuse of code. We will here show that ideas from object-oriented programming are useful also for model representation. For a brief introduction to object-oriented programming see Stefik and Bobrow (1986).

Basic model objects

Models and model components are *objects* in the kernel for model representation. An object has a unique identity within the system and it contains a collection of attributes. There is a number of important types of objects recognized in the kernel. They are representations of model structuring entities discussed in the previous section:

- models,
- terminals,
- parameters and
- realizations.

The last three object types can be used as components of models.

Class objects and relations

In our proposal, all model objects are represented as classes. In object-oriented programming a class describes the properties common to a set of similar objects – it defines an object *type*. For this reason, a model defines a component type rather than a particular instance of a component; the same applies to realizations, terminals, etc. A class can have a number of *attributes* which can be simple variables or relations to other model objects.

There are a three important relations which can be established between model objects. These are:

- has – part-of
- subclass – super class
- connection

The *has*-link is typically used between a model and its terminals, parameters and realizations. Further, a structured realization has this kind of relation to other models indicating the submodels. A has-link is stored as an attribute of the owner. The inverse relation is called *part-of*.

One class can be defined to be a *subclass* of another class – the super class. The subclass will inherit all properties of the super class in addition to the locally defined properties. *Inheritance* is an important concept in object-oriented programming and its use in this context will be discussed below.

A *connection* is a symmetric relation between two terminals and it is stored as an attribute of a structured realization.

EXAMPLE 5.1—Tank model
In this example we will show a model of a tank written in Omola (Object-Oriented Modelling Language) (Andersson, 1989a,b). Omola is a declarative language for model representation that has been designed to support our proposed concepts.

```
Tank IS A Model WITH
 terminals:
  inlet  IS A InPipeTerminal;
  outlet IS A OutPipeTerminal;
  level  IS A OutTerminal;
 parameters:
  area TYPE real := 1.0;
  roh  TYPE real := 1.0;
 realization:
  normalBehaviour IS A SetOfDAE WITH
   equations:
    area*dot(level) =
      inlet.q - outlet.q;
    inlet.p + level*roh*g =
      outlet.p - roh*v*abs(v)/2;
    outlet.q =
      pi*(outlet.d/2)^2*v*roh;
 END;
END;
```

This code represents a tank model with three terminals, two parameters and a realization component stored as attributes. The inlet and outlet terminals are both pipe terminals as in Example 1, but with directed flow components. For `inlet` positive flow is into the tank and for `outlet` positive flow is out from the tank. The realization has three equation attributes. The first equation is a mass balance and the other two are derived from Bernoulli's equation. In Figure 5.1 we can see some of the objects involved and their relations represented graphically. □

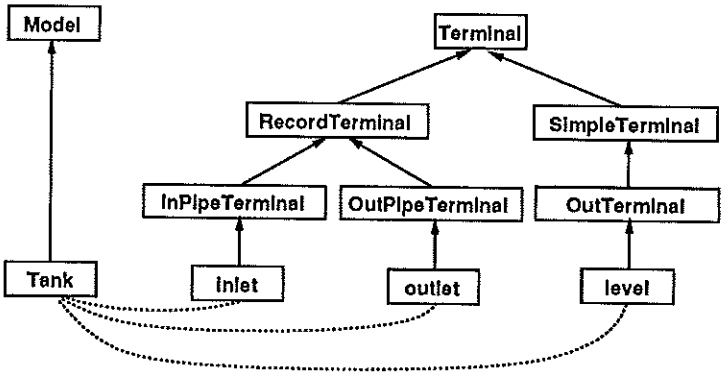

Figure 5.1 Some of the objects and their relations in the tank model. Subclass links are solid while has-links are dashed.

Inheritance

Inheritance is an intricate but powerful concept in object-oriented programming. When a class is defined to be a subclass of another class it will inherit all attributes and properties from the super class. The subclass is then free to add additional attributes or to redefine inherited attributes. Inheritance can be used to separate out some general attributes from a set of similar classes into a common super class.

Inheritance will facilitate reuse of models since carefully designed general models can be saved in libraries. These models or model components can be used as super classes of more specialized model objects. We have already seen how terminals have been defined in this way. The `inlet` and `outlet` terminals of the tank model are subclasses of `InPipeTerminal` and `OutPipeTerminal` which are specializations of the same super class `RecordTerminal`.

As an example of how models can be defined by specializations we can imagine a model of a regulator defining only the terminals: set-point, measure value and control value. This model can be specialized into different types of regulators by means of adding different realizations. We may then define a structured model like in Figure 5.2, containing the most general regulator model. The structured model can then be specialized to contain different regulator models.

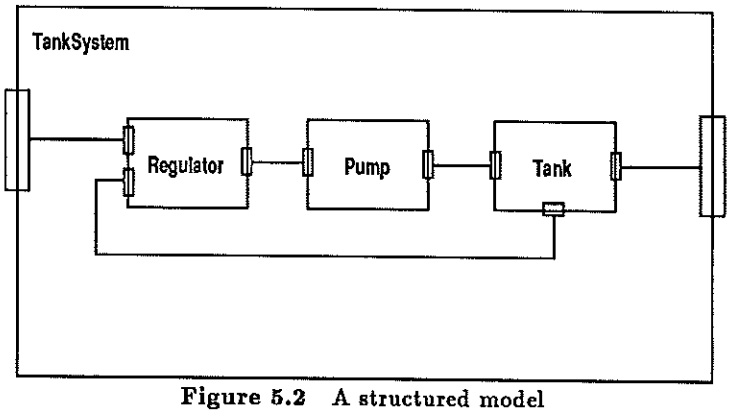

Figure 5.2 A structured model

Interpretation of model objects

Model structures represented in the kernel or in Omola code can be accessed and manipulated by different tools in a CACE environment. We may say that a particular tool that extracts relevant properties of a model *interprets* the model. Different tools may extract different properties and therefore, they interpret the model differently.

Since all model objects discussed so far are classes, i.e., they represent types rather than instances of model objects, one obvious interpretation is to use a model as a template to create a *model instance*. A model instance is, for example, needed when the model is going to be simulated. Then there must be representations for each particular model object and state variable. The instantiation procedure is recursive in the components and submodels. Typically when we want to simulate a model it is first instantiated then all equations are extracted from the primitive models and equations are generated from the connections. Second, the equations are sorted and turned into code that can be used by the DAE-solver. Since the model structure is maintained

in the simulation model (the model instance) the user can access it the normal way, perhaps through its block diagram, and examine or change parameters and initial values.

As examples of other possible interpretations of model objects we can mention

- to generate a graphical picture of a system structure,
- to generate a text descriptions of a model for documentation,
- to generate a special purpose code, e.g., regulator code or
- to turn a model into a form accepted by a particular design package.

6. The User Interface

The user interface is a very important component in any computer based tool and in particular it is important in our proposed environment. A simple user interface has been implemented in our prototype in order to demonstrate the basic concepts. In this chapter we will first give a very brief overview of the current trends in design of interactive user interfaces. Then we will give a short description of the interface of the implemented prototype.

Current trends in user interface design

Human – computer interaction is currently a very active research area. Developments in computer hardware technology have made it possible to create very advanced user interfaces to application programs. However, the methods for designing such advanced interfaces are still rather primitive.

A current trend is to more and more separate the implementation of the user interface from the application program. For simple applications this can be done in a clean and natural way, but in many cases for more complicated application programs this is not a clean cut. Often a good user interface needs a substantial amount of "understanding" of the application. In other words, the user interface needs a model of the application. In this case we have the problem of keeping the application model consistent with changes in the application.

Another trend in interface design is to use higher level specifications of the user interaction. Commonly used are toolkits of various graphical objects and interactors, such as dialogue boxes, push buttons and menus of different kinds. They are often designed for a special computer or a special window manager. The InterViews (Linton et al., 1989) is an example of such a toolkit based on X Window System. A *User Interface Management System* (UIMS) contains a library of interactive objects like the toolkits but it also has a number of tools that helps the interface designer to put the objects together into a complete user interface. The interface designer may describe the interface on a more abstract level, sometimes by a declarative language. This means that the designer specifies what is to be done by the user interface rather than the exact details of how to do it. The designer may also use some formalism to describe dialogue such as transition graphs or BNF (push-down automaton). Some more advanced tools allow the interface designer to build the user interface in

an interactive or semi interactive way with immediate feedback showing the current appearance of the interface. An introduction and survey of UIMS can be found in Mayers (1989).

The ongoing standardization of windowing systems and graphic input and output primitives makes it more attractive to develop and commercialize advanced UIMS software. In a few year's time such systems will probably be more commonly available at reasonable prices.

The prototype interface

In our prototype we have realized the importance of a reasonable good user interface. In the project we have not in particular studied user interface design as such, since the project have been focusing on representation of models rather than the presentation. However, in order to demonstrate the power and appropriateness of the underlaying representation a reasonably advanced interactive user interface had to be designed. We chose KEE[1] as the basic implementation tool for the prototype. One reason was that it provided some amount of support for building user interfaces. KEE uses object-oriented representation of graphical entities. Predefined primitive graphical objects can be specialized and combined into more advanced ones.

A graphical interface in our suggested modelling environment can not be clearly separated from the application — the model representation data base — because it is too much involved in the used data model. The approach taken instead, is to let the user interface operate directly on the model data base. The models represented in the data base may then contain additional information manipulated only by the user interface. For example, a model contains information about how it is presented on the screen, graphically or as text, menus of possible operations, etc.

Direct manipulation of models

The style of interaction in the prototype user interface is based on *direct manipulation*. Most objects, attributes and relations in the model data base can be represented on the screen. The screen representation can be a graphical icon, a diagram or a textual representation. In general every object is mouse sensitive and has an associated menu of operations.

The model data base in our prototype is divided into a set of *libraries*. A library is a collection of model objects and their attributes, and it can be saved and loaded from external memory. Objects in different libraries may have relations. The screen is separated into four important areas:

- an access window for loaded libraries,
- a library display window,
- an editor area and
- one or more general display windows.

The access window for loaded libraries displays a list of all loaded libraries where each entry is mouse sensitive and has an associated menu of library actions. The library display window shows the content of a selected library. For example, it may show the graphical icon of every model object in the library. Two important operations are implemented for most model objects; these are *display* and *edit*. These operations can of course be called from

[1] Knowledge Engineering Environment, KEE is a trademark of IntelliCorp, Inc.

the object's menu which is accessible through its icon but since they are very commonly used there is an alternative short cut. An object can be picked from the library display and an icon contour image can be dragged into an appropriate area of the screen. If an object is dragged into a display window the object will be displayed in that window. If an object is dragged into the editor area of the screen, the object can be edited.

In the editor area of the screen, one of a number of different editors may appear. The type of the object to edit determines which editor that will be invoked. There is a text editor for primitive realizations (equations) and other text definitions. A structure editor is invoked for block diagram editing of structured models and a form is invoked for editing the attributes of simple terminals.

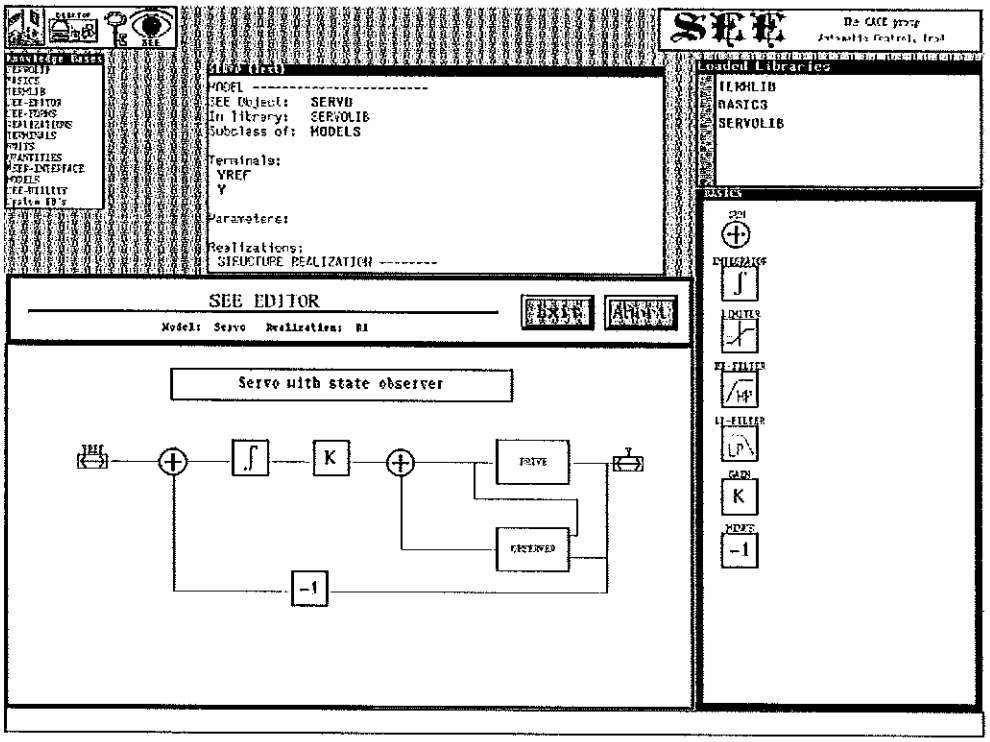

Figure 6.1 The prototype modelling environment.

7. An Application Study

The project has also included an application study which focused on modelling of chemical processes. The aim was to get some evaluation of the ideas and feedback from a real example. The application study has been performed by Bernt Nilsson, who is a chemical engineer interested in modelling. He has played the role of a user who wants to model a medium sized, typical chemical process plant that contains a reaction part with a tank reactor and two tubular reactors, and a separation part with three distillation columns in series.

The modelling work is described and discussed in Nilsson (1989), where also the model can be found. Since the application is a typical chemical plant, he presents an object-oriented modelling approach for chemical plants.

Chemical processes are often complex plants that are composed of a large number of components. However, chemical processes are often built as a number of subprocesses. In the application there are the reaction part and separation part. These subprocesses can be decomposed further into process components or unit operations. This decomposition is easily and neatly described by our hierarchical model decomposition concept. The process components are often standard process equipments such as pipes, pumps, valves, reactors, heat exchangers, distillation columns etc. that are used in different configurations in different processes. A model class allows reuse of a description in several instances and the inheritance mechanism allows adaptation of a model.

Nilsson (1989) describes ways of further decomposing chemical models. One interesting example is the medium and machine decomposition. It is of interest to separate the description of the process components from the descriptions of the chemical media. In today's simulation systems a model of for example a chemical reactor contains a reaction model which can only be modified by setting parameter values. A specification of a chemical reactor should contain the equations describing its thermodynamic and hydrodynamic properties, while the equations describing the reaction should be associated with the chemical media. Nilsson shows that the submodel concept allows a nice medium and machine decomposition.

Regular structures are common in chemical processes. For example, a distillation column may contain a few hundred trays connected in series. To handle this conveniently, Nilsson proposes matrices of submodels and a matrix notation to describe how they are connected. Finite element approaches to distributed parameter systems (partial differential equations) create also regular structures.

Parameterization and generic models are important to increase the flexibility and the reusability of models. The concepts proposed support Nilsson's basic needs of parameterizations. He states that it important to be able to parameterize structural properties like the number of chemical components flowing in a pipe. With matrices of submodels it is possible to let the number of trays in a distillation column be a parameter. He illustrates in several ways that inheritance allows powerful parameterizations. For example, it makes it simple to change the model of the medium in the distillation column.

Nilsson concludes that the proposed model representation is superior to existing ones, but it requires also a good model/user interface and number of tools to make a good modelling environment.

8. Conclusions

First, the results of the project is summed up. Then technology transfer is considered. Third, some more general experiences of software techniques and tools are discussed.

8.1 Results

Most of today's languages for continuous simulation follow the CSSL definition (Strauss, 1967). It has served well for over 20 years. We think it is time to capitalize on the enormous development of information technology and reconsider the foundations of model representation. Our proposal is a modest effort in that direction.

The major contribution of the project is that experience in model structuring, progress in numerical analysis and new ideas in object-oriented design are collected and turned into a coherent scheme for model representation. Our proposed model representation scheme is general, powerful, clean and easy to understand. The result is presented as a design proposal of a kernel for model representation. The kernel is intended as central model representation data base in an environment of tools for system engineering. The basic features of the kernel are:

- Declarative and equation based behaviour descriptions to make the models versatile and useful for various applications.
- Hierarchical models with well defined interfaces based on terminals and parameters.
- Terminal attributes for automatic check of connection consistency.
- A model may have several behaviour descriptions to support model versions and alternative behaviour.
- Object-oriented representation where classes with inheritance facilitates reuse and incremental model development.
- An internal representation which preserves the structure of models.
- The kernel allows integration of customized user interfaces and various tools.

A prototype implementation has been written in Common Lisp and KEE. To get some feedback and evaluation of the ideas, the project has also comprised an application study focusing on modelling of chemical processes. Experiences from the prototype and from the application study indicate that the ideas are sound and that the kernel proposal may indeed serve as a basis for a new generation of modelling, design and simulation tools.

The proposed kernel ought to be of interest in all areas of engineering and for all who use models and simulation. The ideas have been presented at conferences and the prototype has been demonstrated for a number visitors from industry and universities. People from many different areas of engineering who have struggled with similar problems of model representation, have found our solutions very interesting.

8.2 Development and technology transfer

The scientific results of the project are and will be made public through articles in international magazines and as conference contributions (see Appendix A). Some of the results have and will be published as licentiate and doctoral theses.

The contact net with Swedish and foreign universities and companies that develop CACE tools is extensive and functioning. We are now extending it to include also researchers and developers working with model development tools and simulation in general. The international conferences give good opportunities to exchange ideas and information and to make acquaintance with new people. Besides control engineering conferences we have also participated in conferences aimed at modelling and simulation in general as well as conferences aimed at special applications as chemical engineering and building simulation.

Implementation of the kernel

A very good way of transfering results like that of our project is of course to make the tools available to many people. The experiences from Simnon show that useful program components are a very good way of spreading new ideas and methods. Our prototype is written in Common Lisp and KEE. The advantage for us of using KEE was that the prototype could be implemented with a modest effort. However, since KEE is very expensive, we do not expect the prototype to be widespread.

To make our tools generally available, it is necessary to implement them using cheaper and more commonly available languages and software components. We think that it is not the task of a university to develop, market and maintain commercial and professional software. But we realize that we have a responsibility of transfer the results of our project and making them generally available. A project supported by STU (STU project 89-01837 "A kernel for modelling and simulation") has just been started to implement a kernel for model development and simulation, which someone else can develop further into a commercial product of the prototype kernel. C++ is used as the basic implementation language. An economic reality is that it is expensive to develop professional software and the market for CACE-products is relatively small. However, our kernel may be of interest in most areas of engineering and ought to have a much larger market. There are companies and groups that have expressed interest in making a commercial product. However, it is too early to make any predictions now and we welcome all proposals.

Application projects

Another good way of spreading new ideas and methods is to have joint application projects with developers and users. However, to be able to transfer the results in application projects, implementation of the tools are needed. In application projects the developers get feedback and can modify and improve the tools. For a special application they can develop customized tools and user interfaces. Model libraries can be built which can be of use not only for the participating part, but also for a whole line of business.

We are planning to run a number of application projects. However, as pointed out above, we need implemented tools and the implementation project has therefore been given priority. A proposal for an application project together with Sockerbolaget has been submitted to STU's research program

DUP. The application is modelling of sugar crystallization. Simulation and simulators have a central role in DUP, which aim is to investigate how process operators' tasks can be supported by computer based tools. It is natural to set up and finance application projects in DUP. STU's program "Applications of the information technology" is another possible source of financing application projects.

Standardization

Much could be gained if we could agree upon a common set of ideas. It is time to lay the foundation for a new standard for model representation. IFAC has a working group on standards for CACSD Software. We are participating in this work. It has not addressed non-linear systems yet, but it has focused on linear systems.

It may be remarked that to build flexible model libraries we must also agree on common principles for model development. This is a hard task, but it might be possible to achieve in certain application areas.

There is an international association IBPSA (the International Building Performance Association), which promotes the science of building performance simulation in order to improve the design, construction, operation and maintenance of all types of buildings. IBPSA's international membership includes architects, engineers, building managers, academics, software developers, and government representatives concerned with building performance. IBPSA organized a conference Building Simulation '89 on June 23–24, 1989 in Vancouver, Canada. At this conference Per Sahlin, The Swedish Institute of Applied Mathematics, ITM, Stockholm and Edward Sowell, California State University, Fullerton, California presented a proposal for a neutral format for building simulation models to allow users to share models (Sahlin and Sowell, 1989). This proposal is inspired and influenced by the results of our project.

8.3 Experiences of software techniques and tools

The project has dealt with design of tools for model development and simulation and to do this we have exploited ideas, approaches, methods and techniques from computer science as well as used existing software. Hence we may also ask what we have learned that can be of more general interest.

Object oriented programming

Object oriented programming (for overview see e.g. Stefik and Bobrow, 1986) is a technique for structuring programs and to support reuse. The basic ideas are data abstraction and inheritance.

Objects and abstraction is natural in engineering. Block diagrams and and other kinds of schematics and flow graphs are common. Blocks have often well defined interfaces. In control engineering it is common to talk about input/output models, where only the relations between the inputs and outputs are known. Nothing is then said about the internal structure or the implementation of it. When the model also defines internal structure, we speak about internal models.

Although the ideas of object oriented programming are or at least seem to be natural, it is not self evident how to use them in a special application. Zobel and Cummings (1989) discuss use of object orientation for digital signal

applications. They had found that it is many cases not obvious whether operations should be implemented as methods or as special processing objects. For example, should FFT be a method of signals or a special object (machine)? They were going to carry through both approaches to get a deeper insight and experiences.

In discrete event simulation the models can perform the simulation themselves by sending messages to each other. This is not possible in continuous time. The differential-algebraic equations must be solved simultaneously. It could be done in an object oriented fashion by having a Solver object that collects the equations from all the models, solves them and returns the result to the models.

We have used object orientation on several levels. First, for the architecture of the system to get an flexible and extendible integrated environment which allows customized user interfaces. Second, the modelling concepts are object oriented. Third, the internal model representation is also object oriented.

A kernel for model development must allow interactive definition and creation of new model classes (types). Interactive languages like KEE, CLOS, Smalltalk allow interactive definition of new classes and a model class can basically be implemented as a class in the implementation language. In compiled languages like Simula and C++, it is not possible to define new classes interactively. It means that it is not possible to represent model classes directly as classes in the implementation language. An extra layer to handle definition of new model classes and inheritance between model classes must be implemented.

Databases

Databases are central. We need them to store models, parameter data, measurements, results of calculations etc. Common representations are needed to make the tools integrated.

Today's databases can handle a large set of independent data efficiently, but in CACE the amount of data are moderate, but the relations are complex. For example, a model may be a linear version of another model at a certain operating point for some given parameter values. Object oriented databases is a promising approach.

Graphics and user interfaces

Computer graphics gives good possibilities to improve the user interface. It can be used to make concepts, properties, structures and other information more concrete. Direct manipulation is an interesting technique which allows the user to operate on visual objects and get immediate visual feedback. Visual metaphors must be selected carefully to give the user a correct conception.

Graphics must be designed carefully to be useful and endurable in the daily use. The primary use of graphics should not be spectacular demonstrations.

Unfortunately, it is laborious to implement graphics. First, portability is a major concern. As a user you want to have a homogeneous environment. The advantages of having a standard window system for all CACE programs are quite obvious and uncontroversial, but it should be noted that CACE programs are not the only use of a workstation. The user will use the native, vendor-supplied window system, and would therefore prefer that one also in

CACE programs. The same also applies for text editors. The situation seems to improve and X Window System is today a de facto standard. Second, there are today very few tools available for definition and implementation of user interfaces, but it is an active area. Hopefully there will be commercial user interface management systems (UIMS) available within a few years.

AI and expert system techniques

The complexity of AI has speeded up and influenced the development of powerful workstations, high interactivity, computer graphics, animation, object orientation, direct manipulation etc.

We consider the expert system technique as a useful and powerful programming technique. The kernel does not itself contain any expert system, but we have exploited ideas on information representation and declarative programming; equations to describe behaviour. Rules can be used to define events. Deduction of unspecified model attributes and consistency checking can be implemented by rule based systems.

Some people claim that they have knowledge based simulation when they provide simple model libraries. Knowledge based simulation is, however, in our opinion more than providing a model library. There should be facilities that assist the user to select the proper models and model versions as well as to evaluate the results.

Symbolic manipulation and computer algebra

The increasing computing capacity makes it possible to perform symbolic manipulation. The user can give his problem on for him a suitable form. Symbolic manipulation can then be used to simplify the problem and to generate descriptions that the numerical tools need. Analytic expressions may give better insight than tables of numerical values.

Existing commercial packages for computer algebra such as MACSYMA, REDUCE, Scratchpad, Maple and Mathematica are powerful. Unfortunately, it is not easy to use exisiting packages for computer algebra in other tools. They are interactive and assume that they are run by human beings. The results returned from the packages are on a format intended for human beings. They are not built to be run or called by other programs. They can of course be run as separate processes and comunication can be done via pipes, mailboxes etc. depending on the operating system. The difficulty is to decode the text strings returned by the package. The reference manuals do not give any formal specification of the format. So if we want to write a program to decode it, we have first to investigate what is returned. All this is laborious to do but the situation is even worse. Since the format is not formally specified, a new release of the package may change (improve) the format.

Libraries of routines for symbolic manipulation, like the numerics libraries would be very useful.

Acknowledgements

The authors would like to thank Professor Karl Johan Åström, Bernt Nilsson and Dag Brück for many useful discussions and Tomas Schönthal for his work on the prototype. This work has been supported by the National Swedish Board for Technical Development (STU) under contracts 87-2503 and 87-2425.

References

ANDERSSON, M. (1989a): "An Object-Oriented Modelling Environment," *Proceedings of the 1989 European Simulation Multiconference*, Rome, June 7–9, 1989, pp. 77–82.

ANDERSSON, M. (1989b): "Omola – An Object-Oriented Modelling Language," Report TFRT-7417, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ANON (1987): "Challenges to Control: A Collective View," *IEEE Transactions on Automatic Control*, **AC-32**, No. 4, April 1987, 275–285.

ELMQVIST, H. (1975): "Simnon – User's Manual TFRT-3091, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden,".

ELMQVIST, H. and S.E. MATTSSON (1989): "Simulator for Dynamical Systems Using Graphics and Equations for Modeling," *IEEE Control Systems Magazine*, **9**, 1, 53–58.

FLEMING, W.H. (Ed.) (1988): *Future Directions in Control Theory – A Mathematical Perspective*, SIAM Reports on Issues in the Mathematical Sciences, Society for Industrial and Applied Mathematics, SIAM, Philadelphia.

KARNOPP, D. and ROSENBERG, R. (1971): *System Dynamics — A unified approach*, Wiley, New York.

KHEIR, N.A. (Ed.) (1988): *Systems Modeling and Computer Simulation*, Marcel Dekker, Inc., New York.

KREUTZER, W. (1986): *System Simulation – Programming styles and languages*, International Computer Science Series, Addison-Wesley.

LINTON, M.A., J.M. VLISSIDES and P.R. CALDER (1989): "Composing User Interfaces with InterViews," *IEEE Computer*, **22**, 2, February 1989.

MATTSSON, S.E. (Ed.) (1987): "Programplan för ramprogrammet Datorbaserade hjälpmedel för utveckling av styrsystem (Computer Aided Control Engineering, CACE)," Final Report TFRT-3193, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. (1989a): "On Modelling and Differential/Algebraic Systems," *Simulation*, **52**, No. 1, 24–32.

MATTSSON, S.E. (1989b): "Modelling of Interactions between Submodels," *Proceedings of the 1989 European Simulation Multiconference*, Rome, June 7–9, 1989, pp. 63–68.

MAYERS, B. (1989): "User-Interface Tools: Introduction and Survey," *IEEE Software*, **6**, 1, January 1989, 15–23.

NAGEL, L.W. (1975): "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Technical Report ERL-M520, Electronics Research Lab, University of California Berkeley.

NILSSON, B. (1989): "Structured Chemical Process Modelling — An object oriented approach," Lic Tech thesis TFRT-3203, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

SAHLIN, P. and E.F. SOWELL (1989): "A Neutral Format for Building Simulation Models," *Proceedings of Building Simulation '89*, Vancouver, June 23–24, 1989, The International Building Performance Simulation Association, pp. 175–180.

STEFIK, M. and D.G. BOBROW (1986): "Object-Oriented Programming: Themes and variations," *AI Magazine*, **6:4**, 40–62.

STRAUSS, J.C. (Ed.) (1967): "The SCi Continuous System Simulation Language (CSSL)," *Simulation*, Dec 1967, 281–303.

ZOBEL, R.N. and A.M. CUMMINGS (1989): "Developments in an Object Oriented Environment for DSP Applications," *Proceedings of the 1989 European Simulation Multiconference*, Rome, June 7–9, 1989, pp. 263–268.

A. Published Papers and Conference Contributions

ANDERSSON, M. (1989): "An Object-Oriented Modelling Environment," in G. Iazeolla, A. Lehman, H.J. van den Herik (Eds.): *Simulation Methodologies, Languages and Architectures and AI and Graphics for Simulation*, 1989 European Simulation Multiconference, Rome, June 7–9, 1989, The Society for Computer Simulation International, pp. 77–82.

ANDERSSON, M. (1989): "An Object-Oriented Language for Model Representation," IEEE CACSD'89, Tampa, Florida, December 16, 1989.

BRÜCK, D.M. (1988): "Modelling of Control Systems with C++ and PHIGS," *Proceedings of the USENIX C++ Technical Conference*, Denver, Colorado, October 17–20, 1988, pp. 183–192, Also available as report TFRT-7400, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

BRÜCK, DAG M. (1989): "Experiences of Object-Oriented Development in C++ and InterViews," *Proc. TOOLS'89*, Paris, France, November 13–15, 1989, Also available as report TFRT-7418, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. and S.E. MATTSSON (1989): "Simulator for Dynamical Systems Using Graphics and Equations for Modelling," *IEEE Control Systems Magazine*, **9**, 1, 53–58.

MATTSSON, S.E. (1988): "On Model Structuring Concepts," *Preprints of the 4th IFAC Symposium on Computer-Aided Design in Control Systems (CADCS)*, August 23–25 1988, P.R. China, pp. 269–274.

MATTSSON, S.E. (1989): "On Modelling and Differential/Algebraic Systems," *Simulation*, **52**, No. 1, 24–32.

MATTSSON, S.E. (1989): "Modelling of Interactions between Submodels," in G. Iazeolla, A. Lehman, H.J. van den Herik (Eds.): *Simulation Methodologies, Languages and Architectures and AI and Graphics for Simulation*, 1989 European Simulation Multiconference, Rome, June 7–9, 1989, The Society for Computer Simulation International, pp. 63–68.

MATTSSON, S.E. (1989): "Concepts Supporting Reuse of Models," *Proceedings of Building Simulation '89*, Vancouver, June 23–24, 1989, The International Building Performance Simulation Association, pp. 175–180.

NILSSON, B. (1989): "Structured Modelling of Chemical Processes with Control Systems," Annual AIChE Meeting 1989, San Francisco, Nov 5–10, 1989.

NILSSON, B., S.E. MATTSSON and M. ANDERSSON (1989): "Tools for Model Development and Simulation," in Larsson, J.E (Ed.): *Proceedings of the SAIS '89 Workshop*, AILU—The AI Group in Lund, Lund University and Lund Institute of Technology.

B. Reports

Lic Tech Thesis

NILSSON, B. (1989): "Structured Chemical Process Modelling — An object oriented approach," Lic Tech thesis TFRT-3203, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Master Theses

GRANBOM, E. and T. OLSSON (1987): "VISIDYN – Ett program för interaktiv analys av reglersystem," Master thesis TFRT-5375, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

JOHANSSON, M. (1988): "Interaktiv plottning av mätdata i flera dimensioner," (Interactive plotting of measurement data in several dimensions), Master thesis TFRT-5390, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

JEPPSSON, U. (1988): "An Evaluation of a PHIGS Implementation for Full Graphics Control Systems," Master thesis TFRT-5389, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

NILSSON, A. (1988): "Object-oriented Graphics for the Future Instrument Panel," Master thesis TFRT-5382, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

VALLINDER, P.A. (1988): "Some Methods for Tearing of Differential/Algebraic Systems," Master thesis TFRT-5384, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Other reports

ANDERSSON, M. (1989): "Omola – An Object-Oriented Modelling Language," Report TFRT-7417, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

BRÜCK, D.M. (1989): "Scones — An Interactive Block Diagram Editor for Simnon," Report TFRT-7423, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. and M. ANDERSSON (1989): "A Kernel for System Representation," Report TFRT-7429, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. and K.J. ÅSTRÖM (1988): "The CACE Project— Steering Committee Meeting, 1987-11-25," Report TFRT-7375, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. (1988): "The CACE Project— Steering Committee Meeting, 1988-06-01," Report TFRT-7395, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. (1989): "The CACE Project— Steering Committee Meeting, 1988-11-23," Report TFRT-7412, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. (1989): "The CACE Project— Steering Committee Meeting, 1989-09-08," To appear, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

NILSSON, B. (1987): "Experiences of Describing a Distillation Column in Some Modelling Languages," Report TFRT-7362, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

C. Lectures

Oct 2, 1987. Sven Erik Mattsson: "Hibliz," Studsvik, Nynäshamn, Sweden.

Feb 29, 1988. Sven Erik Mattsson: "Overview of the CACE project and Hibliz," Foxboro Company, Foxboro, Massachusetts, USA.

March 16, 1988. Sven Erik Mattsson: "Hibliz and the expert system interface for Idpac," Department of Information Processing, Umeå University, Umeå, Sweden.

March 16, 1988. Sven Erik Mattsson: "The CACE project," the Department of Information Processing, Umeå University, Umeå, Sweden

April 26, 1988. Sven Erik Mattsson: "The CACE project," Intelligent Automation Laboratory, Department of Electrical and Electronics Engineering, Heriot-Watt University, Edinburgh, Scotland.

May 20, 1988. Sven Erik Mattsson and Bernt Nilsson: "The CACE project and tools for model development and simulation," the management group of the DUP project at STU, Stockholm, Sweden.

Aug 23, 1988. Sven Erik Mattsson: "On Model Structuring Concepts," the 4th IFAC Symposium on Computer-Aided Design in Control Systems, CADCS'88, August 23 – 25, 1988, Beijing, P.R. China.

Sept 22, 1988. Sven Erik Mattsson: "Methods and languages for development of simulation models," workshop on the use of simulators in the process industry, arranged by the STU-program DUP, Stockholm, Sweden

Oct 20, 1988. Dag Brück: "Modelling of Control Systems with C++ and PHIGS," the USENIX C++ Technical Conference, October 17 – 20, 1988, Denver, Colorado, USA.

Nov 2, 1988. Karl Johan Åström: "Drum water modelling," a one day seminar on Simulation and Advanced Control of Power Plants at Sydkraft, Malmö, Sweden

Nov 2, 1988. Sven Erik Mattsson: "Future Modelling and Simulation Environment," a one day seminar on Simulation and Advanced Control of Power Plants at Sydkraft, Malmö, Sweden.

Nov 11, 1988. Dag Brück: "Object oriented design in C++," Ericsson Radar Systems, Mölndal, Sweden.

Dec 14, 1989. Sven Erik Mattsson: "Future Modelling and Simulation Environment," Workshop on Future Research Needs in CACSD, Cambridge, 14 & 15 December 1988. Arranged by the Science and Engineering Research Council, SERC in UK.

March 14, 1989. Dag Brück "Experiences of C++ and UNIX," STFI, Stockholm, Sweden

May 7, 1989. Bernt Nilsson: "Tools for Model Development and Simulation," SAIS '89, the annual workshop of the Swedish Artificial Intelligence Society, Lund, Sweden

May 31, 1989. Sven Erik Mattsson: "The CACE project and new tools for model development and simulation," the council of SIGSIM, Sweden.

June 8, 1989. Mats Andersson: "An object-oriented modelling environment," ESM'89, the 1989 European Simulation Multiconference, June 7–9, 1989, Rome, Italy.

June 8, 1989. Sven Erik Mattsson: "Modelling of interaction between submodels," ESM'89, the 1989 European Simulation Multiconference, June 7–9, 1989, Rome, Italy.

June 23, 1989. Sven Erik Mattsson: "Concepts supporting reuse of models," Building Simulation '89, Vancouver, Canada.