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

VISUAL MODELING OF DEVS-BASED MULTIFORMALISM SYSTEMS BASED ON HIGRAPHS

Herbert Praehofer
Dietmar Pree

Systems Theory and Information Engineering
Johannes Kepler University
A-4040 Linz, Austria

ABSTRACT

This paper presents a graphical modeling method and tool for DEVS model and DEVS-based combined discrete/continuous model specification. In DEVS-based modeling, atomic model behavior specification is organized around different phases which define a partition of the state space of the model. The phase transitions depict the qualitative state changes and naturally lend themselves to be represented by a state transition diagram. Our representation of these phase transitions is based on the higraph extension to conventional graph representations. In higraphs, the area of the diagram is used to represent set enclosure and exclusion and the Cartesian product which leads to remarkable reduction in the diagram's complexity. An interactive modeling tool based on the graphical representation developed is presented.

1 INTRODUCTION

Graphical representations have advantageously been employed to ease discrete event simulation modeling, model documentation, and model communication. Dependent on the particular simulation world view in hand, different forms of representations have emerged. *Event graphs* (Schruben 1983) have been developed to model event-oriented models as graphs showing the events and event dependencies. *Activity cycle* diagrams (Poole and Szymankiewicy 1977) are capable representing activity-scanning models by showing the cycles of activities the various entities in the model traverse. *Block diagrams* (Schriber 1977) or *process networks* (Pritsker 1977) have made the programming of process-interaction models popular. These diagrams are flowcharts which show the movements of the entities, usually the jobs, through the various operations of the system. Cota and Sargent introduced a new version of the process world view (Cota and Sargent 1992) and *control flow graphs* (Cota and Sargent 1989) as a means for its graphical representation. Although introduced as a conceptional tool for developing parallel simulation algorithms, they are a useful representation of simulation models. A control flow graph model is represented by an directed graph where the nodes depict various states of the model and the edges the event transitions.

The *DEVS formalism* (Zeigler 1976,1984,1990) being the system theoretic formalism for modular, hierarchical discrete event modeling and simulation has been extended by Praehofer (Praehofer 1991a, 1991b, Pichler and Schwaertzel 1992) to facilitate combined discrete/continuous multiformalism modeling and simulation. The *DEVandDESS formalism*, coming into being by a combination of the DEVS and the *differential equation specified system formalism* (DESS), allows the construction of atomic and hierarchically coupled combined models. In the DEVS formalism and its DEVandDESS multiformalism extension, atomic model specification is organized around various *phases* which denote global system states. Actually, the different phases of a model represent a partitioning of the state space of the system into mutual exclusive blocks where the different blocks identify qualitatively differing system behaviors. In combined modeling, the phases can be used to associate different derivatives with different phases and the phase transitions mean a change from one derivative to another. Oeren (1991) termed such an modeling approach *multimodeling*. Fishwick (1991) and Fishwick and Zeigler (1992) developed a methodology for qualitative model engineering based on the multimodeling approach.

While coupled DEVS and DEVandDESS models lend themselves to be graphically represented as block diagrams, a graphical representation of DEVS-based atomic models is still missing. The state space phase partitioning and the dynamic behavior specification organized around phases can serve as a basis for a

graphical representation. The phases and phase transitions are naturally represented by a state transition diagram similar to those of finite state automatons and the control flow graphs of Cota and Sargent. In the directed graph, the nodes depict the phases and the edges the event transitions. However, in contrast to finite state automaton diagrams, with the transition edges we have to associate the complex state event specifications of the events represented by the edges. There is general consensus that state diagram representations of complex systems get unwieldy through the unmanageable, exponentially growing multitude of states with a multitude of linking edges, all of which have to be arranged in a flat unstratified fashion, resulting in an unstructured, and chaotic state diagram. The *higraphs* extension (Harel 1987, 1988) of the conventional graph representations offers a solution to this problem. In higraphs, the area of the diagram is exploited to represent set enclosure, exclusion and intersection and the Cartesian product. The higraph representation nicely fits to our state space phase partitioning.

In this paper we develop a graphical representation for DEVS and DEVandDESS atomic models based on state space phase partitioning and higraph-based state transition graphs. This graphical form of representation provides a foundation for an interactive modeling tool implemented in Common Lisp / CLOS (Steele 1990) employing the *Common Lisp Interface Manager* (CLIM) toolkit (Lucid 92). The interactive modeling tool will serve as a user interface module of the *STIMS modeling and simulation environment* (Praehofer, Auernig, Reisinger 1993). STIMS is a new powerful, object-oriented modeling and simulation environment currently in development and is based on the DEVS and DEVandDESS system specification formalisms.

The outline of the paper is as follows: In section 2 we give a short review of the DEVS and DEVandDESS modeling concepts, discuss the role of phase partitioning in DEVS-based modeling, and introduce models owning several dimensions. In section 3 we discuss graphical representations of DEVS-based models and show how higraphs are used advantageously to achieve compact representations. In section 4 we present our CLIM realized modeling tool.

2 DEVS-BASED MULTIFORMALISM MODELLING

2.1 DEVS-Based Modeling Reviewed

Zeigler (Zeigler 1976, 1984, 1990) developed the *discrete event specified system* (DEVS) formalism as a mathematical basis for discrete event modeling. This formalism provides a formal representation of discrete event dynamic systems capable of mathematical manipulation and independent of any computer realization, just as differential equation specified systems serve this role for continuous systems.

In the DEVS formalism, one has to specify basic *atomic models* and, by connecting together these basic models in a modular, hierarchical manner, one has to specify complex *coupled models.* A DEVS-based atomic model is a modular unit. It comprises input and output interfaces in the form of input and output ports through which all the interactions with the environment occur. The interior of the model is represented by state variables. The dynamic state behavior and its outside manifestation is defined employing two types of events. *Input events* lead to external event transitions, i.e., upon occurrence of an input event, the model transits to a state determined by the *external transition function.* The other type of events are *time scheduled, internal events.* For each state the *time advance function* defines the time interval to the next internal event. When this time has elapsed, an internal event occurs. The system produces an output event and transits to a next state determined by the *internal transition function.* Specification of complex coupled models is done by connecting the output and input ports (modular coupling). Coupled models also have their own input and output ports and they can be used as components in bigger coupled models (hierarchical modeling). From their input and output interface, coupled models are not distinguishable from atomic components and, therefore, are reusable as building blocks in the same way as atomic models are.

Based on DEVS, Praehofer (1991a, 1991b) developed system specification formalisms and simulation concepts for combined discrete/continuous multiformalims modeling. He introduced the *DEVandDESS formalism* as a combination of the DEVS and *differential equation specified system formalism* (DESS) as a basic system theoretic formalism for combined discrete continuous modeling. A DEVandDESS atomic model has an input and output interface split up into discrete and continuous input and output ports through which all the interactions with the environment occur. It has a continuous as well as a discrete state set, both split up into state variables. The derivative function and the continuous output function inherited from the differential equation part are used to specify the continuous behavior. Discrete event behavior is inherited from the DEVS part. However, the DEVandDESS knows a further type of event, viz. *state events.* State events are internal

events caused by the continuous changes of the continuous inputs and continuous states and are modeled in the *state event transition function*. Conditions on continuous inputs and continuous states in the state event transition function may become true when continuous states and inputs change continuously. Whenever such a condition becomes true, a state event occurs. Similarly to time events, the system puts out the discrete output determined by the discrete output function and transits to a new state determined by the state event transition function. Strong influences exist between the discrete and continuous parts. On one hand side, the continuous behavior may depend on the current discrete state. On the other hand side, the changes in the continuous states and inputs trigger state events. The events can change the discrete as well as the continuous states leading to discontinuous jumps in the continuous trajectories.

The DEVandDESS coupled model formalism facilitates modular, hierarchical coupling of components which can be either of the discrete, continuous, or combined type. Analogous to DEVS, coupling of the components of different types is done simply by connecting their output and input ports. Couplings from discrete outputs to continuous inputs are allowed. In such a coupling, the event outputs are interpreted as piecewise constant, i.e., an event output determines a constant output value until the next event. However, a coupling from a continuous port to a discrete port is not allowed since the continuous trajectory would imply an infinite number of external state transitions.

2.2 State Space Phase Partitioning in DEVS-Based Modeling

Conventional discrete event modeling approaches and simulation languages emphasize the concept of event, activity or process and de-emphasize the concept of state. The DEVS formalism, however, originating from the systems theory background, emphasize the notion of state. In the DEVS formalism an atomic model dynamic behavior specification is organized around the *phase* variable which denotes some sort of global state the system stays in. Depending on the current phase of the system, it will react differently to external inputs and occurrence of internal events. In appendix A we show a model of a preemptive server which is structured along different phases. The model is either *idle*, or busy with a low priority job (*busyLP*), or busy with a high priority job (*busyHP*), or it may service a high priority job but a low priority job may be *preempted*. Depending on the current phase, the reaction to external inputs and internal events differs.

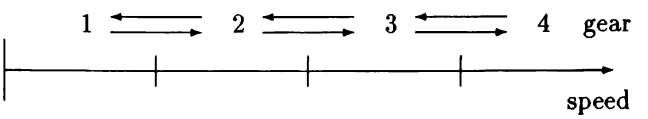

Figure 1: State Partitioning of Automated Transmission Vehicle

In DEVS modeling, the phase actually defines a partition of the state space of the model, i.e., the different phases indicate different, mutual exclusive blocks of the state space. So, in the preemptive server model, the phase *idle* represents that subset of the state space containing that single discrete state, where both queues, the queues with low priority jobs and high priority jobs, are empty. The phase *busyLP* now represents that possible infinite subset where no high priority job is in the system but there is at least one low priority job. Similarly, the phases *preempted* and *busyHP* define the subsets of states where there are high priority jobs and a low priority has been and has not been preempted.

In combined DEVandDESS modeling these issues are getting even clearer. Here the phase variables often are used to define the partition of the continuous state space or they are used to define the systems current discrete input value. In any case the phase transitions are done by discrete event transitions and signify a qualitative change in the dynamic behavior of a *multimodel*. The transitions are either external when the phases depend on the input, time scheduled when they depend on particular times, or state event when they depend on particular values of the the continuous state space. Let us clarify these issues by considering two similar simple models, viz. a vehicle with a stick operated transmission system and a vehicle with a rudimentary automatic transmission system which changes gears at particular speeds only. The phases of the system obviously are given by the different gears which determine different system behaviors. In the model of the hand-operated system, the gears are determined from outside. Thus the phase transitions are determined by the external transition function. In the model of the automatic system, however, the gear changes occur when the speed reaches certain thresholds. The phase transitions are modeled by state events in the state event transition function. The different gears are directly associated to certain subsets of the continuous speed variable as depicted in figure 1.

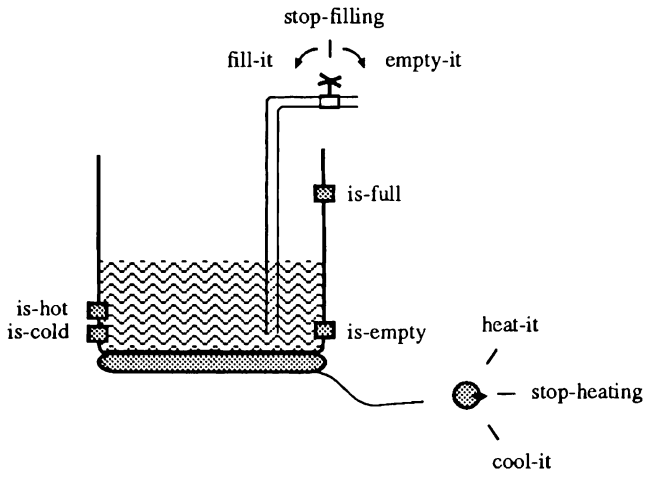

Figure 2: Boiling Water Pot System

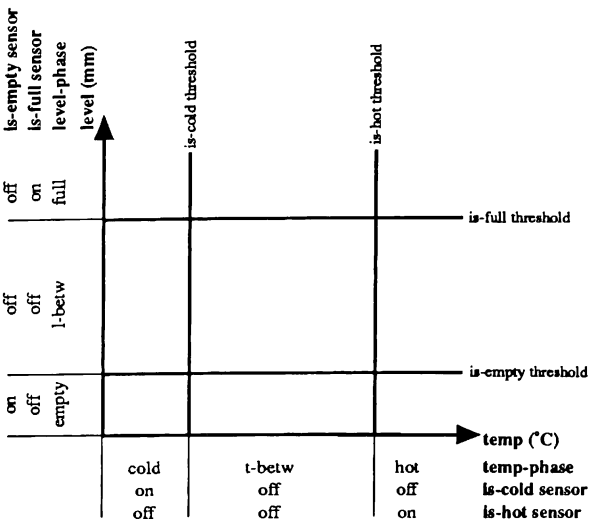

Figure 3: Two-Dimensional State Space Partitioning of Pot System

2.3 More-dimensional State Spaces in DEVS-Based Modeling

So far DEVS modeling has concentrated on systems with one-dimensional state space only, i.e., systems which only have one phase variable and the partitioning is one-dimensional. However, in continuous and combined modeling, systems with one dimension are the exception. Most systems show several, if not a number of independent dimensions. More-dimensional combined models usually also have independent phase partitions for the different dimensions. This motivated to introduce a new formalism which is a specialization of the usual DEVandDESS formalism. We call it *n-dimensional DEVandDESS* and it is characterized by owing several dimension and for each dimension *dim* there is one

- continuous state variable *dim*,
- one phase variable *dim-phase*, and
- one sigma variable *dim-sigma* to define the time to the next time event relative to that dimension.

Figure 2 shows a two-dimensional system of a pot which can be heated and cooled, filled and emptied (see Praehofer, Bichler and Zeigler (1993) for a more detailed description of the system and for event-based control of the system). The two dimensions are the *temp* and the *level* dimension representing the liquid level and the liquid temperature, respectively. The system has two discrete command inputs - the *heat-com* and the *fill-com* input - with three different commands for each, viz. *heat-it*, *cool-it*, *stop-heating* and *fill-it*, *empty-it*, *stop-filling*, respectively. The system's discrete outputs are given by four simple threshold sensors, viz. *is-cold*, *is-hot*, *is-empty*, and *is-full*. The values for output sensors are *on* and *off* and they react at particular threshold values of the two continuous state variables *level* and *temp*. Each state dimension is partitioned into three mutual exclusive blocks according to the threshold sensor output values. Figure 3 shows the state partitioning. There are 3 times 3 mutual exclusive blocks which have different sensor output values and which are denoted by the phases *cold*, *t-betw*, *hot* and *emtpy*, *l-betw*, *full*, respectively. Although the continuous system variables influence each other, the state events modeling the phase transitions are independent in the two dimensions.

Although most important for combined modeling, n-dimensional models can also advantageously be employed in pure discrete DEVS modeling. Different dimensions in DEVS models should be employed if several independently executing processes can be identified in one atomic model. For example, in a multiserver system modeled as one atomic model component, the different servers are independent executing components only interfering through the common waiting queue. Each server has its own phase and sigma variable. The sigma variable for one server defines the time to next end-of-service event for the particular server. The time to the next internal event of the whole multiserver model, i.e., the value of the time advance function, is given by the minimum of

the sigma values over all servers.

3 HIGRAPH-BASED PHASE TRANSITION DIAGRAMS

3.1 DEVS-Diagrams

Graphical representations are advantageously employed to ease simulation modeling and model documentation. As conventional simulation modeling emphasizes the concept of event, activity, or process, their graphical representation also are based on the notion of the event, activity, or process. Our approach naturally lends itself to be represented as a finite state diagram, i.e., a directed graph where the nodes represent the different phases of the model and the edges represent the event transitions. We call this representation *DEVS-diagrams*. However, in contrast to the usual finite state transition diagram of finite state machine automaton, for *DEVS-diagrams* complex procedures are associated with the event edges. Only the phase changes are represented by the edges. In background with each transition edge there is a complex state transition affecting the arbitrary complex state space and depending on an arbitrary complex condition.

To represent purely discrete DEVS models, the nodes denoting the different phases of the model are linked by two types of transition edges representing the internal and external event transitions, respectively. With each internal transition edge we associate the following code:

- a condition which is tested before the event is selected,
- a priority value to arbitrate in case of several executable internal transitions,
- next state values for a number of state variables, and
- output values for different output ports.

With each external transition edge we associate the following code:

- an input port,
- a condition which is tested before the event is selected,
- a priority value to arbitrate in case of several executable external transitions, and
- next state values for a number of state variables.

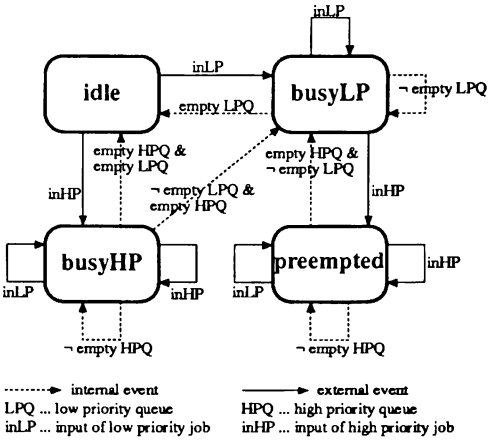

Figure 4: DEVS-Diagram for Preemptive Server Model

With a DEVS-diagram specified model then we associate the following dynamic behavior: A time scheduled internal event is executed when the time advance value has elapsed. The conditions of the internal event edges starting at the current phase node are tested. The transition edge whose condition evaluates to true and with the highest priority value is executed, i.e., the next phase is entered as given by the edge, the next state values are assigned to the state variables, and the output events are generated. Upon the occurrence of an input event, the external transition edges are tested. The transition edge with the appropriate input port and the condition evaluating to true is selected and executed. Figure 4 shows the DEVS state transition diagram of our preemptive server model.

To model DEVandDESS systems, we employ one more event edge type, viz. edges for state event transitions. Similar to time scheduled transitions, with the state event transition edges we associate the following code:

- a state event condition testing one continuous state variable being greater or smaller a particular threshold,
- a further arbitrary condition which is tested before the event is selected,
- a priority value to arbitrate in case of several executable internal or state event transitions,
- next state values for a number of state variables, and
- output values for different output ports.

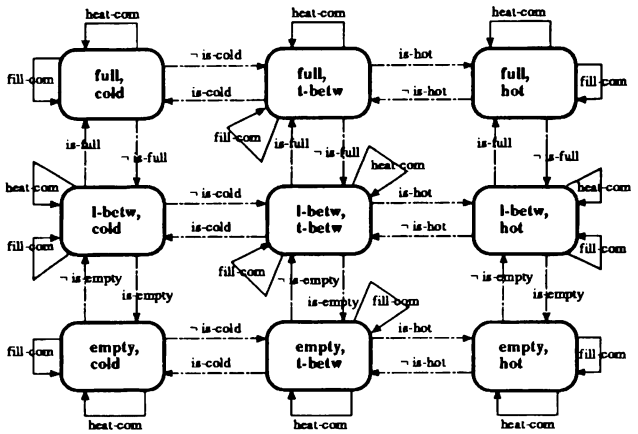

Figure 5: DEVS-Diagram of the Pot System

The state event is to occur, when the continuous value reaches the particular threshold. The conditions are tested and if true the transition is executed. In case of multiple executing transition edges, the priority value arbitrates.

Also, the continuous behavior of the continuous part of the formalism is modeled within the diagram. With the different phases, we specify the differential equations for the continuous state variables and the output values for the continuous output ports.

There is general consensus that the state diagram representation of complex systems get unwieldy through the unmanageable, exponentially growing multitude of states with a multitude of linking edges, all of which have to be arranged in a flat unstratified fashion, resulting in an unstructured, and chaotic state diagram (Harel 1987). Figure 5 shows a state transition diagram of the pot model represented in figure 2 with the state phase partitioning given in figure 3. One sees that already for this quite simple example, the diagram gets quite complex. All the possible combinations of the two phase variables have to be represented explicitly with all the possible transitions between each other. This leads to a lot of redundancy in event transition specification. In our example the state event transitions for the temperature and the level dimension are independent from each other and therefore could be specified independently. The input events, however, do not depend on the phases at all and only effect the derivative functions. But in the flat diagram all the transitions from every node have to be specified explicitly. The derivatives for the temperature and the level are all the same expect the phase *hot*. But to allow changing derivatives for different phases, we have to give the derivative to each phase node explicitly. To solve this problem, Harel (1987, 1988) introduced higraphs and higraph-based state transition diagrams.

3.2 Higraphs and Higraph-Based DEVS-Diagrams

Higraphs (Harel 1988) are a general extension of conventional graph representations by introducing means for representation of (1) set enclosure, exclusion and intersection and (2) the Cartesian product. This has been accomplished by exploiting the area of the diagram similar to the well-known concept of Venn diagrams. Higraphs have a lot of potential applications and have advantageously been employed for the *statechart* visual formalism for specification of complex reactive systems (Harel 1987) which is the basis for the *Statemate* design environment (Harel et al 1990). Our application to the specification of DEVS-diagrams is similar to statecharts, however, differs from it in the way transitions are specified and in further details.

In higraph-based representations, *atomic-blobs* in the form of rectangular shapes are used to represent basic mutual exclusive sets. In our application, atomic-blobs are used to represent the basic phases which are the blocks, i.e., mutual exclusive subsets, of the state set partitioning. They correspond to the nodes in our DEVS-diagram approach above. Atomic-blobs now can be clustered to compound blobs. A *cluster-blob*, called *or-blob* in the sequel, merely is the union of the atomic-blobs it encloses within its contour. This enclosure is a union operation and not a membership operation. With that, arbitrary combinations of atomic-blobs can be built. An or-blob forms a more abstract concept and can be used to represent equivalence relations of atomic-blobs in respect to a particular edge. An edge originating from an or-blob means that this edge applies equivalently for all the atomic-blobs enclosed in the or-blob. Figure 6 shows a higraph-based version of the DEVS diagram of the preemptive server model. As can be seen, an input event at port *inLP* has the same behavior for phases *busyLP*, *busyHP* and *preempted*. Therefore, the respective external event edge originates from an or-blob enclosing exactly these three phases which means that the transition applies if the system is in the *busyLP*, *busyHP* or *preempted* phase. An input event at port *inHP* will show the same reaction in the *busyHP* and *preempted* phase but a different in the *busyLP* phase. An or-blob with phases *busyHP* and *preempted* is the origin of the respective external event edge. Similarly, equivalent internal transitions are observed in phases

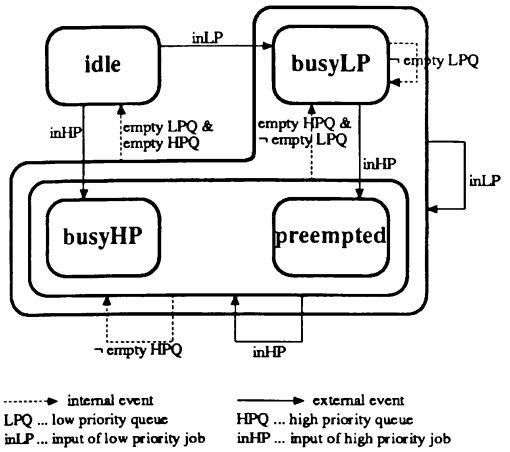

Figure 6: Higraph-Based DEVS-Diagram of Preemptive Server Model

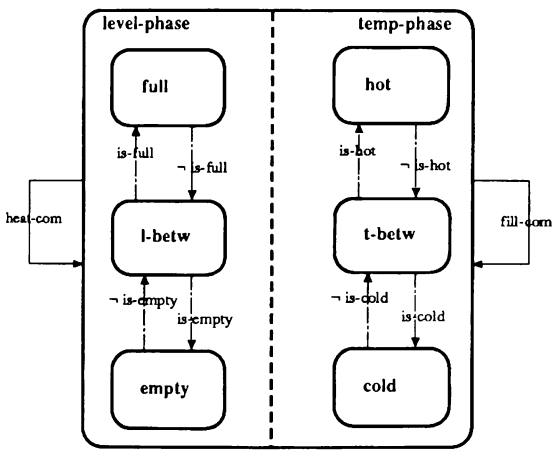

Figure 7: Higraph-Based DEVS-Diagram of Pot Model

busyLP, *busyHP*, and *preempted* when both queues are empty and in phases *busyHP* and *preempted* when the high priority queue gets empty.

To represent the Cartesian product, the statespace with several dimensions is combined in one big blob — called *and-blob* — which is then divided into several or-blobs separated by dashed lines. The or-blobs making up the and-blob represent the different dimensions of the Cartesian product. The and-blob now is not longer made up by the union of the atomic-blobs it contains but is made up by the product of the atomic-blobs of each dimension. That is, the atomic-blobs for each dimension exist in parallel. Figure 7 shows the higraph-based DEVS-diagram of the pot model. The phases of the two dimensions are represented by two orthogonal or-blobs denoted by the phase variables *temp-phase* and *level-phase*. The product of the 3 atomic-blobs for each dimension make up the 3 times 3 is 9 possible phase configurations seen in the original diagram of of figure 5. The state event transitions for these two dimensions can be represented independently. The external input events do not effect the phase transitions and therefore can be specified at the outmost blob. The derivatives for the continuous states applying for all phases expect the hot phase, should be specified for the outmost blob. Special derivatives then can be specified for the hot phase which, in our semantic of DEVS-diagrams, overwrites the general specification. All in all, this results in a remarkable reduction of the diagrams complexity. Also it should be recognized that the blob diagram is a very good and compact representation of the possible partitioning of the state space of the system with the atomic-blobs being the most granular units.

4 A CLIM IMPLEMENTED VISUAL MODELLING TOOL

The user interface toolkit *Common Lisp Interface Manager* (CLIM) (Lucid 1992) is employed to realize an interactive modeling interface for the STIMS modeling and simulation environment. A block diagram editor is implemented for coupled model specification. Coupled models are specified by drawing the component and coupling structure of a coupled model. For atomic model interactive specification, a user interface module is realized which is based on the graphical higraph-based DEVS-diagram representation presented above.

CLIM is a portable, powerful, high level user interface management system toolkit intended for Common Lisp / CLOS software developers. It acts as an abstract, high-level generic layer that provides a consistent interface across a large set of hosts and allows achieving the look and feel of the target host system without implementing it directly and without using the low-level implementation language of the host system.

CLIM is based on the object oriented concepts provided by CLOS. But in contrast to conventional object-oriented systems, CLIM also brings object oriented programming to the surface, to the user interface itself. A CLIM program is organized around three object types, viz. the *application objects* which are the internal objects building up the application, the *display objects* which serve as graphical, on-screen representations of the application objects, and the *presentations* which establish the link between appli-

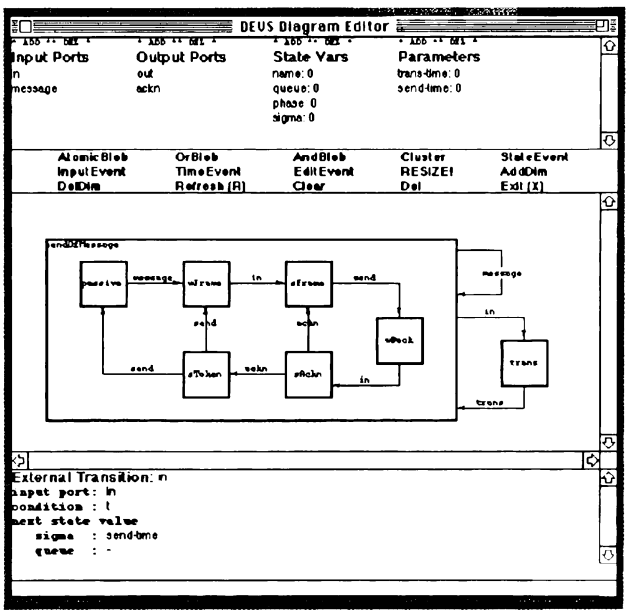

Figure 8: Token-Ring Protocol Model Specification using the DEVS-Diagram Specification Tool

cation and display objects. By that, CLIM provides a novel way to connect input and output to the semantics of an application.

In CLIM the application's user interface, i.e., the component that interacts directly with the user, is called the *application frame*. It usually is partitioned into several functional divisions, called *panes*, like drawing areas, menu-bars, or text editor windows. An application frame can have several layouts, i.e., arrangements of panes, which may be changed by the application. Figure 8 shows the application frame for the DEVS-diagram model specification tool. The first pane is employed to show and define the input, output, parameter, and state variables. Then, the second pane is used to interactively develop the DEVS-diagram of the model. In the last pane the state transition specification of event edge selected in the DEVS-diagram is presented in a formatted manner. The various fields giving conditions, next state and output values of the event specification can be edited using a simple, built in text editor. This combined graphical and textual specifications can be translated into a running atomic model ready to be simulated within the STIMS environment.

Special emphasis has been put on the program for the graphical, interactive development of DEVS-diagrams. The placing of the atomic-blobs and and-blobs is done by the user with the mouse. Or-blobs can be placed by the user if top down development of blob structures is desired. But, or-blobs also can be placed and layouted automatically when bottom-up modeling is required. The atomic-blobs which should be clustered have to be selected, a contour which encloses all these atomic-blobs but excludes all the other atomic-blobs is created automatically. To specify the edges, one has to give the starting point and the end point of the edge which has to be on a blob contour. The edge itself is laid automatically using a heuristic approach so that overlapping of edges is avoided.

5 SUMMARY AND OUTLOOK

We presented the DEVS-diagram method and tool for graphical representation of DEVS-based models which is based on state space phase partitioning and the higraph extension of conventional graph representations. As has been shown, DEVS-based models advantageously are depicted using a state transition diagram. Also, it has been shown that the higraph-based version of the DEVS-diagram representation leads to a remarkable reduction of the diagrams complexity. An CLIM implemented interactive tool for interactive DEVS-based atomic model specification has been presented. However, this first implementation is still a prototype needing further maturation. An extension of our method and tool planned for the future is to introduce inheritance of model behavior of DEVS-diagram specified models.

ACKNOWLEDGMENTS

This work has been supported by the Austrian Ministry of Sciences and Research under contract "Simulation of Intelligent Systems".

APPENDIX A: Preemptive Server Model

input ports:	inLP	(input of low priority jobs)
	inHP	(input of high priority jobs)
output ports:	out	
state variables:	phase	(phase of the system)
	LPQ	(low priority queue)
	HPQ	(high priority queue)

phase	**input event at port inLP**
idle	job into LPQ; hold-in busyLP serv-time
busyLP	job into LPQ; continue
busyHP	job into LPQ; continue
preempt	job into LPQ; continue
phase	**input event at port inHP**
idle	job into HPQ; hold-in busyHP serv-time
busyLP	job into HPQ; hold-in preempted serv-time
busyHP	job into HPQ; continue
preempt	job into HPQ; continue

phase	**time scheduled event**
idle	—
busyLP	put out first of LPQ
	if LPQ empty then passivate-in idle
	if ¬ LPQ empty then hold-in busyLP serv-time
busyHP	put out first of HPQ
	if ¬ HPQ empty then hold-in busyHP serv-time
	esleif ¬ LPQ empty then hold-in busyLP serv-time
	esleif LPQ empty then passivate-in idle
preempt	put out first of HPQ
	if ¬ HPQ empty then hold-in preempted serv-time
	elseif ¬ LPQ empty then re-schedule job in busyLP
	elseif empty then passivate-in idle

REFERENCES

Cota, B. A. and R. G. Sargent. 1989. Automatic Lookahead Computation for Conservative Distributed Simulation. Techn. Report No. 8916, Simulation Research Group, Syracuse University, Syracuse, New York.

Cota, B. A. and R. G. Sargent. 1992. A modification of the process interaction world view. *ACM Trans. on Modeling and Compuer Simulation* 2:109–129.

Fishwick, P. A. 1991. Heterogeneous decomposition and inter-level coupling for combined modeling. In *Proceedings of the 1991 Winter Simulation Conference*, 1120–1128, Phoenix, AZ.

Fishwick, P. A. and B. P. Zeigler. 1992. A multimodel methodology for qualitative model engineering. *ACM Trans. on Modeling and Compuer Simulation* 2:52–81.

Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming* 8:231–274.

Harel, D. 1988. On visual formalisms. *Comm. of the ACM* 31:514–530.

Harel, D. et al. 1990. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Trans. on Software Eng.* 16:403–414.

Lucid. 1992. *Common Lisp Interface Manager.* Technical Manual, Lucid Inc.

Oeren, T. I. 1991. Dynamic templates and semantic rules for simulation advisors and certifiers. in:*Knowledge Based Simulation: Methodology and Application*, ed. P. A. Fishwick and S. A. Modjeski, 53–76. New York: Springer.

Pichler, F. and H. Schwaertzel (eds.). 1992. *CAST Methods in Modelling.* Berlin: Springer.

Poole, T. G. and J. Z. Szymankiewicz. 1977. *Using Simulation to Solve Problems.* Maidenhead: McGrawHill.

Praehofer, H. 1991a. *System Theoretic Foundations for Combined Discrete- Continuous System Simulation.* PhD thesis, Johannes Kepler University, Linz, Austria.

Praehofer, H. 1991b. System theoretic formalisms for combined discrete-continuous system simulation. *Int. J. of General Systems* 19:219–240.

Praehofer, H., F. Auernig and G. Reisinger. 1993. An environment for DEVS-based multiformalism simulation in Common Lisp / CLOS . *Discrete Event Dynamic Systems: Theory and Applications* (to appear).

Praehofer, H., P. Bichler and B. Zeigler. 1993. Synthesis of endomorphic models for event-based intelligent control. *Proc. of the 4th Conference on AI, Simulation and Planning in High Autonomy Systems*, Tucson, AZ, IEEE/CS Press (to appear).

Pritsker, A. A. B. 1977. *Modeling and Analysis Using Q-GERT Networks.* New York: John Wiley.

Schriber, T. J. 1977. *Simulation Using GPSS.* New York: John Wiley.

Schruben, L. 1983, Simulation modeling with event graphs. *Comm. of the ACM* 26:957–963.

Steele, G. 1990. *Common Lisp: The Language.* Burlington: Digital Press.

Zeigler, B. P. 1976. *Theory of Modelling and Simulation.* New York: John Wiley.

Zeigler, B. P. 1984. *Multifacetted Modelling and Discrete Event Simulation.* New York: Academic Press.

Zeigler, B. P. 1990. *Object Oriented Simulation with Modular Hierarchical Models.* New York: Academic Press.

AUTHOR BIOGRAPHIES

HERBERT PRAEHOFER is a faculty member of the Department of Systems Theory and Informations Engineering at the Johannes Kepler University of Linz. He got his M.S. and Ph.D. degrees in computer science from the University of Linz in 1986 and 1991, respectively. His research interests include discrete event and combuined simulation methodology, system design, object-oriented techniques, model-based reasoning, and knowledge-based techniques.

DIETMAR PREE is a graduate student in computer science at the Johannes Kepler University of Linz. Currently he is working towards his M.S. degree. His research interests include object-oriented programming, interactive user interfaces, real-time system design, and visual specification techniques.