1998

Visual Interactive Modeling in a Java-based Hierarchical Modeling and Simulation System

Robert G. Sargent
Syracuse University

Thorsten Daum
Syracuse University

Visual Interactive Modeling in a Java-based Hierarchical Modeling and Simulation System

Robert G. Sargent
Thorsten Daum*
Syracuse University†

Abstract

The desired set of properties of a modern simulation system is presented. The portion of the Hierarchical Modeling And Simulation System-Java (HiMASS-j) used for specifying Hierarchical Control Flow Graph (HCFG) Models is described. The specification of HCFG Models in HiMASS-j is by visual interactive modeling through the use of graphical user interfaces and dialog boxes. HCFG Models are specified using two complementary hierarchical specification structures: one to specify the components that comprise a model and how these components are interconnected, and the other to specify the behaviors of individual atomic components. A simulation of a traffic intersection using HiMASS-j is briefly discussed.

1 Introduction

This paper has two purposes. First, in Section 2, we present and briefly discuss a desired set of properties that we believe a modern simulation system should have. Second, we describe the portion of the Hierarchical Modeling and Simulation System-Java (HiMASS-j) used for specifying Hierarchical Control Flow Graph (HCFG) Models via visual interactive modeling (VIM). HiMASS-j is currently a prototype Java-based discrete event simulation system for specifying and executing HCFG Models.

In order to understand the specification of HCFG Models in HiMASS-j, a minimal understanding of the HCFG Model paradigm is required. Thus, we give in Section 3 a brief overview of the HCFG Model paradigm. We note, however, a deeper understanding of this model paradigm will provide more and better insights into HiMASS-j. A short description of the HCFG Model paradigm can be found in "An Overview of Hierarchical Control Flow Graph Models" (Fritz and Sargent 1995) and a more detailed description can be obtain in an expanded version of this overview paper located at `http://www.cat.syr.edu/~srg/`.

Specifying HCFG Models in HiMASS-j is described in Section 4. Section 5 contains a brief description of a simulation of a traffic intersection in Magdeburg, Germany using HiMASS-j. Section 6 contains a short discussion of the HiMASS-j software and Section 7 is the summary.

* Also a student at the Otto-von-Guericke Universität Magdeburg.
† 439 Link Hall, Simulation Research Group, Syracuse University, Syracuse, NY 13244.

2 Properties of a Modern Simulation System

In this section we first present and then briefly discuss the desired set of properties that we believe a modern simulation system should have. These properties are:

1. visual interactive modeling capability
2. layered modeling approach
3. model element reuse
4. library for model elements
5. experimental frame
6. simulation engine that can efficiently execute a simulation model on different types of computer architectures/computers (including the web)
7. visual interactive experimental design and (input and output) data analysis capability
8. animation capability
9. be object oriented and
10. be designed and implemented using modern software approaches and techniques

We use the term 'visual interactive' in a broad sense. Visual interactive (VI) implies the capability of a user to interact with objects on the computer screen via a mouse or a keyboard. An object can be a string (e.g., a name), an icon, a tool button, etc. VI usually occurs with graphical user interfaces (GUIs) and dialog boxes. VI is not, e.g., writing computer programs in text via a keyboard.

A modern simulation system should use a powerful modeling paradigm. This paradigm should have (i) a graphical representation that can be used for VI modeling, (ii) hierarchical modeling capability, (iii) a powerful behavior specification capability (such as having a 'wait until' construct), (iv) scaling capability for model elements, and (v) the capability for building models top down or bottom up. (See Sargent (1992) for additional requirements of a modeling paradigm.)

Model specification should be VI wherever appropriate. A layered approach to modeling should be used. This means that different levels of model abstraction and different levels of model elements can be used for model specification. This approach should allow (i) a modeler to specify a model and its model elements at the most basic level such as being able to write an event routine, (ii) the movement between levels to be VI, and (iii) a modeler to determine what a model element is doing (which usually means moving to a lower level) and this should be visual (as contrasted, e.g., to computer code) where appropriate. A library with VI capability should be provided for the storage and reuse of model elements. Model elements should allow the use of parameters to increase element reuse. The experimental frame concept should be used in order to specify the desired values of a model for each simulation run. This specification should be VI.

A modern simulation system should be able to execute a specified simulation model on different types of computer architectures and systems without requiring a modeler to add additional information to a model. This requires a modeling paradigm to have a representation such that algorithms can be developed to obtain the information needed for model execution algorithms automatically from the representation of a specified model.

The simulation engine should have model execution algorithms for sequential, parallel, and distributed computers. Furthermore, in today's environment a simulation system should provide for web use.

An extensive capability should be provided to perform simulation input and output analyses. Such capability should be VI wherever appropriate and include the ability to specify (i) data collection, (ii) tactical and strategic experimental designs, (iii) how the input and output data analyses are to be conducted, and (iv) how the results are to be presented. Exploratory data analysis capability should also be included. (See Robinson (1997) for a limited discussion of visualization and simulation data analysis.) A straight forward interface should be provided to allow the use of external spread sheets. Animation capability should be provided either internally or with a straight forward interface to an external animation system.

A modern simulation system should be object oriented. This provides for abstraction, encapsulation, types, and instances. This should increase the reuse of model elements. Design and implemention of the software system should follow object oriented methodology. The graphical representations used by VIM and the logic of the model should be separated by using a suited design approach such as the Model-View-Controller paradigm (Gamma et al. 1995).

3 An Overview of HCFG Models

An HCFG Model can conceptually be viewed as consisting of a set of independent and encapsulated components which interact with each other solely via message passing. Two kinds of encapsulated components are used: an atomic component (AC), which has its own thread of control and operates concurrently with other ACs, and a coupled component (CC), which couples together ACs and/or other CCs. The CCs provide hierarchical relationships among components. Each CC is specified by the use of a CC Specification (CCS). A CCS is a directed graph whose nodes represent components and whose directed edges represent channels between component ports. Channels carry messages between the input and output ports of components. Each channel connects exactly one output port to one input port and each port is connected to only one channel. The hierarchical relationships of all components and their interconnections are contained in the Hierarchical Interconnection Graph (HIG). A HIG tree shows the hierarchical relationships among the components but not their interconnections.

Each AC contains a set of (local) variables including a (local) simulation clock, a set of input ports, a set of output ports, a set of parameters, and an HCFG, which describes the behavior of that AC. An HCFG is a hierarchically organized set of Macro Control States (MCSs) and their interconnections. A MCS is a state-based behavior specification structure and is an augmented directed graph whose nodes are control states (CSs) and/or other MCSs and whose directed edges give the set of possible control state transitions. A CS is a formalization of the "process reactivation point" (Cota and Sargent 1992). Edges originating from CSs have three attributes: a condition, a priority, and an event. The condition specifies when an edge can become a candidate for traversal, the priority is used to break

ties when more then one edge is a candidate for traversal at the same simulation time, and the event specifies a state transition for the AC which is executed whenever that edge is traversed during simulation execution. Three different kinds of edges, which depend on the condition attribute, are used: TimeEdges, BoolEdges, and PortEdges. (These can be viewed as different types of 'wait until.') To specify an edge, a priority and event is specified along with an edge type and either a time delay function, a boolean function, or an input port, depending on the type of edge. Edges originating from MCSs do not have attributes. Each HCFG has a point of control (POC), which moves from CS to CS to indicate the state that the HCFG (i.e., the AC) is in. The POC leaves a CS over the edge that becomes `true` first. An HCFG tree shows the hierarchical relationships among the MCSs but not their interconnections.

The HCFG Model paradigm supports and HiMASS-j implements the use of types and instances of model elements. The model elements are the ACs, CCs, MCSs, event routines, time delay functions, and boolean functions. The types of model elements are specified and instances of the types are used to specify HCFG Models. Libraries of types can be established and this provides for reuse. Having libraries of types allows the "layered" approach to modeling to be used where ACs, CCs, and MCSs can be used for model specification if the appropriate ones are available, and if not, then the needed elements can be build.

The specification of an HCFG Model requires one HIG and one HCFG for each type of AC in the model. HCFG Model specification in HiMASS-j is via VIM through the use of graphical user interfaces (GUIs) which includes the use of dialog boxes. The HCFG model paradigm supports and HiMASS-j implements the use of experimental frames (EFs) (Zeigler 1984). The use of EFs allows the values of the parameters of the model elements, the model's initial conditions, etc. to be specified separately from the HCFG Model specification.

The HCFG model paradigm favors an "active resource" view of modeling over an "active transaction" view. Modeling from an active resource view means that the system is modeled from the point of view of the system's resources by describing the behaviors and interactions of those resources. We use the active resource view for specifying models in this paper.

4 Specifying HCFG Models in HiMASS-j

In this section we discuss how HiMASS-j can be used to specify HCFG Models. HCFG Models are specified in HiMASS-j via VIM using GUIs. Our emphasis will be on specifying HCFG Models top down and from scratch, i.e., not using libraries of model elements. A simple way to specify and view HCFG Models is to use the Model Navigator window. The Model Navigator contains the HCFG Model tree and is used to navigate, i.e. to move, among the model's CCs, ACs, and MCSs. The model tree consists of the HIG tree and the HCFG tree of each AC in the model. (The model tree does not show the top MCS of an HCFG since it is the internal view of an AC.) The Model Navigator containing a sample model tree is shown in Figure 1. Each node of the model tree has a symbol to indicate the kind of model element, the instance name of the model element, and the type name of

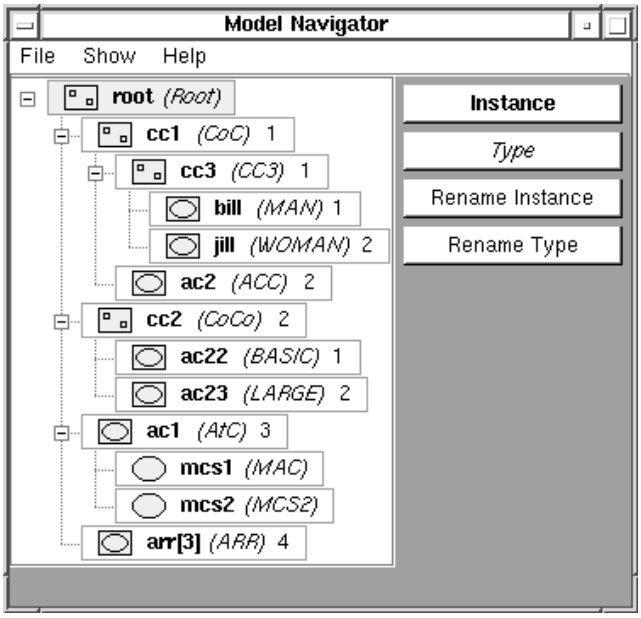

Figure 1: Model Navigator Window

the model element given in parentheses. Each component node also has a priority which is used to established priorities among ACs to break event time ties that may occur across ACs. The top node (named root unless renamed) of the model tree is always displayed and this is the HIG root (top) node.

The Model Navigator has four tool buttons on its right: Instance opens a GUI window containing the instance of the selected model element in the model tree, Type opens a GUI window containing the type of the selected model element, and the other two buttons open dialog boxes to rename either a selected model element instance or type. HiMASS-j has a CC GUI window for working with the CCs of a HIG and a MCS GUI window for working with the MCSs of the HCFGs. A CC GUI window opens if an instance of a component (AC or CC) or if a type of CC is selected, and a MCS GUI window opens if a MCS or a type of AC is selected.

4.1 HIG Specification

The first step in specifying an HCFG Model from top down is to open the Model Navigator. The model tree in the Model Navigator will contain one node which is a CC with the instance name ‘root’ of type ‘Root’. A modeler opens the CC GUI window for Root by selecting the Type button while the root node is selected (highlighted). Figure 2 shows the CC GUI window with a sample Root CC in it. (The canvas area would be blank when Root is initially opened.) Note that the type name of the CC is given at the top of the window

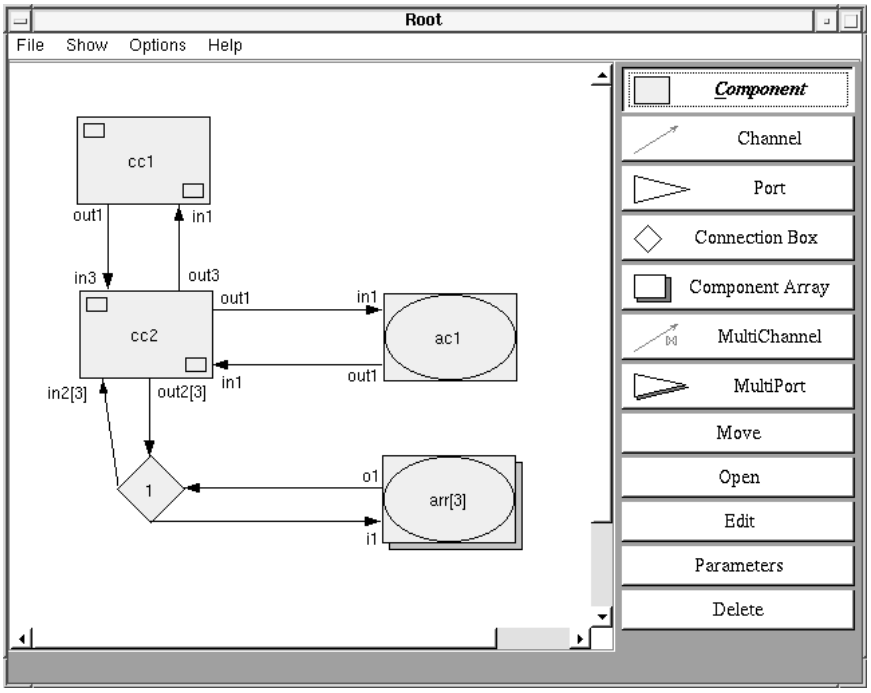

Figure 2: CC Graphical User Interface Window

and that there are several tool buttons on the right side of this GUI. These tools are used to either specify a CC (which contains ACs and/or other CCs), modify a CC, or change the layout of the CC on the canvas. A user selects a tool by clicking on the appropriate button, and then uses the tool in the canvas area. To create a new component, a modeler first selects the Component tool and then clicks on the canvas at the desired location for the new component. A Component dialog box will open for the modeler to select the kind of component and to enter the component type name, the component instance name, and for the component priority either a numerical value or a variable whose value is specified in the EF. (A Component dialog box is similar to the Array dialog box shown in Figure 3.) In Figure 2 there are two CCs called cc1 and cc2 and an AC called ac1. Note the symbols used for these two different kinds of components and that instance names are given on the components. Type names and priorities can also be shown, optionally. As new components are specified in a CC, they are automatically added to the model tree in the Model Navigator. Note how the components contained in Root CC are in the model tree in Figure 1.

In Figure 2 there are, e.g., two channels between ac1 and cc2. Channels are specified by using the Channel tool. A new channel between components is specified by first selecting the Channel tool and then clicking inside the component where the channel originates. A Port dialog box will open to enter the name of that component's new output port for connection to the new channel. Next, click inside the component where this new channel

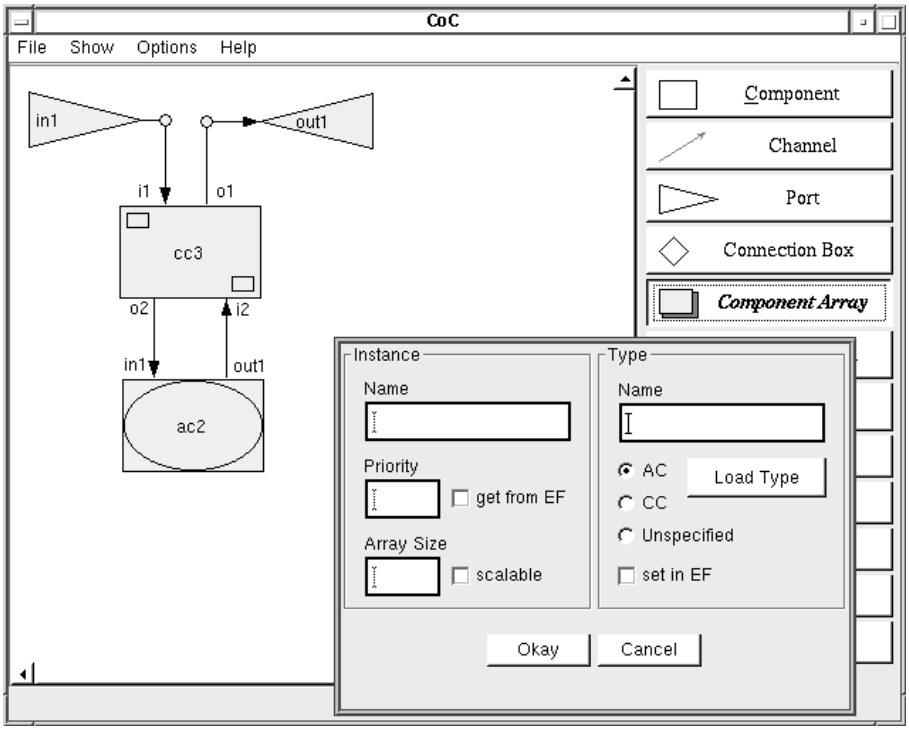

Figure 3: Internal View of a CC with Array Dialog Box

terminates. A Port dialog box will open to enter the name of that component's new input port for connection to the new channel. The new channel and the names of the new ports automatically appear.

HiMASS-j provides scaling for components, channels, and ports. A component array consists of a homogeneous array of components. (A component is equivalent to a component array of size one.) A typical element of the array is specified and shown. After the Component Array tool is selected and the mouse clicked at the desired location for the component array on the canvas, the Array dialog box shown in Figure 3 opens. A modeler selects the kind of component that the array has and enters the array's instance name, type name, either a numerical value or variable (if the EF is being used) for the array's priority, and either a numerical value or a variable whose value is specified in the EF for the array size in the dialog box. In Figure 2 there is an array containing 3 ACs called arr. Note that the array symbol indicates whether the components are ACs or CCs and that the size of the array is given with the instance name. As new arrays are specified, they are automatically added to the model tree. Multichannels are arrays (or bundles) of channels that have their size specified by a modeler and are created similarly to channels. The major differences are that multichannels are connected to multiports and that their size must be specified in the MultiPort dialog box.

Because model elements in a HIG can be of different sizes, not all channels between model elements can be clearly represented using a purely graphical notation. A "connection box" that is represented by a diamond can be used to connect channels and multichannels of different sizes to model elements. A Connection Box tool is provided to specify connection boxes on the canvas similar to how components are specified. A Connection Box window (obtained by using the Edit button) is used to make the connections of the channels entering and leaving a connection box. The ports that the channels entering and leaving the connection box are connected to are automatically given by HiMASS-j, and it is straightforward and simple to make the appropriate connections through visual interaction. In Figure 2 note the connection box named 1, the multichannels of size 3 between the multiports of CC cc2 and the connection box, and the channels between the (three) elements of the array arr and the connection box.

The Open tool opens up a GUI window of the component's type to either view or specify that component. If the component is a CC, then a CC window is opened; otherwise a MCS window is opened. If a CC window is opened for a CC other then the top CC, then the ports for that CC are shown. (By definition the top CC has no ports.) For example, the CC GUI window for CC CoC (which is the type for the CC instance cc1 contained within Root) is shown in Figure 3. If no components have yet been specified for CoC, then only the input port in1 and output port out1 would be shown. In top down modeling these ports (with their names) would have been specified in an instance of the CC in the CC window; e.g., the ports of CoC were specified in the instance cc1 in the CC Root. Note that the components in CoC are in the model tree in Figure 1.

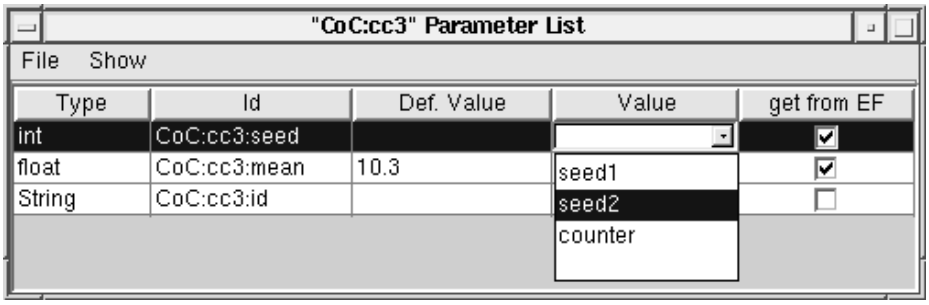

Figure 4: Parameter Window

The purpose of the Port and MultiPort tools is to specify ports and multiports when specifying model elements bottom up instead of top down. The Move, Edit, and Delete tools provide the common editing capabilities. The Parameter tool allows for the specification of parameter values for ACs and CCs. Clicking inside the desired component opens the Parameter window which is shown in Figure 4. A modeler can specify an absolute value (such as a number or string) for a parameter, assign the value of an accessible variable, and/or mark the *get from EF* box to indicate that this parameter can be overridden in the EF (see Figure 8).

4.2 HCFG Specification

The MCSs in HCFGs are specified in HiMASS-j by using a MCS GUI window. To specify an HCFG top down, the modeler first opens a MCS GUI window for the top MCS of that AC. This is accomplished by either (i) using the Model Navigator to select the appropriate AC and then clicking on the Type button, or (ii) using the Open tool on the appropriate AC in a CC GUI window. Figure 5 shows the MCS GUI window containing the top MCS of AC AtC.

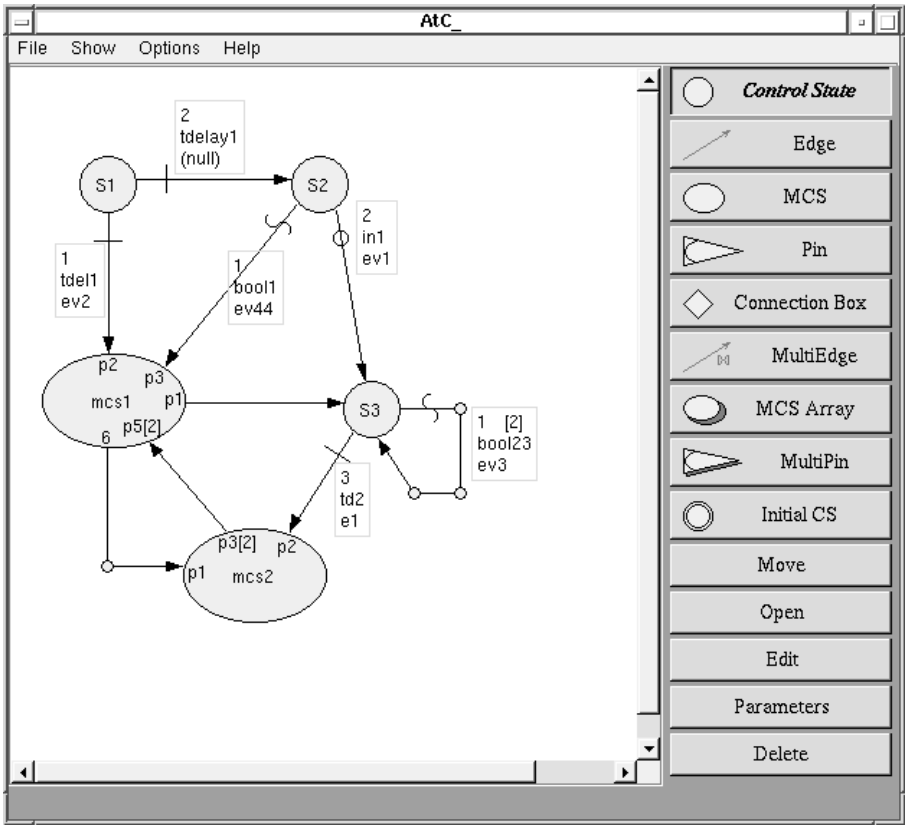

Figure 5: MCS Graphical User Interface Window

One can readily see that there are a set of tool buttons on the right side of the MCS window. The Control State and MCS tools are used to specify Control States (CSs) and MCSs. In Figure 5 there are 3 CSs (S1, S2, and S3) and two MCSs (mcs1 and mcs2). These tools operate similar to the Component tool in the CC window. New MCSs are automatically put into the model tree as they are specified. Note the MCSs mcs1 and mcs2 in the model tree in Figure 1. (The MSC AtC_ is not shown in Figure 1 since this is the top MCS of the AC AtC and thus the internal view of this AC.)

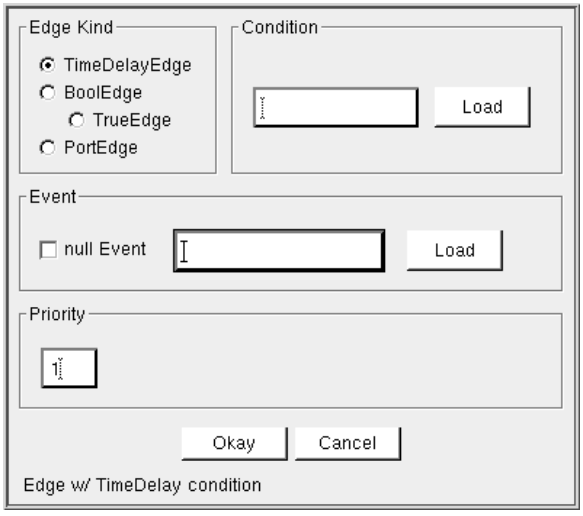

Figure 6: Edge Dialog Box

The Edge tool operates similar to the Channel tool. If the (new) edge being specified starts at a MCS, a Pin dialog box opens when the MCS is clicked asking for the name of the (new) pin that the (new) edge will originate at. A pin connects edges at a MCS boundary in the same way a port connects channels at a CC boundary. If this edge terminates on a CS, then the edge appears when that CS is clicked, and if the edge terminates on a MCS, then a Pin dialog box opens when that MCS is clicked to specify the pin name prior to the edge appearing. (See the edges leaving MCS mcs1 and the names of the pins of MCS mcs1 in Figure 5.) If the edge specified starts at a CS, then an Edge dialog box opens. The Edge dialog box, shown in Figure 6, provides for several entries and selections. A modeler first selects the edge condition for the type of edge being specified. If a TimeEdge or a BoolEdge is selected, then the name of the time delay function or boolean function is entered into the Condition portion of the dialog box. If a PortEdge is selected, then the associated port is selected from the list of input ports given in the Condition portion of the dialog box (or added if building from bottom up). The name of the event routine is specified in the Event portion of the dialog box and the edge priority is specified in the Priority portion of the dialog box. (A TrueEdge is a BoolEdge that is always true and a null Event is an event that does nothing and these can be specified by clicking on the appropriate buttons.) See the edges leaving the CSs in Figure 5. Note that the edge attributes are given in an attribute box located near each edge. The top entry in each edge attribute box gives the edge priority, the second entry gives either the time delay function, the boolean function, or the input port name depending on the kind of edge, and the last entry gives the name of the event routine. In Figure 5, the two edges leaving CS S1 are TimeEdges, the edge going from CS S2 to MCS mcs1 is a BoolEdge, and the edge going from CS S2 to CS S3 is a PortEdge. Note the symbols used to denote the diffenent edge types.

Figure 7: Event Editor window

To specify a time delay function, a boolean function, or an event routine, one first selects the Edit tool and then selects the desired entry in an edge attribute box. An editor window for that entity will then open. See Figure 7 for the Event Editor window, which contains a sample event for event ev1 of MCS AtC_ (Figure 5). The event ev1, which is for a PortEdge, receives the message waiting at input port in1, stores the message timestamp in the trace file, and sends the message to output port out1. Event routines are specified using Java syntax. Helper functions are provided to assist in the specification of events and these are the buttons on the right side of the dialog box. The Time Delay Function and the Boolean Function Editors are similar to the Event Editor. Time delay and boolean functions are also specified using Java syntax.

HiMASS-j provides scaling for edges, MCSs, and Pins. A multiedge is an indexed array of edges. There are two multiedges in Figure 5. One connects between multipins on MCS mcs2 and MCS mcs1, and is size 2. The other multiedge is from CS S3 back to itself, and is also size 2. Note that the size of multiedges leaving CSs are given in square brackets in the attribute box. (The edge attributes of multiedges leaving CSs are also indexed.) A MCS array is a homogeneous array of MSCs similar to a component array. Connection boxes are also available for connecting multiedges.

The purpose of the Pin and MultiPin tools is to specify pins and multipins when specifying MCSs bottom up instead of top down. The Initial CS tool allows a modeler to specify the initial control state of an HCFG. The Move, Edit, and Delete tools provide the common editing capabilities. The Open tool opens up a GUI window of a MCS. The Parameter tool allows for the specification of parameter values for MCSs, time delay functions, boolean functions, and event routines. It operates similar to the Parameter tool in the CC GUI window.

4.3 Experimental Frame

HiMASS-j provides EF support for specifying or overriding the default settings of model element attributes. These attributes include (i) the initial CSs of ACs, (ii) the values of instance parameters and type variables, (iii) the priorities of components, (iv) the sizes of model element arrays that use variable scaling, and (v) the types of model elements at runtime, which allows the model structure to be changed. These specifications are aided by dialog boxes and require no programming.

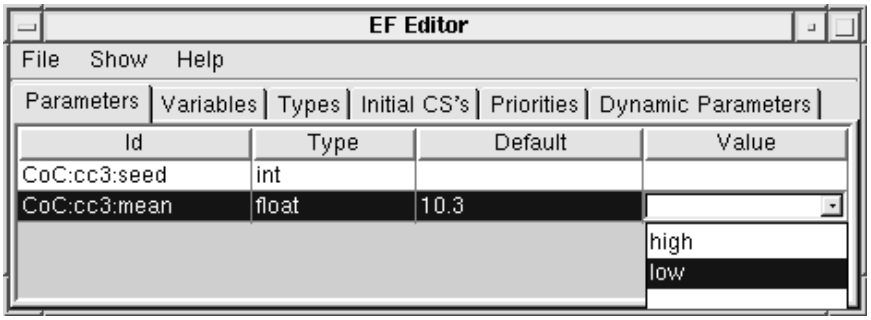

Figure 8: EF Window Showing Parameters Being Specified

When specifying model element attributes, a modeler can check the *get from EF* box in the dialog box being used. This will generate an entry in the EF file of the model. An EF entry includes the unique identifier of the attribute, its type, and may include a default value.

An EF file can be manipulated in the EF window, which can be opened either from within HiMASS-j or independent of HiMASS-j. In the EF window a modeler can specify a value for an attribute by (i) typing an absolute value (such as a number or a string) in the *Value* field of the corresponding EF entry or (ii) by assigning the value of a variable of the same type. Figure 8 shows the Parameter section of the EF window that contains entries for the two parameters *seed* and *mean* of the CC instance cc3 that is contained within type CoC. The entry for *mean* is highlighted and the list of variables that have the same type as *mean* is displayed. The modeler can change the default value by entering a number in the *Value* field or by selecting one of the variables *high* or *low*. A modeler must specify a value for each entry in the experimental frame if no default value has been specified. This is the case for the *seed* parameter in Figure 8.

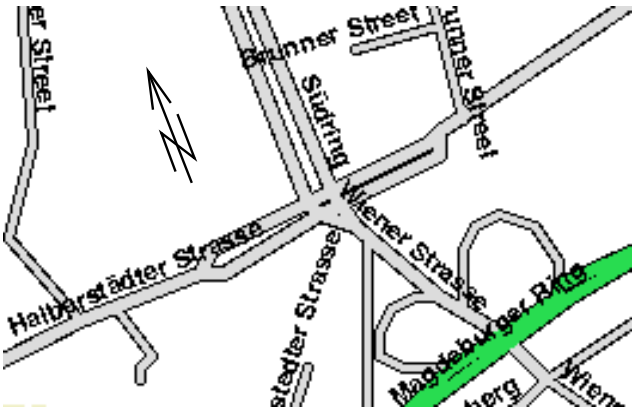

Figure 9: The Intersection of Südring and Halberstädter Straße

5 Traffic Intersection Example

HiMASS-j was used to create a simulation model of the intersection of Halberstädter Straße und Südring, two major streets in the city of Magdeburg, Germany (shown in Figure 9). The streets have four to six vehicle lanes and streetcar tracks. Traffic consists of vehicles, streetcars, and pedestrians, and is regulated by traffic lights. The purpose of this model is to study the relationships between traffic light timing patterns and the waiting times of vehicles in the intersection. (A more comprehensive description of this model can be found in Daum 1997).

The HIG of the intersection model has five levels. It contains over 400 AC instances and over 60 CC instances that were specified using 14 AC types and 20 CC types. The *complexity* of the intersection is modeled in the HIG; the ACs have simple HCFGs. Figure 10 shows part of the model tree in the Model Navigator window.

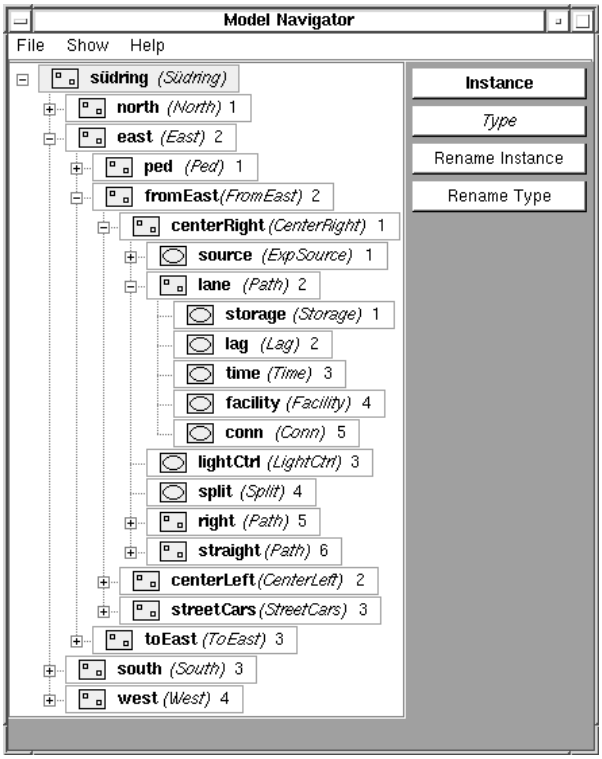

Figure 10: The Model Tree

The intersection model was specified from top down. The top level CC, which is called Südring and is shown in Figure 11, consists of four CCs: north, south, east, and west. These CCs are connected by several multichannels to handle the interactions between them. Each of these CCs contains three CCs: one CC for pedestrian traffic and two CCs for inbound

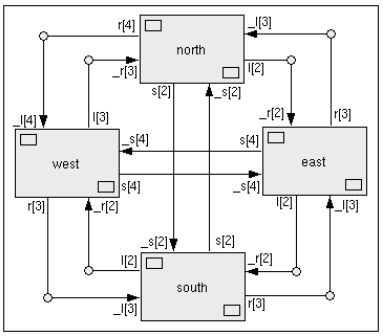

Figure 11: The Top Level CC

and outbound traffic, respectively. (See Figure 10.) An inbound CC, e.g., contains a CC for each vehicle lane, one CC for streetcars, and one AC for a traffic light. An inbound vehicle lane CC contains at least the following: a Source AC which generates messages representing vehicles, a LightControl AC which can block vehicles based upon the state of the corresponding traffic light, and a Path CC. A Path CC is used to specify sections of the intersection where traffic can flow unrestricted: a continuous stretch of road, streetcar track, or sidewalk. A Path CC (shown in Figure 12) contains five ACs that model basic properties of a path such as its capacity and the minimal time an entity (a vehicle, streetcar, or pedestrian) needs to travel the entire length of the path.

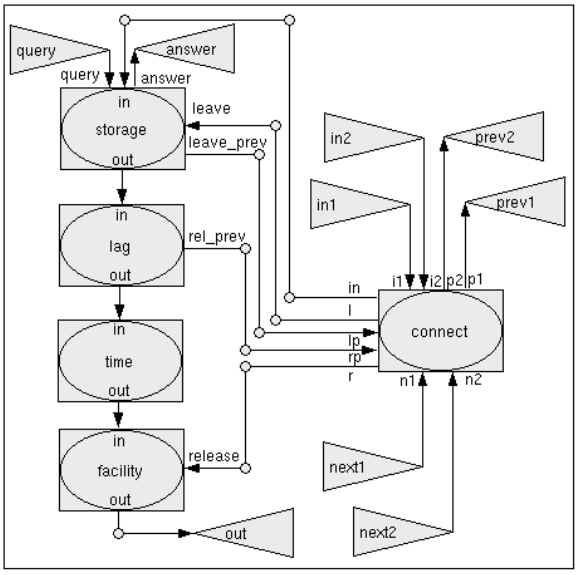

Figure 12: The Path CC

The ACs employed in the model have simple HCFGs. For example, the TrafficLight AC shown in Figure 13 has three CSs (R(ed), Y(ellow), and G(reen)) and three TimeEdges, which connect R to G, G to Y, and Y to R. The time delay functions red, yellow, and green return the times the light is in a particular state. The events redToGreen and yellowToRed send messages that cause the connected LightCtrl AC to change its state accordingly.

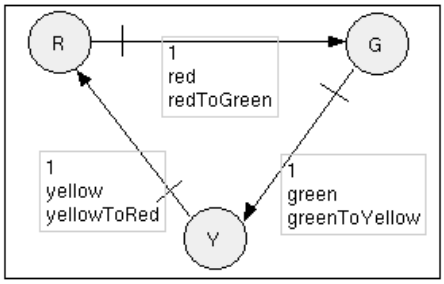

Figure 13: The TrafficLight HCFG

The intersection model makes extensive use model element parameters and the EF. For example, the means of the interarrival times for instances of the Source ACs and the values to be returned by the time delay functions of instances of the TrafficLight AC are specified in the EF. This makes changing the average arrival rate of entities to the intersection (e.g., for a different time of day) and the traffic light cycle easy.

The simulation model produces a trace file for each simulation run that is used for an animation of the model using Proof Animation (Wolverine Software 1995).

6 HiMASS-j Software

HiMASS-j is an object oriented simulation system written entirely in Java (Arnold and Gosling 1996). It includes over 800 kilobytes of sources in over 160 Java classes in three packages. HiMASS-j has been developed and tested on both SUN workstations running Solaris and Pentium based personal computers running Linux and Windows 95. Due to the platform independent nature of Java, HiMASS-j can run without the need to recompile on a wide variety of machines, including several Unix architectures and Apple Macintosh computers. HiMASS-j makes extensive use of object-oriented programming features supported by Java such as encapsulation, inheritance, polymorphism, and overloading, and specific Java features such as serialization and platform independence.

A compiled HiMASS-j model is independent of the VIM part of HiMASS and can be executed by any software that implements a simulator for HCFG Models and that accepts time delay and boolean functions and event routines specified in Java syntax.

HiMASS-j was implemented using the SUN Java Development Kit (JDK) version 1.1.x, available from `http://java.sun.com`, and the Java Generic Library (JGL). JGL is currently free and available at `http://www.objectspace.com/jgl`.

A user of HiMASS-j needs to be familiar with modeling using the HCFG Model paradigm and a basic understanding of the Java syntax. HiMASS-j can be downloaded from `http://www.cat.syr.edu/~srg/` and run with any JDK 1.1.5 (or newer) compatible Java development system that includes a Java compiler, such as the SUN JDK.

7 Summary

The set of properties that we believe a modern simulation system should have was given. A subset of these were illustrated in our description of the portion of HiMASS-j that is used for specifying HCFG Models via VIM. A number of features of HiMASS-j were not discussed and many of these would also illustrate some of the properties that we believe are desirable in a modern simulation system. Some of the features of HiMASS-j not discussed are specifying messages, model initial conditions, data collection, and HCFG Models bottom up (instead of top down); specifics on specifying event routines, time delay and boolean functions, and variables and their values; and specifics on the use of scaling, EF, and model element reuse. (See Daum and Sargent (1997) for additional discussion of HiMASS-j.)

A brief description of a traffic intersection simulation using HiMASS-j was presented. The hierarchical nature of the HCFG Model paradigm allowed for the representation of a complex system in a way that was intuitive and comprehensible. The capabilities for reuse in HiMASS-j simplified the building of the model. For example, only 14 AC types had to be specified in order to use over 400 AC instances. The ability to parameterize model element instances was crucial for effective reuse. The visual modeling aspect of HiMASS-j provided a natural way of modeling, thus a verified model was achieved faster than with text based tools for model specification. VIM worked extremely well for the animation of this simulation model because the visual structure of the HIG could be specified to resemble the animation layout. Further details of the intersection model can be found in Daum (1997). (Some other papers on modeling using the HCFG Model paradigm are Farr et al. (1995) and Sargent (1997).)

A few comments were made on the HiMASS-j software. (See Daum and Sargent (1997) for additional information.) HiMASS-j was designed and implemented using modern software approaches and techniques. HiMASS-j differs significantly from HI-MASS (Fritz, Sargent, and Daum 1995), an earlier prototype for specifying and simulating HCFG Models that is C++ based. Some of the differences are that in HiMASS-j both the HIG and the HCFGs are specified via VIM, there is a clear distinction between types and instances of model elements, the model tree is displayed, there is a model navigator, the use of parameters is permitted in all of the model elements, reuse of model elements is easy, the EF has extensive capability and is VI, and the Java language is used which allows HiMASS-j to be used on the web.

Much has been said about the substantial speed sacrifice that one sometimes has to accept when executing programs in Java instead of C++. However, Java's disadvantage in this regard has already begun to diminish. Next generation just-in-time compilers significantly increase the speed of Java applications. In addition, efforts are under way to build a Java front end to the GNU compiler (Bothner 1997) that would allow ahead-of-time compilation of Java applications into native code providing for performance equal to C++ applications.

References

Arnold, K. and J. Gosling. 1996. *The Java Programming Language*. Reading, Mass.: Addison Wesley.

Bothner, P. 1997. A Gcc-based Java Implementation. In: *Proc. of IEEE Compcon '97*.

Cota, B. and R. Sargent. 1992. A Modification of the Process Interaction World View. *ACM Trans. Model. Comput. Simul.*, 2, 2, 109–129.

Daum, T. 1997. An HCFG Model of a Traffic Intersection Specified Using HiMASS-j. In: S. Andratdottir, K.J. Healy, D.H. Withers, and B.L. Nelson, eds., *Proc. of the 1997 Winter Simulation Conference*, 158–165.

Daum, T. and R. Sargent. 1997. A Java Based System for Specifying Hierarchical Control Flow Graph Models. In: S. Andratdottir, K.J. Healy, D.H. Withers, and B.L. Nelson, eds., *Proc. of the 1997 Winter Simulation Conference*, 150–157.

Farr, S., A. Sisti, D. Fritz, and R. Sargent. 1995. A Simulation Model of a Surveillance Radar Data Processing System Using HI-MASS. In: C. Alexopoulos, K. Kang, W. Lilegdon, and D. Goldsman, eds., *Proc. of the 1995 Winter Simulation Conference*, 1364–1370.

Fritz, D. and R. Sargent. 1995. An Overview of Hierarchical Control Flow Graph Models. In: C. Alexopoulos, K. Kang, W. Lilegdon, and D. Goldsman, eds., *Proc. of the 1995 Winter Simulation Conference*, 1347–1355.

Fritz, D., R. Sargent, and T. Daum. 1995. An Overview of HI-MASS (Hierarchical Modeling and Simulation System). In: C. Alexopoulos, K. Kang, W. Lilegdon, and D. Goldsman, eds., *Proc. of the 1995 Winter Simulation Conference*, 1356–1363.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. *Design patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass.: Addison-Wesley.

Robinson, J. 1997. Visualize a Port in Africa. In: S. Andratdottir, K.J. Healy, D.H. Withers, and B.L. Nelson, eds., *Proc. of the 1997 Winter Simulation Conference*, 70–77.

Sargent, R. 1992. Requirements of a Modelling Paradigm. In: J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, eds., *Proc. of the 1992 Winter Simulation Conference*, 780–782.

Sargent, R. 1997. Modeling Queueing Systems Using Hierarchical Control Flow Graph Models. *Mathematics and Computers in Simulation*, 44, 3, 233–249.

Wolverine Software. 1995. *Using Proof Animation (Second Edition)*. Annandale, Va.: Wolverine Software Corporation.

Zeigler, B. 1984. *Multifacetted Modelling and Discrete Event Simulation*. London: Academic Press.