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

HIERARCHICAL OBJECT NETS – A METHODOLOGY FOR GRAPHICAL MODELING OF DISCRETE EVENT SYSTEMS

Carsten Thomas

Research and Technology Department
Daimler-Benz AG
Alt-Moabit 91b, 10559 Berlin, Germany

ABSTRACT

In this paper we introduce Hierarchical Object Nets as a modeling methodology for discrete event systems based on the DEVS (Discrete Event System Specification) formalism. We explain the modifications we propose and show how the methodology may serve as a basis for an open visual object-oriented modeling and simulation system. Further, the advanced concepts of active simulation objects and the prototype oriented model base are explained. The advantages of the concept for modeling, simulation and evaluation are reviewed.

1 MOTIVATION

Modeling and simulation of large structured discrete event systems like automated manufacturing systems demand highly developed modeling formalisms and simulation tools. Simulation systems under consideration have to be applicable in very different fields, from abstract system design to detailed description of single components. The formal means of description has to be flexible enough to express structured systems of different character (discrete, continuous, and control processes). It should support modularization and model re-use as well as the choice of the level of abstraction, its change and the combination of different abstraction levels within one model. For application in industry, it must be possible to embed modeling and simulation tools into given engineering processes and to re-use already formalized knowledge. Contemporary concepts like graphical modeling as described by Ozden (1991) and interactive simulation as proposed by O'Keefe (1987) have to be considered. To our best knowledge, current simulation systems only partially fulfill these requirements.

Based on work done on object-oriented graphical modeling (Thomas 1991) we developed the concept of Hierarchical Object Nets (HON) as a basis for an open (i.e. modularly expandable) visual object-oriented modeling and simulation system. The approach is based on the DEVS (Discrete Event System Specification) formalism developed by Zeigler (1984, 1987). Some of its original concepts have been modified or expanded to raise the expressibility or ease the modeling, and to suit the needs of advanced simulation and animation.

2 A BRIEF LOOK AT THE DEVS FORMALISM

The DEVS formalism is the system-theoretic manifestation of the object-oriented modeling paradigm. A discrete event system can be viewed as a hierarchical aggregation of communicating system entities. The DEVS formalism supports the specification of modular discrete event models in a hierarchical manner, thus preserving the structure of the real system down to the desired degree of refinement.

The formalism permits the specification of basic model components, from which larger models are built, and the specification of hierarchical aggregations of these components. Basic model components (*atomic models*) are specified as:

$$M = <X, S, Y, \delta, \lambda, \tau>$$

- X: the set of input events;
- S: the sequential state set;
- Y: the set of output events;
- δ: the transition function, which may be separated into
  - an internal transition function $\delta_i : S \rightarrow S$ and
  - an external transition function $\delta_x : Q \times X \rightarrow S$ with the total state $Q = \{(s, e) | s \in S, 0 \leq e \leq \tau(s)\}$;
- λ: the output function $\lambda : S \rightarrow Y$;
- τ: the time advance function $\tau : S \rightarrow R^{+}_{0,\infty}$, mapping the state S to the non-negative reals.

Atoms can be connected to form *coupled models*. Coupled models are defined as:

$$N = \langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} \rangle$$

D: the set of component identifiers;
for each i in D,
  M_i: the DEVS for component i in D;
  I_i: the set of influencees of i;
for each j in I_i,
  $Z_{i,j}$: the i-to-j-output translation function $Y_i \rightarrow X_j$.

Zeigler (1987) showed that the DEVS formalism is *closed under composition*. That is, coupled models may serve as components in larger coupled models, thus permitting the specification of hierarchical models. A detailed description of the DEVS formalism can be found in (Zeigler 1984).

Zeigler (1984) uses abstract simulators to execute the models. Abstract simulators are an algorithmic description of how to carry out the instruction implicit to the models to generate their behavior. There is a one-to-one relation between abstract simulators and model components. For each component, there is one abstract simulator which sends and receives messages, calls transition functions and maintains event time information locally.

There are several simulation environments which are direct implementations of the DEVS formalism. DEVS-Scheme developed by Zeigler (1990) is a Scheme-(i.e. Lisp-)based environment employing the modeling power and expressive strength of an AI language. With DEVSIM++, Kim and Park (1992) duplicate the simulation part of this system as an implementation in C++ mostly to gain simulation execution speedup. The CommonLisp-based implementation STIMS-CLOS by Prähofer (1991) also gives proof of the rising interest in this formalism.

3 HIERARCHICAL OBJECT NETS

3.1 Introduction

The DEVS formalism has been developed as a system theoretic fundament for discrete event system specification. It incorporates a level of generality, expressive strength and flexibility not reached by other approaches. The formalism provides a basis for extending the view on models, allows handling them as knowledge used to answer a multiplicity of questions rather than as "just simulation models". However, when applied only to simulation problems the flexibility becomes a problem. This affects the ease-of-use of a DEVS-based simulation system and its simulation efficiency. We propose hierarchical object nets as a modification of DEVS in order to extend the formalism's practicability to make this powerful means available to practice.

The concept of Hierarchical Object Nets is a modification of the original DEVS formalism with respect to the functionality of coupled models, and message (i.e. external event) propagation. All implementation differences between realizations of the Hierarchical Object Net concept and systems based on the original DEVS and are stressing these modifications. The outcome from these differences is described in detail in Section 5.

3.2 Differences between HON and DEVS

A major difference between Hierarchical Object Nets and the original DEVS formalism is the functionality of a coupled model. In our approach, a coupled model is not just a container for its components, but may also have some "atomic behavior". That is, it may have internal and external transition functions, a sequential state, an output function and a time advance function. To mark the difference between a coupled model as defined by Zeigler and the modified definition proposed here, we call our version an *aggregate*.

Input events to the aggregate are filtered by the aggregate's external event function. They may or may not be propagated to the components. In case of propagation, the aggregate's output function is applied, the time advance for this operation is zero. External events of components which are directed to other components of the same aggregate are propagated directly, without activation of the coupled model. For this reason, all objects (aggregates as well as atoms) maintain lists of their influencees. Component output targeted to the outside of the aggregate is handled like an external input event for the aggregate (when viewed as an atom). It follows that this output may be filtered in the same way as the input events discussed earlier.

Dealing with events to and from aggregate components in this way allows flattening the model with respect to simulation and distributed time synchronization. The aggregate then serves as a gate between its components and the outside model environment as shown in Figure 1. Synchronization of distributed simulation may be done using well-known and efficient algorithms described elsewhere. Moreover, computational overhead resulting from the synchronization related message passing between coupled model and its components is minimized.

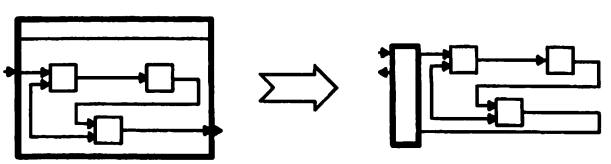

Figure 1: Flattening an Aggregate

As a disadvantage of this approach one could see the lack of an implicit tie-braking mechanism as known from DEVS. The selector proposed in the original formalism stresses the fact that the coupled model manages execution token distribution (and thus, synchronization) by supervising event propagation also between its components. However, in our approach, tie-braking mechanisms can be implemented as part of the model wherever necessary.

From the system theoretic point of view, every aggregate can be transformed into a combination of a coupled model and an atom. However, since the semantics of an aggregate are strongly related to the functionality of its atomic part, both parts should be viewed and handled as one entity.

3.3 Object Behavior

3.3.1 Active Objects

Zeigler introduced the concept of abstract simulators as explained in Section 2. This concept has been abandoned for the Hierarchical Object Net approach. Focusing only on simulation (i.e. the dynamics of a model), the division between the knowledge about a model and the execution of its implicit dynamics is not necessary. This step opens the way to the advanced concept of *active objects*. Active objects are "alive" from their instantiation up to their destruction. Editing, simulation and animation stimuli are handled locally by the objects themselves in the way external events are handled. The behavior of objects in those three areas of activity is defined by their appropriate external transition function section. Thus, relying on the object behavior, a simulation system may handle modeling, simulation, and animation in a fully decentralized way.

3.3.2 Editing Behavior

The editing of objects defines or alters their state. For atomic objects, these are the variables describing the state, for aggregates this is the internal structure, consisting of components and their internal coupling.

Using active objects we are able to decentralize the editing, i.e. to perform the modeling without using some central model editor. Instead, the individual objects use their own editing methods, thus individually controlling the definition and modification of their state.

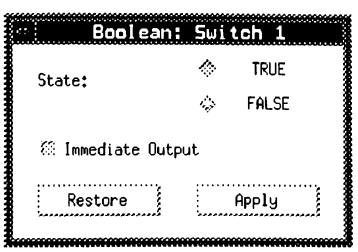

Figure 2: Atom State Editor Window

The attributes of atoms are usually changed by means of dialog boxes as shown in Figure 2. Figure 3 shows a graphical net editor window as used for an aggregate. User interaction in the editors is handled by the objects like external events changing the objects state.

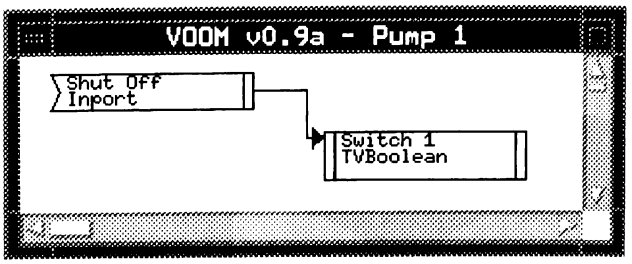

Figure 3: Aggregate Net Edtor Window

Applying this concept, a model is an aggregate object with a corresponding editor window. When the user inserts a new component into this window, this component is immediately instantiated, making its own editor window or dialog available at once. If the component is an aggregate, the model now may be further refined.

3.3.3 Simulation Behavior

In line with the object-oriented paradigm simulation is also locally controlled. Objects (atoms and aggregates as well) maintain local event lists and local virtual time information. Incoming events are processed according to their time stamps; generated local events are scheduled and output events are produced. The way in that current object state, local virtual time and the information contained in the input events (the time stamp and additional information) are combined to obtain internal and output events describes the objects simulation behavior. The adherence to the causality constraint (event processing in their timely order) is locally controlled. This is a major difference to the original algorithm described by Concepcion (1985) and Zeigler (1990), where the event synchronization is supervised by the coupled model for its direct components.

3.3.4 Animation Behavior

The modeling paradigm stressed in this concept leads to the idea that a hierarchical, object-oriented animation may fit the hierarchical model best. As implemented here, it provides a basis for an easy-to-implement, high-quality visualization of the simulated process.

Atomic objects have methods to represent themselves and their state during animation. For instance, this can be done by naturalized shapes (like switches for a Boolean object). Usually, aggregates do not visualize themselves, but merely provide an animation canvas for their components which can be moved and concealed, and thus unify

the animation methods of their components (Thomas 1991).

The object-oriented animation serves as an experimentation interface to the user of the simulation system. The direct connection of animation and simulation objects also enables run-time interaction not available with other systems.

4 PROTOTYPES AND CLASSIFICATION

The characteristics of Hierarchical Object Nets described above allow a comfortable use of top-down modeling principles. To be able to use bottom-up modeling variants and re-use models implemented earlier, a simulation model management system has been implemented. The prototype/instance concept described in this section is based on an object-oriented database management system.

An object prototype is a kind of "recipe" for an object instance. It includes descriptions of the object's initial state (i.e. the initial values of state variables for atoms or the initial component structure and coupling of an aggregate, respectively) and its message interface. The latter consists of a description of available input and output ports and the message types which are accepted or sent. This information can be used for static message type checking during coupling scheme editing.

Simulation object instances are "working copies" of prototypes. The state of instances can be altered by editing or during simulation without interfering with other instances derived from the same prototype. The term "object" is used throughout this paper (as usually done in literature) for prototypes and for their instances as well.

Instances must be transformed into prototypes to be available for re-use. Then, information about the object's state or structure and the object's message interface is saved in a database and the usage information of the prototypes is updated for all affected models in the database. Changes to prototype definitions are observed by the model management system and may be propagated automatically to all instances of the respective prototype.

Further to the use in coupling scheme checking, the interface information contained in object prototypes is used as the basis of an interface-oriented object classification concept. This approach is different from the *System Entity Structure* concept developed by Zeigler (1984). While the taxonimy relations contained in a SES are built upon implementational inheritance knowledge, our classification relies on inheritance of the models port and message interface (i.e., from the system-theoretic point of view, on input and output set inheritance). A prototype may be a decentant of another prototype if its interface at least comprises the interface of its parent. However, since the scheme implements a half-order, no assumtions can be made about the taxonomic relations of classes residing in different paths of the inheritance tree.

5 ADVANTAGES OF THE CONCEPT

5.1 Effects on Modeling

Hierarchical Object Nets belong to the group of net-based graphical means for model description. This leads to the use of a purely graphical modeling environment without the necessity of any textual means of description. One major advantage of graphical net editors and dialog boxes for parametrization is that these tools may be understood intuitively; they gain the advantages of visual programming environments described by Glinert (1990). Moreover, the graphical environment in connection with the interface description based message type checking prevents a variety of errors during modeling. The learning curve for such a system is much steeper than that for a language based, textual system.

Elements in Hierarchical Object Nets may be interchanged regardless of whether they are aggregates or atoms, as long as the used ports of the objects' interfaces are compatible (i.e. belong to common anchestors). This modularity is based on Zeigler's proof that the DEVS formalism is closed under composition and on the object classification scheme described above. The potential semantic identity between different implementations of model elements can be used to maintain different versions of model components to answer different questions. Also, the level of abstraction can be altered in this way and thus the system allows stepwise model refinement.

Prähofer (1992) explaines three different levels of complexity for the implementation of variable structure models (component exchange, component and coupling creation and destruction, and changes forced from outside a model). The interface-oriented simulation object classification may provide a basis to address all of these problems. Employing this concept, even the use of simulation objects as message parameters seems to be possible.

The hiding of the object's implementation by its interface (corresponding to the encapsulation in object-oriented modeling) is the basis for the definition of model libraries. These libraries are collections of aggregate prototypes. The inclusion of application-specific model libraries opens the way to extend of the simulation system in a modular fashion. The availablity of domain-specific modules allows the modeler to stay within the usual level of abstraction and to deal with simulation theory only marginally. Implementation details of the library objects used will be of no interest in most cases due to the completeness of description given by interface and functional specification.

Stressing the fact that all input and output to and from aggregate components is filtered by the aggregate, the im-

plementation of partially automated rapid simulation seems possible for some application areas. The aggregate observes input and output and does behaviour adaption. When the desired confidence level is reached, the functionality of the aggregate's atomic part replaces that of the contained coupled model.

Usually, coupled models will be implemented by means of the aggregates standard net-editor windows. Components to be instantiated and the inner coupling of aggregates is put in directly. For special application areas, this instantiation and coupling process may be hidden to the modeler. Such special editor windows are useful to allow the modeler to stick to description means which are better known (like Petri nets or others) or more applicable under certain circumstances (e.g. when they allow to reuse knowledge already formalized within an engineering process). Internally, an automated projection of the application specific means of description onto Hierarchical Object Nets has to be done; the appropriate objects have to be instantiated and connected. The "look-and-feel" of the adapted modeling approaches is kept, and the appropriate set of interpretative analytical means can be provided. Models can be iteratively refined while changing the modeling approach more than once. E.g., a HON model of an automated assembly line may contain a control specification expressed as a Petri net, which in turn contains places made up from queues implemented as atomic HON objects. Whether an (atomic or aggregate) object may serve as a component in a model of a different type depends on its interface.

The proposed way of incorporating different modeling approaches is reminiscent of Hybrid Heterogeneous Hierarchical Modeling as proposed by Miller and Fishwick (1992). However, due to its focussation on simulation aspects our concept lacks means for hybrid (combined symbolic, numerical, and knowledge-based) analysis methods.

5.2 Effects on Simulation

The immediate availability of simulation objects instantiated with a model editor retracts the borders between editing process and simulation. Models may be executed in early design stages; and models can be run, stopped, altered and further executed without re-compilation and re-start. In this way, the turn-around times for error tracing and correction can be reduced.

Another positive outcome for debugging is, that models can be animated using the model editor. Opening some of the editor windows, the modeler selects objects to be observed and may set simulation breakpoints.

The instantiation and destruction of aggregate components and inner coupling is an aggregates reaction on stimuli initiated by the user through the editor window. These stimuli may also be produced by the executed model itself, thus changing its own structure. This feature of model driven self-restructuring permits e.g. the implementation of evolutionary or learning models.

The concept of Hierarchical Object Nets allows the modeler to choose the desired level of abstraction. This, in connection with the partially atomic nature of aggregate objects can be the basis for an automated rapid simulation system. For rapid simulation, fine-grained structured aggregate representations of systems are replaced by their atomic functional counterparts. Behavior adaptation can be at least partially automated when the aggregates themselves are enabled to supervise incoming and outgoing external events. Even the transition from the structural to the functional representation and back could be automated for some applications.

Localization of simulation control within the objects provides an ideal basis for distributed simulation. The partially atomic nature of aggregates allows to flatten the object hierarchy with respect to distributed model time synchronization. As described earlier in this paper, standard synchronization techniques may be used; model modularization even allows the mixing of different synchronization algorithms within one model to gain optimal results in heterogeneous computing environments.

5.3 Effects on Result Evaluation and Presentation

The combination of raw simulation output to gain usable simulation results and their representation depends very much on the model and the simulation goal. Often, these results must be combined with information only loosely related to the real system which has been modeled or must be collected over several simulation runs. Most known simulation systems tell between simulation model on the one hand side and input generation, result evaluation and presentation on the other. As a variant of the DEVS formalism, Hierarchical Object Nets provide a basis for the implementation of Zeiglers (1984) experimental frames, thus permitting the definition of the experimentation shell for a model using the same methodology as for the model itself. Integrated object-oriented animation as briefly described in this paper gives the chance to extend the experimental frame to serve as an visual interactive experimentation environment.

Another advantage gained from the proposed animation concept is that the implementation effort for a process-oriented high-quality on-line animation is relatively low in comparison to other known simulation systems. The animation implementation is restricted to the positioning of the object shapes on an animation canvas and the definition of their visibility attributes. Model dynamics are automatically viewed by the objects themselves using their local state variables. In the current state of development, the animation concept does not allow animation

focus on "areas of interest". To have a work-around for such situations, animation is not restricted to the build-in concept. Using files and communication objects, simulation data may be given to other programs and can be visualized in post-run animations by other tools.

The application of the integrated animation gives some advantages over the use of external tools. During process-oriented animation, e.g., the freedom of interaction with the model can reach about the same level as during the model-oriented animation using the model editor. This allows the use of the integrated animation as a front end (e.g. operation panel) for the running model, so that a human being may interact with the simulated process. One may think of, for instance, an animated interactive control panel for a simulated manufacturing system. Then, the model may be used to test the panel design (its ergonomy) as well as to train the operator (e.g. on how to deal with dangerous situations for the system).

6 IMPLEMENTATION ISSUES

The concept of Hierarchical Object Nets is realized by a simulation system prototype currently under development at Daimler-Benz. To gain simulation speed, C++ has been chosen as implementation language. Since incorporation of new atomic models currently requires a re-compilation of the simulation system, we refrained from giving users a chance to define new atomic models. Instead, the library of atomic models is extensive and comprises even basic elements like variables (numbers, strings), containers (matrices, stacks) and operators. The model base management system sufficiently supports the modeller in finding appropriate (atomic or aggregate) models. However, more flexibility may be gained by also incorporating a graphical approaches for atomic model definition like Harel's (1988) statecharts.

Further to using C++ as the implementation language, portability is maintained by the use of a graphic library which employs the graphic user interface of the respective platform's operating system and an object-oriented client/server database system. Functionality for decentralized editing, the simulator kernel methods for uniprocessor simulation and basic parts of the object library are already implemented. Current implementational work is concentrated on animation integration and object library extension. The applicability of the concept is tested by modeling and simulating a complex automated transport system. Further work will concentrate on simulator kernel improvement and probably the implementation of a distributed version of the simulator.

7 CONCLUSIONS

We see the concept of Hierarchical Object Nets and its realization by the simulation system prototype currently under development as a major step towards an easy-to-use yet powerful simulation environment. With its roots in the DEVS formalism, the approach bases on flexible and expressive description means. The realization integrates several promising directions of development like purely graphical object-oriented modeling, visual interactive simulation, hierarchical animation, and others.

Simulation systems developed on the basis of Hierarchical Object Nets could be applied to all kinds of simulation problems related to discrete event systems. However, the main strength of the prototypical system will lie in the combination of simulation and interactive run-time animation and the flexibility and ease-of-use brought to the modeling process. Typical applications will be found in mixed abstraction level structured system modeling and decision support related to such systems, in customer-salesperson communications, and in technical training environments.

REFERENCES

Concepcion, A. I. 1985. Distributed Simulation on Multiprocessors: Design, Specification, and Architecture. Doctoral Dissertation, Wayne State University; Detroit, MI

Glinert, E. P. 1990. *Visual Programming Environments: Paradigms and Systems*. Los Alamitos: IEEE Computer Society Press.

Harel, D. 1988. On visual Formalisms. *Communications of the ACM* 31: 514-530

Kim, T. G., and Park, S. B. 1992 The DEVS Formalism: Hierarchical Modular Systems Specification in C++. In *Proceedings of the 1992 European Simulation Multiconference*, 152-156, York: SCS

Miller, D. P., and Fishwick, P. A. 1992. Hybrid Heterogeneous Hierarchical Models for Knowledge-Based Autonomous Systems. submitted to *International Journal of Computer Simulation*

O'Keefe, R. 1987. What is Visual Interactive Simulation (And is there a Methodology for Doing it Right)?. In *Proceedings of the 1987 Winter Simulation Conference*, 461-464,

Ozden, M. H. 1991. Graphical Programming of Simulation Models in an Object-oriented Environment. *Simulation* 56: 104-116.

Prähofer, H. 1991. System Theoretic Formalisms for Combined Discrete-Continuous System Simulation. *International Journal of General Systems* 19: 219-240

Prähofer, H. 1992. Modelling and Simulation. In *CAST Methods in Modelling*, eds. F. Pichler, and H. Schwärzel, 123-241. Wien: Springer-Verlag.

Thomas, C. 1991. Visual Object-oriented Modeling – A Proposal. Technical Report No. CMU-LASIP-91-05, Carnegie Mellon University. Pittsburgh, PA.

Zeigler, B. P. 1984. *Multifaceted Modeling and Discrete Event Simulation*. London: Academic Press.

Zeigler, B. P. 1987. Hierarchical Modular Discrete Event Modeling in an Object-oriented Environment. *Simulation* 49: 219-230.

Zeigler, B. P. 1990. *Object-oriented Simulation with Hierarchical, Modular Models – Intelligent Agents and Endomorphic Systems*. London: Academic Press.

AUTHOR BIOGRAPHY

CARSTEN THOMAS is researcher at the Daimler-Benz Systems Technology Research Group in Berlin, Germany. He graduated in Automation Technology and Cybernetics at the Chemnitz University of Technology (Germany) in 1991 and is currently persuing a doctoral degree in Electrical and Computer Engineering at the Technical University of Braunschweig (Germany). His research interests are focused on modeling and simulation of highly automated systems, particularly on hierarchical multiformalism models and their use in system engineering frameworks.