*Proceedings of the 1993 Winter Simulation Conference*
*G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)*

# EMPLOYING DATABASES FOR LARGE SCALE REUSE OF SIMULATION MODELS

Martin Hitz

Data Engineering Department
University of Vienna
Rathausstr. 19/4, A-1010 Vienna
AUSTRIA

Hannes Werthner

Department of Statistics, OR, and Computing
University of Vienna
Universitätsstr. 5, A-1010 Vienna
AUSTRIA

Tuncer I. Ören

Computer Science Department
University of Ottawa
Ottawa, Ontario
CANADA, K1N 6N5

## ABSTRACT

To enhance reusability in the field of simulation, model bases must be equipped with powerful tools for retrieval, modification and aggregation of simulation models. In this paper, the role of database management systems supporting a modeling environment is discussed and reuse oriented query interfaces are presented. Specifically, we advocate a graph browser operating on a semantic network of reusable components.

## 1 INTRODUCTION

Reusability of software artifacts has become a well established requirement in the software engineering community during the last decade, and its importance in the field of simulation has also been pointed out by many authors already (e.g., Reese and Wyatt 1987, Ulgen, Thomasma, and Otto 1991). However, simulation models still seem to be lacking reusability (Lung et al. 1992): Many modelers still view each simulation project as unique and often construct from scratch a specific model to meet the objectives. As a result, cost overruns and poor performance are usually associated with the simulation building process.

Consequently, modeling and simulation environments must be complemented by specific features to enhance reusability. Such a reusability-oriented modeling environment has to support at least four crucial aspects of reusability in the simulation field:

1. Storage and documentation of models,
2. Construction of new models and modification of stored models,
3. Retrieval of models which satisfy the modeler's needs,
4. Composition of new aggregate models from existing parts, paying heed to semantic constraints as far as possible.

In this situation, the role of database management systems (DBMSs) in simulation environments can be envisaged as twofold:

Firstly (and classically), DBMSs can be used as data stores for all kinds of data relevant to the study at hand, e.g. input or output time series and the like. In this context, DBMSs can be used as a standard interfacing mechanism between distinct simulation models, mediating the information flow in aggregate (compound) models or in experimentation / optimization submodules of integrated simulation environments (Ören et al. 1992; Guariso, Hitz, and Werthner 1993).

Secondly, on a meta level, DBMSs may serve as tools supporting model bases, i.e. repositories for model descriptions. Instead of storing specific model code (i.e., input to simulation packages), the structural and semantic aspects of simulation models can be managed, together with (procedural) knowledge of how to construct the input of a specific package prior to model execution. In this latter role, DBMSs significantly enhance the reusability of simulation models.

Naturally, it is most convenient to use a DBMS as a persistent data store. It relieves the designer of a simulation environment of many technical burdens, like multi-user access, transaction control, backup and recovery etc. Moreover, access is provided by standardized mechanisms, thus guaranteeing physical data independence, an important asset for the software engineering process. However, while it is fairly obvious that data can be suitably managed within a data base management system, the organization of model definition, storage and retrieval is more complex.

The authors' research activities of the last four years in the area of simulation and decision support systems have lead to the development of several working prototypes in different software and hardware environments, all of which are focusing on (among other aspects) certain problems of reusability in these areas (Guariso, Hitz, and Werthner 1988, 1989; Guariso et al. 1990; Hitz and Werthner 1992). Their general architectural framework is described by Guariso, Hitz, and Werthner (1993) and depicted in Figure 1. SIMOBA, the latest of these prototypes (Nemec 1993), emphasizes the role of a DBMS

supporting the model base (the shaded component in Figure 1) and appropriate query mechanisms for retrieval of simulation models and represents the main corner stone of this paper. Although we are currently using this prototype to study reusability issues within the rather constrained domain of simulation with its well defined formalisms, we are also aiming to apply our findings to the problem of software reusability in general.

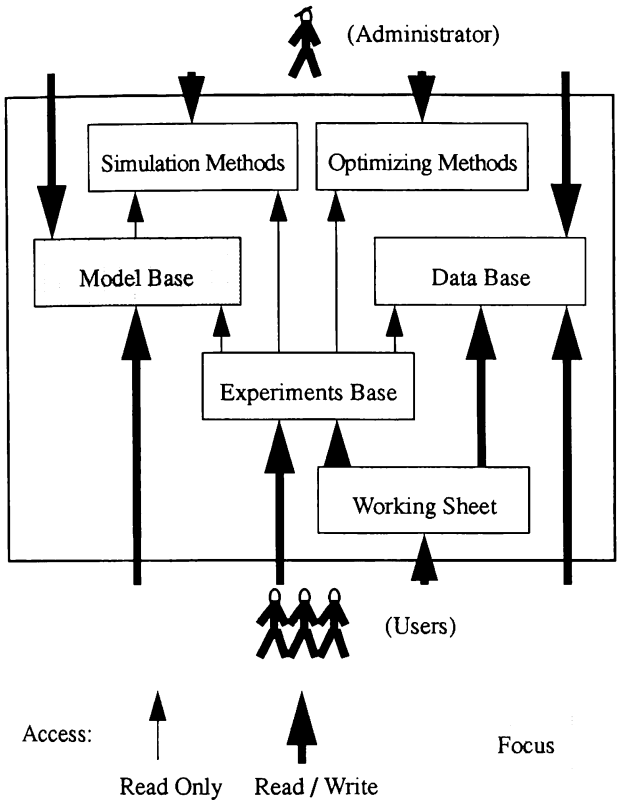


Figure 1: Proposed Environment Architecture

In what remains, we will to some extent elaborate the four main aspects of reusability as stated above. We begin with the definition of the domain under study, characterizing the data we are dealing with. We then shortly describe how we deal with storage, modification and aggregation of models, thus covering points 1, 2, and 4 of the list above. The retrieval aspect is covered by a description of the query and browsing interface of SIMOBA in Section 4, followed by some remarks on implementation issues. Finally, we explain the directions of our future work in this area.

## 2 THE MODEL BASE

The structural variety of possible models considered in the following is restricted to the classical definition by Zeigler (1976) based on input, state and output variables related by a set of ordinary differential or difference equations. They can be partitioned into three main categories: basic, static and compound models. Basic models are simply standard dynamic models, while static (memoryless) models are represented by non dynamic relationships. Compound models are aggregations of models of any type obtained by connecting the input and output variables of the components.

### 2.1 Entities

The model base serves as a repository for both, models and model classes. As usual, a class is defined as a collection of instances with similar structure. That is, for a model class, all structural attributes (like number and dimensions of variables, the state transition function, etc.) are defined, while all value-oriented properties (values of parameters, initial conditions, etc.) are added only upon instantiation. For model classes, derivation with property inheritance is defined. A subclass inherits all structural information from its superclass, but may add new features and redefine output and state transition functions. (In the following, whenever it does not make any difference, we will not explicitly distinguish between classes and instance, i.e., we will sloppyly refer to an arbitrary entry of the model base by the term "model").

Besides quantitative simulation models, we also consider qualitative simulation models (for a detailed description see Guariso, Rizzoli, and Werthner (1992)). Such models can be seen as a qualitative abstraction with respect to structure, functions and value ranges of variables. One such model may be the qualitative abstraction of more than one numerical model.

### 2.2 Attributes

A model stored in the model base must be defined by the following set of attributes:

- A name, which should be as explanatory as possible of its main features
- An extensive textual description of the relevant characteristics of the model (both these items support to different extents the user selection within a specified application domain)
- An internal type indicator ({basic, static, compound} x {discrete, continuous, hybrid}). This is relevant to the user only in the model building stage.
- A (possibly empty) set of input variables (each described by name, textual information, unit of measurement and a logical proposition defining the set of permissible values (called "value range" in what follows))
- A set of output variables (name, textual description, unit of measurement, value range and the function which computes its value)

- A (possibly empty) set of parameters (each described by name, textual information, and default value)
- A time unit
- A time step for discretization of differential equations (this is, in a certain sense, a characteristic of the model since its correct value depends on both the parameters and the equations of the model).

Basic and compound models must contain in addition:

- A set of state variables (each described by name, textual information, equation of the transition function, value range and default initial value)
- A default simulation method, i.e. a numerical procedure for solving recursive algebraic equations in case of discrete time systems or differential equations in case of continuous time systems
- Information on its internal structure, in the case of compound models composed of several interlinked submodels

While basic and compound models look similar within this framework, they differ in so far as the above attributes are given explicitly for basic models, whereas for compound models they are deduced from attributes of their components at the time of creation.

In addition to the above described essential features of models, we add several descriptors as suggested by Guariso and Werthner (1988) in the form of a Thesaurus-guided keywording scheme, such as purpose and objective of the model, the domain of its application, etc. Together with the structural attributes defined above, these descriptors form a kind of a faceted classification scheme as suggested by Prieto-Diaz and Freeman (1987) and Prieto-Diaz (1989).

### 2.3 Relationships

Apart from the "single model attributes" defined in the previous subsection, we also store semantic relationships between models in the data base that are exploited by the retrieval process as explained in Section 3.

It is important to reduce the amount of data entry necessary by the user when new models are stored, because if the model base reduces to a trivial, "flat" collection of unrelated entries, reuse is not likely to take place. Thus, the environment supports automatic insertion of some structure-based relationships, like instance-of, part-of, etc. We also allow so-called "virtual" relationships that can be derived from explicitly stored relationships via certain derivation rules.

The following relationship types are considered in the current prototype (this set is not well defined in so far as new relationship types may also be defined dynamically as explained in Section 5):

The *instance-of* relationship links models to model classes. These links may well be used for defining virtual relationships. For example, a standard derivation rule says that if two classes A and B are in relationship X, so are their respective instances a and b. *Instance-of* is also used for maintenance purposes: When a model class is changed, the changes are automatically propagated to their instances, unless the instance-of links are flagged to inhibit this propagation.

*Part-of*: This represents the classical aggregation relationship. In the modeling domain, it is used to form compound models of arbitrary deep nesting levels by connecting input/output ports of submodels. The *part-of* relationship may be expressed on the class level as well as on the instance level.

*Connected-to*: While *part-of* relates objects of different aggregation levels, *connected-to* describes the linkage information between two component models (on the same level of aggregation). *Connected-to* is defined only for model instances, but also establishes a corresponding relationship on the class level (*friend-of*).

*Is-a*: This is the classical generalization/specialization relationship. It holds if the participating model classes represent two descriptions for one and the same physical situation, one model being more detailed than the other, i.e. more variables and parameters. *Is-a* features property inheritance, which may be exploited by both, the query mechanism and the browser, which may again infer "virtual" arcs in a similar way as with the *instance-of* relationship.

*Friend-of*: This is a virtual relationship, it holds between two classes A and B, iff there are at least two instances a of A and b of B for which relation *connected-to* holds.

*Similar-to*: This relationship establishes a semantic "affinity" metric: From the user's point of view, certain models may show a similar behavior. This relationship is weighted, i.e. a positive distance is given on each arc. Transitive closures are possible as discussed by Prieto-Diaz and Freeman (1987) and Pintado and Tsichritzis (1989).

*Qualitative-abstraction-of*: This relationship links qualitative models to corresponding quantitative models. It may again be used for the inference of virtual arcs.

## 3 STORAGE, MODIFICATION, AND AGGREGATION

The operations defined on the model base are:

- definition of a new class,
- deletion of a class or model,
- editing of a class definition,
- instantiation of a class, and
- definition of relationships between entries.

These operations allow the user to define and manage

his own lattice of model classes. In fact, he can specify new classes by progressively specializing existing ones (both instantiating a set of class attributes or adding new ones) or by generalizing them (deleting some attributes of the class from which it is derived) (see, for instance, Shriver and Wegner (1987)). Though each class has a unique "parent" class, in the sense that the user derives it from a single existing class, he must be allowed to arbitrarily modify them during the editing process. For instance, the class of water quality models will contain a state variable representing the oxygen content which will be inherited by all its subclasses. The subclass of river water quality models may contain in addition a variable (and the related equation) for the algal biomass, but its subclass representing a specific mountain river may not need it.

Though in principle the classes of static, basic, and compound models may also form a hierarchy, it has been decided to represent them as separate descend of the class "model", since in all the implementations developed, the methods to create a new class significantly depends on the reserved class and this clearly separates the lattice of the user classes inheriting these methods. The differences in these methods depends on the fact that, for basic and static model classes, the user must supply a strongly structured set of information that can be easily represented by a frame to fill in, while for compound models, also the topology of the system (i.e., the connections between component models) must be given, as will be described in the following section.

Aggregation of given parts in order to produce a new product is an important feature of any environment supporting reusabilty. In our case, compound models built of stored or newly defined component models may be defined interactively by a special purpose graph editor.

To specify the topological information of compound models, the user first selects component model classes from the model base and then must define, in graphical form, the connections between input and output variables of the components. For compound models of deeper structure (i.e. having compound models as components), both, the bottom-up and the top-down approach must be supported. While the bottom-up approach is straightforward - connecting existing models yield a new (compound) model - the top down definition uses "virtual" model classes with a specified number of input and output variables, but without internal structure. The instantiation of such a virtual class may be completed at any time prior to simulation. An example of a compound model defined in this way is presented in Figure 2.

In practice, during construction of a compound model, the model base manager must supervise the linking of variables in order to ensure unit-compatibility, thereby supplying conversion functions on the links where necessary, and completes the user's specification by inferring the compound model's attributes from its components. The sets of input and output variables is computed as the unions over all corresponding sets of the component models, excluding those variables that are linked to each others. Similarly, the sets of state variables and parameters are the unions of the respective component sets. Name clashes can be resolved by explicit qualification with the name of the component model.

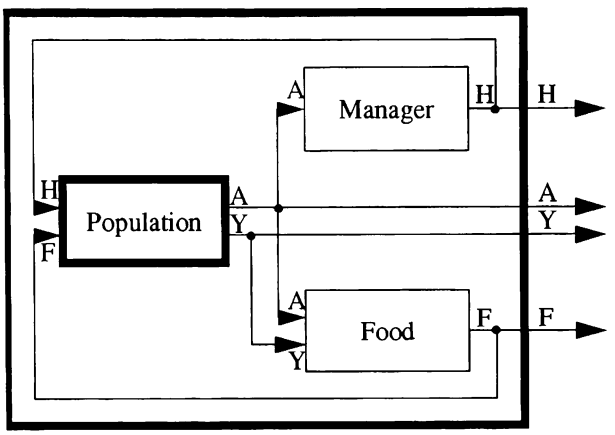


Figure 2: A Compound Model

During the aggregation process, the possibly different time units and time steps of the components are taken into account and the time step of the compound model is computed as the greatest common divisor of all the components' time steps. However, the simulation itself should proceed for each component with its own time step, thus allowing the required precision (discretization can be coarser for blocks with a slower dynamics), with a minimum of computation. In quite the same way, if the user selects different simulation methods for different submodels (say, for instance, Euler and Runge-Kutta), each submodel will be integrated with its own method, unless specified otherwise during the instantiation.

Non-virtual relationships which are not collected automatically during the construction process must be defined explictly. This is done in another graphics oriented component of the user interface, an interactive graph browser, in a way similar to the browsing process described in the next section.

## 4 RETRIEVAL

Retrieval is necessary, whenever a user needs a model for simulation or construction of a compound model. The larger the model base, the better the likelihood of existence of an appropriate model. However, retrieval becomes an issue when the model base is populated by, say, hundreds of entries. Two types of unsatisfying results may arise from any query:

- The resulting set of candidates is empty or does not contain useful candidates.
- The resulting set is too big to be inspected thoroughly.

In the first case, it is usually easy to generalize the query, which, however, often yields case two in turn. In the second case, restricting the query is often not appropriate, because of the risk to eliminate candidates of interest from the target set. Consequently, we designed distinct interfaces, which may collaborate with each other. The first interface supports "classical" queries regarding properties of models, while the second interface provides the user with an associative navigational tool to browse through the highly structured model base. Both interfaces operate either on the whole model base or on subsets thereof, stemming from previous retrieval steps.

The query interface supports more or less arbitrary complex query expressions over the structural and semantic attributes as defined above, thus enabling the user to retrieve models satisfying the constraints imposed by a specific problem at hand. The result of a query, a set of entities together with the relationships holding between them (which are invisible at this stage), may be refined subsequently in the following manner: The set may be modified by the results of another query, combining the two results with standard set operations, or it may be explored interactively via the navigational interface (see below).

The navigational interface features an interactive graph browser. It inspects a "current" node at a time. This node is displayed together with its "environment", which is defined as the subgraph induced by all paths of a (parameterized) maximum length, starting from the current node and using arcs from a user-defined subset of arc types. The arcs used to establish this environment represent the relationships defined in the model base (either explicitly or virtually). The user may interact with the multigraph in the following way:

- Selection of a new "current" node and re-display of the environment. This is the main browsing activity.
- Including/excluding certain arc types for the subsequent interaction(s).
- Changing the diameter (the maximum path length) of the environment.
- Inspecting/editing node information (and/or arc information, where applicable).
- Combination with results of the query interface by highlighting nodes which are members of the respective result set or restricting the environment to the subgraph induced by this set.

The major strength of this approach lies in its staged nature: It enables the user to eliminate significant parts of the search space by defining a "filter", based on the structural and semantic properties of the entity looked for - as has been proposed by many authors. However, in a second step, the remaining part of the repository (i.e. the result of the query) may be inspected interactively, whenever it is too large to be grasped at once by the user, thus enabling the user to explore all semantic relationships defined until a suitable item is found.

Whenever a node is found to qualify for reuse, the user may switch to the construction interface (cf. Section 3) where modification or aggregation can take place. Alternatively, the experimentation subsystem may be invoked, where actual simulation / optimization runs are performed (Guariso et al. 1990; Guariso, Hitz, and Werthner 1993).

## 5 IMPLEMENTATION ISSUES

The data model as sketched in Section 2, especially with its type hierarchies, aggregation hierarchies and semantic associations, calls for implementation on top of an object oriented DBMS. While this is certainly on the agenda for future developments, until now, the authors have not yet pursued this idea (even though the last prototype developed has been implemented in C++). This is mainly due to the practical reason that object oriented DBMSs have not yet been available on most hardware platforms employed so far (cf. Section 6). Moreover, while we could have got along with uncertain performance, prizing of these products is still an issue. Thus, we decided to transform our object oriented data model to a flat relational schema and implemented it by means of a relational DBMS.

To keep the system as general as possible (and thus reusable itself) the database schema implemented only reflects an abstract meta data model. Most of its concrete aspects are handled as data entries in meta relations. Relevant parts of the corresponding ER-model is shown in Figure 3 (excluding attributes):

An *Entry* (described by *Keywords*) is either a model instance or a model class, as defined by the corresponding *Entry Type*. Models contain *Variables* of different types (*Variable Type*: input, output, state, and - somewhat arbitrarily - parameters). *Variables* are measured in *Units* which in turn define the physical *Dimension*. Within compound models, *Variables* may be connected to each other via (physical) *Links*. Apart from these physical links, *Entries* are linked via *Relationships* of different *Relationship Types*. *Restrictions* may be imposed on *Relationships* depending on their *Relationship Types* (e.g., is-a may only connect two class entries). Virtual *Relationship Types* are defined by a *Derivation Rule*.

Relationship types may thus be defined dynamically by an administrator. Consequently, derivation rules of virtual relationships must also be entered as data. To sup-

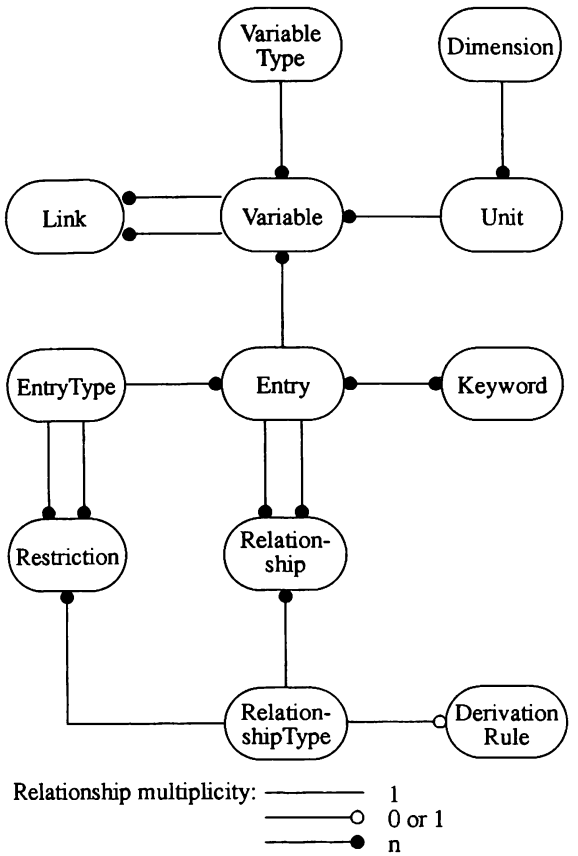


Figure 3: "Meta" ER-Model

port this, derivation rules are simply coded in Prolog and entered as character strings into the data base. When the system is about to derive virtual relationships, these rules together with Prolog facts corresponding to the "Relationship" relation of the current data base are fed into a Prolog interpreter which is used as an inference machine. The results are read by the system and presented to the user. In the remainder of this section, we explain this process and some possible shortcuts.

For example, assume that the *similar_to* relationship as defined for classes may be derived automatically at the instance level, if it holds for the corresponding classes. Consider two classes A and B which are known to be *similar_to* each other, and two instances a and b thereof for which no relationship is explicitly given. The fact base thus looks something like this:

```
instance_of('a','B').
instance_of('b','B').
similar_to('A','B'). % similar_to
similar_to('B','A'). % is symmetric
```

The derivation rule specified for similar_to would be:

```
similar_to(X, Y) :-
 instance_of(X, XC),
 instance_of(Y, YC),
 similar_to(XC, YC).
```

To find all entries "similar_to" instance a, SIMOBA would define the goal

```
similar_to('a', X)?
```

Prolog would give the (in this tiny case unique) answer

```
X = b
```

As the structure of the derivation rule given above has turned out to be very common among the rules currently implemented, meta rules "check(Pred)" are used to factor out this commonality:

```
check(Rel(X,Y)) :- Rel(X, Y). % use
 % facts or ortho-rules, if provided
check(Rel(X,Y)) :-
 undirected(Rel),
 Rel(Y,X). % for symmetric relations
check(Rel(X,Y)) :-
 derived_via(Rel, Via),
 Via(X, X0),
 Via(Y, Y0),
 check(Rel(X0, Y0)).
```

For standard rules of this type, the user need not specify the Prolog code proper, but must only provide some additional database entries. Returning to the example, the facts are:

```
undirected(similar_to).
derived_via(similar_to,instance_of).
```

The corresponding tables of the data base look essentially like the example given in Figure 4 (in which D stands for directed, V for virtual, B for built-in).

| Relation | D | V | B |
|---|---|---|---|
| similar_to | N | Y | N |
| instance_of | Y | N | Y |
| is_a | Y | N | Y |

| Relation | DerivedVia |
|---|---|
| similar_to | instance_of |
| similar_to | is_a |

Figure 4: Relations Describing Standard Rules

Finally, the goal is changed to

```
check(similar_to('a', X))?
```

which, of course, yields the same answer as before. Other standard derivations can be handled in a similar way, reducing the overall burden of the administrator, while keeping the system as flexible as possible.

## 6 CONCLUDING REMARKS

The ideas presented herein have been implemented as four working prototypes for different software and hardware environments, with emphasis laid on different parts of an integrated environment. Figure 5 summarizes the historic development, which is not yet complete, as the overall integration of all subsystems is still missing.

Apart from this task, the following enhancements of the retrieval process are planned for the near future:

- Full text retrieval in associated documents
- Queries based on the value ranges stored with the variables

On a side track, we also try to adapt the tool presented to support reusability in the domain of general software components.

We also hope to get an approximative answer to one of the main questions related to reuse library systems like the one proposed: Are users willing to spend the additional amount of work to enter their artifacts into the library? If the contents of a reuse library does not reach a "critical mass", its usefulness is likely to vanish. We do not know the answer yet, but we tried our best to eliminate any unnecessary administrative overhead.

| System | Year | Hardware | Software |
|---|---|---|---|
| InGenOSS | 1987 | Xerox 8088 | Interlisp DOS |
| QualSim | 1989 | Xerox | Interlisp Loops |
| MoBase | 1989 | Xerox | Interlisp |
| MoNet | 1990 | Apollo | C Unix Dialog |
| SiMoBa | 1993+ | $80\frac{3}{4}86$ | C++ Windows SQL Prolog |

Figure 5: History of Prototypes

## ACKNOWLEDGEMENTS

The authors thank Professor Giorgio Guariso of the Politecnico di Milano whose long lasting co-operation has laid a solid foundation for the work presented.

## REFERENCES

Guariso, G., M. Hitz, and H. Werthner. 1988. A knowledge based simulation environment for fast prototyping. In *Proceedings of the European Simulation Multiconference*, Nice 1988.

Guariso, G., M. Hitz, and H. Werthner. 1989. An intelligent simulation model generator". *Simulation* 53(2).

Guariso, G., M. Hitz, M. Schauer, and H. Werthner. 1990. MoNet: Eine Simulationsumgebung für hierarchische Input/Output- Modelle. In *Proceedings of the 5th Int. Symp. on Computer Science for Environmental Protection*, Vienna, September 1990. Springer.

Guariso, G., M. Hitz, and H. Werthner. 1993. An integrated simulation and optimization modelling environment. Technical Report TR 93-04, Institut für Statistik, Operations Research und Computerverfahren, University of Vienna, Austria.

Guariso, G., A. Rizzoli, and H. Werthner. 1992. Identification of model structure via qualitative simulation. *IEEE Transactions on Systems, Man and Cybernetics* 22(5).

Guariso, G., and H. Werthner. 1988. A software base for environmental studies. *Computer Journal* 31(6).

Hitz M., and H. Werthner. 1992. A graph oriented approach to enhance reusability in *-bases. In *Proceedings of WISR '92, 5th Annual Workshop on Software Reuse*, Palo Alto, CA, October 1992.

Lung, C.-H., J. K. Cochran, G. T. Mackulak, and J. E. Urban. 1992. Empirically analyzing software reuse in a simulation environment. *Proceedings of WISR '92, 5th Annual Workshop on Software Reuse*, Palo Alto, CA, October 1992.

Miller, L. H., and A. Quilici. 1991. A knowledge-based approach to encouraging reuse of simulation and modeling programs. In *Proceedings of WISR '91, 4th Annual Workshop on Software Reuse*, Reston, Virginia, November 1991.

Nemec, T. 1993. Verwaltung wiederverwendbarer Software-Module am Beispiel einer Simulationsmodellbank. Master's Thesis, Institut für angewandte Informatik und Informationssysteme, University of Vienna, Austria.

Ören, T. I., D. G. King, L. G. Birta, and M. Hitz. 1992. Requirements for a repository-based simulation environment. In *Proceedings of the 1992 Winter Simulation Conference.*

Prieto-Diaz R., and P. Freeman. 1987. Classifying software for reusability. *IEEE Software.*

Prieto-Diaz, R. 1989. Classification of reusable modules. In *Software Reusability, Vol. I, Concepts and Models*, ed. A. Perlis and T.J. Biggerstaff. ACM Press.

Pintado, X., and D. Tsichritzis. 1989. SaTellite: A navigation tool for hypermedia. In *Object Oriented Development*, Centre Universitaire d'Informatique, Université de Geneve.

Reese R., and D. L. Wyatt. 1987. Software reuse and simulation". In *Proceedings of the 1987 Winter Simulation Conference.*

Shriver B., and P. Wegner (eds.). 1987. *Research directions in object-oriented programming.* The MIT Press, Cambridge, Mass.

Ulgen, O. M., T. Thomasma, and N. Otto. 1991. Reusable models: Making your models more user-friendly. In *Proceedings of the 1991 Winter Simulation Conference.*

Zeigler, B. 1976. *Theory of modelling and simulation.* John Wiley.

## AUTHOR BIOGRAPHIES

**MARTIN HITZ** is Assistant Professor at the Data Engineering Department of the University of Vienna. He received a Master's degree and Ph.D. in Computer Science from the Technical University of Vienna. His interests include information systems design, reusability, and object oriented software development. He is a member of the IEEE Software Reuse Working Group of the Software Engineering Standards Subcommittee.

**HANNES WERTHNER** is Assistant Professor at the Institute of Statistics and Computer Science, University of Vienna. He received a Master's degree and Ph.D. in Computer Science both from the Technical University of Vienna. His main research interests include simulation environments, OO design, information systems and multimedia.

**TUNCER I. ÖREN** is a Professor of computer science at the Department of Computer Science, University of Ottawa, where he leads the Simulation and Software Quality Assurance Research Group. For a long time, Dr. Ören has been envolved with the design and development of integrated and integrative simulation environments. As part of his recent research activities, he investigates the role of repositories to support reusability in the area of modeling and simulation.