Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

AN OVERVIEW OF HIERARCHICAL CONTROL FLOW GRAPH MODELS

Douglas G. Fritz
Robert G. Sargent

Simulation Research Group
Syracuse University
439 Link Hall
Syracuse, New York 13244, U.S.A.

ABSTRACT

Hierarchical Control Flow Graph Models define a modeling paradigm for discrete event simulation modeling based upon hierarchical extensions to Control Flow Graph Models. Conceptually, models consist of a set of independent, encapsulated, concurrently operating model components where each component has its own thread of control and the components interact with each other solely via message passing. Two primary objectives for Hierarchical Control Flow Graph Models are: (1) to facilitate model development by making it easier to develop, maintain, and reuse models and model elements and (2) to support the flexible and efficient execution of models. Hierarchical Control Flow Graph Models use two complementary types of hierarchical model specification structures, one to specify components and interconnections and the other to specify component behaviors.

1 INTRODUCTION

Hierarchical Control Flow Graph (HCFG) Models are a hierarchical modeling paradigm for discrete event simulation that are based on and designed to be a hierarchical extension of Control Flow Graph Models.

1.1 Foundation

Cota and Sargent (1990a) developed the Control Flow Graph (CFG) Model representation based on the modified process interaction world view (Cota and Sargent 1992). The primary objective of CFG Models was making information useful for parallel simulation explicit in the model representation. Conceptually, a CFG model is a set of independent, encapsulated, concurrently operating model components where each component has its own thread of control and the components interact with each other solely via message passing.

In the CFG Model representation, the behavior of each model component (or process) is specified by a Control Flow Graph. The interactions between components are accomplished via message passing over a set of directed channels which interconnect model components and specify the static routing of all intercomponent message traffic. Each channel generally carries only one type of message which implies that there may be multiple channels between two components, each of which carries a different type of message. Messages leave a component via an output port and enter a component via an input port. Each channel connects exactly one output port to one input port and each port is connected to exactly one channel. Messages queue on input ports until the Control Flow Graph describing the behavior of the receiving component decides to receive them; i.e., components in CFG Models are "active" receivers (Cota and Sargent 1992). The timestamp used on an intercomponent message is the time at which the message is sent. (This is in contrast to the method generally used in parallel and distributed simulation in which the timestamps specify the time at which the messages are to be received.) The specification of channels is accomplished via an Interconnection Graph. An Interconnection Graph is a directed graph where the nodes represent the model components and the directed edges represent the channels along which intercomponent messages flow.

Cota and Sargent (1990b) developed a set of algorithms for the execution of CFG Models that allow CFG Models to be executed on either sequential computers or parallel/distributed computers without any additional modeler input. The availability and quality of "lookahead" information is a key element in parallel/distributed simulation (Fujimoto 1990). The parallel/distributed execution algorithms for CFG's use information that is "explicit" in the model's representation in order to generate "automatic lookahead" information, thus eliminating the need for a modeler to manually add "lookahead" information as is the common practice.

1.2 Evolution

While the CFG Model representation language can be used for modeling, it was not designed for that purpose. The Control Flow Graph representation is straightforward to use in the modeling of simple systems but can become complex when modeling more complex systems. HCFG Models were developed as a hierarchical modeling (specification) language that allows CFG Models to be used as a model representation language, thus preserving the ability to use the existing CFG Model execution algorithms for model execution. HCFG Models are a true superset of CFG Models in that any valid CFG Model is also a valid HCFG Model.

HCFG Models are used for model development. The HCFG Models can then be transformed into equivalent CFG Models (Cota, Fritz, and Sargent 1994) and executed using any of the CFG algorithms. The relationships between HCFG Models, CFG Models, and the execution algorithms are shown in Figure 1.

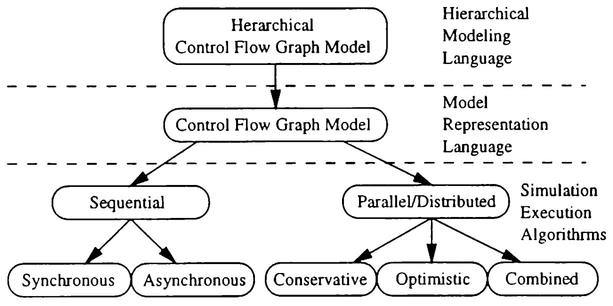
Herarchical Control Flow Graph Model
Hierarchical Modeling Language
Control Flow Graph Model
Model Representation Language
Sequential
Parallel/Distributed
Simulation Execution Algorithms
Synchronous
Asynchronous
Conservative
Optimistic
Combined

Figure 1: Modeling Language, Representation Language, and Algorithms

There are two primary objectives for HCFG Models. The first objective is to facilitate model development by making it easier to develop, maintain, and reuse models and model elements. This objective is facilitated by supporting the reuse of encapsulated model elements and the use of hierarchical structures as an aid in managing model complexity. The second objective is to support the flexible and efficient execution of models. This objective is addressed by providing an algorithmic mapping from HCFG Models into equivalent CFG Models (Fritz and Sargent 1993), thus enabling execution using CFG execution algorithms.

HCFG Models use two independent and complementary types of hierarchical model specifications. The first type of specification, called a Hierarchical Interconnection Graph (HIG), is used to specify the set of encapsulated components which comprise the model and how those components are interconnected. The second type of specification, called a Hierarchical Control Flow Graph (HCFG), is used to specify the behaviors of the individual model components. Each of the two types of model specifications have a graphical representation.

1.3 Organization

The remainder of this paper is organized as follows. Section 2 discusses the specification of the components which comprise a model and how those components are interconnected. Section 3 then discusses how behaviors of individual atomic components are specified using the paradigm. A brief overview of simulation execution is given in Section 4, and the use of "experimental frames" is discussed in Section 5. Finally we summarize this paper in Section 6.

2 COMPONENTS AND CHANNELS

In an HCFG Model, the model components and their interconnections (i.e., the channels) are specified via a HIG. A HIG is a hierarchical extension of the CFG Model Interconnection Graph which allows the modeler to specify model components hierarchically by supporting the concept of "coupling" together existing model components to form new model components. A HIG specifies the components which comprise the model and how those components are interconnected. The interconnection specification is a message routing specification that uses a set of channels to define a static message routing pattern for all intercomponent messages transmitted over the course of a simulation. Each model has exactly one HIG.

2.1 Components

The basic building block in the HIG is the model component. Model components are encapsulated entities which have an external view and an internal view. From the external view, all model components have the following attributes: a name (instance name), a type (type name), a set of input ports, and a set of output ports. (Internal views are covered below.) The distinction between "instance" and "type" is significant. If multiple model components are "instances" of the same type of component, then those components all share the same type definition.

The encapsulation boundary formed by a component means that the internals of a component are hidden from the outside view. The converse also holds. The exception to this "hidden" rule is the component's set of ports. This is because ports cross the encapsulation boundary and are thus visible on both sides (internal and external views). Ports have the same identifier (name) on both sides of a component encapsulation boundary.

The HCFG Model paradigm specifies model elements and relationships but does not dictate how these elements and relationships should be represented. In this paper we use graphical representations when we feel they more clearly convey information than a textual representation would. Some conventions we follow for the graphical representation of model components are: components are represented via boxes, channels are represented by line segments and their directions by arrows, port identifiers (names) are given near the ports, component type names start with an uppercase letter and are enclosed in parentheses "()", and instance names start with a lower case letter and are not enclosed in parentheses.

An external view of a simple example of a component is shown in Figure 2. This model component named "theBlueServer" is of component type "ExpServer". It has three input ports: "new–jobs", "suspend–operation", and "restart–suspended–job", and one output port: "completed–jobs".

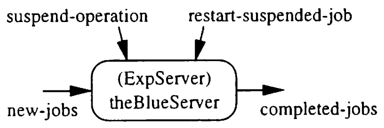

Figure 2: Example: A Simple Component

HCFG Models use two different classes of model components: atomic and coupled. Atomic Components (AC's) correspond to the components used in CFG Models (i.e., each component is an independent, encapsulated, concurrently operating entity whose behavior is specified via a corresponding component behavior specification). Behavior specifications for AC's is discussed in Section 3. Coupled components are discussed in the following subsection.

2.2 Coupled Components

Coupled components are encapsulated model components formed by coupling together other components (atomic and/or coupled) to form new components. Coupled components do not have behavior specifications. The internal view of a coupled component is the view from inside the component but outside all enclosed subcomponents. The internal view of a coupled component is specified via a "Coupled Component Specification (CCS)". A CCS specifies (1) a set of subcomponents which are coupled together to form a new coupled component type and (2) how those subcomponents are interconnected. Note that a CCS defines a component type and all instances of that type of component in a model share the single type definition.

Figure 3(a) shows how three components, "a1", "a2", and "b" can be coupled together to form a new coupled component type "C". A "coupling" defines a port to port mapping which determines the routing of all intercomponent message traffic. From the internal view of a component, we refer to ports which connect components within that component to the "outside world" as "external ports". Thus component type "C" in Figure 3(a) has an external input port "process" and an external output port "done". Note that the port "process" is connected to port "new-jobs" of component "b". An external view of component type "C" is shown in Figure 3(b). Instances of this new component type "C" are encapsulated model components which can be used anywhere in a model that a component is required. A component's port names are identical from both the external and internal views of the component. Figure 3 also shows two instances of the same type of component "(A)". The port identifiers (names) of the two instances of "(A)" are identical.

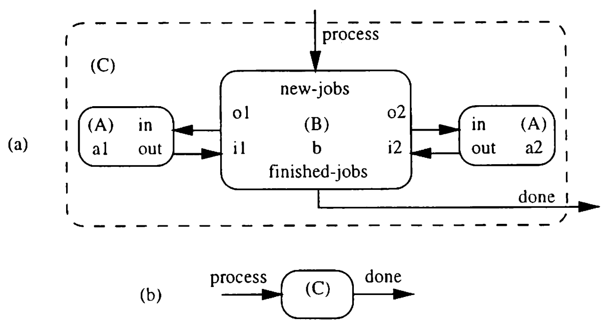

Figure 3: Coupling of Components

2.3 Hierarchical Structures

Coupled components support the development of hierarchical models using "top down" recursive decomposition of components into smaller and simpler components, "bottom up" composition of existing components to form new components, or a combination of these two methods. Using a "top down" modeling approach for HCFG Models one would first define a component type and later specify the internal view (definition) for the component type. HCFG Models can be constructed by recursively partitioning components into a set of coupled subcomponents until each of the remaining (non-partitioned) components have behaviors that can be easily specified using HCFG's. All non-partitioned components are AC's and thus have behavior specifications. If a particular model component has a "natural parallelism" in its behavior, then that component is a candidate for partitioning since each AC has its own thread of control.

We represent the hierarchical relationship of the set of components which comprise an HCFG Model using a structure called a "HIG tree". A HIG tree is a rooted tree structure in which the nodes of the tree represent the model components. A HIG tree shows the hierarchical relationship of the components in a model, but none of the interconnections. A HIG tree can be automatically generated from a HIG, but the converse is not true as the interconnection (coupling) information is not present in the HIG tree. Each HCFG Model has one coupled component type which encloses (and defines) the entire model. We commonly refer to this component as the "root" or "top level" component as this component represents the root node of the HIG tree. This top level component is the only component in a model which has no external ports. All internal nodes of the HIG tree represent coupled components and the leaf nodes of the HIG tree represent AC's.

We also have a "HIG type tree" which is a similar structure to the HIG tree. In a HIG type tree, the nodes of the rooted tree represent component "types" rather than components. A HIG type tree can be automatically generated from a HIG tree, but the converse is not true as the component name information is not present in the HIG type tree. We next use a simple example to illustrate the concepts of the HIG and HIG type trees.

Suppose that we have an HCFG Model whose top level component "M" is as shown in Figure 4. Assume that component types "A" and "C" are as shown in Figure 3 and also that component types "A" and "B" are AC's. The HIG for this model is completely specified by two CCS's (Coupled Component Specifications), one for the top level component type "M", and one for the coupled component type "C". No CCS is required for component types "A" and "B" as they are AC's and thus have behavior specifications instead of CCS's.

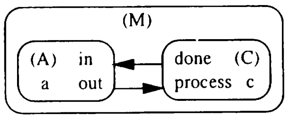

Figure 4: A Top Level CCS

The HIG tree for this model would be as shown in Figure 5(a) (component type names are shown in parentheses). The HIG type tree for this model is shown in Figure 5(b). Since all names in the HIG type tree are type names, there is no need to enclose the names in parentheses as is done in the HIG tree. The three vertical bars in Figure 5(b) indicate replication (i.e., more than one component of type "A" is contained within a component of type "C"). The "(2)" next to the three vertical bars indicates that there are two components of type "A" are contained in a component of type "C" as subcomponents.

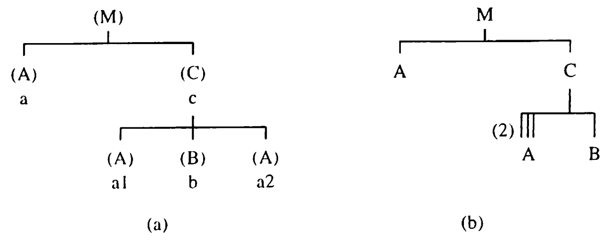

Figure 5: HIG Tree and HIG Type Tree

An HCFG Model consists of a set of model components arranged in a hierarchical (rooted tree) structure. The top level component of the HIG tree encloses and defines the model itself, the internal nodes of the HIG tree are coupled components which specify a coupling of lower level components (via a CCS), and the leaf nodes of the HIG tree are AC's which have behavior specifications. In the following section we discuss how the behaviors of the individual AC's are specified in an HCFG Model.

3 BEHAVIOR SPECIFICATION

The behavior of each type of AC in an HCFG Model is specified using an HCFG. An HCFG is a hierarchical extension of CFG's which allows the modeler to use hierarchy in the specification of the behavior of an AC type by supporting the recursive decomposition of the component's behavioral state space.

3.1 Control Flow Graphs

In a CFG Model the behavior of an AC is specified using a state based specification called a CFG. A CFG is required for each distinct type of AC in a model. A CFG consists of an augmented directed graph, where the nodes are control states (A control state is a formalization of the "process reactivation point" (Cota and Sargent 1990a; Zeigler 1976)) and the edges show the possible control state transitions. (Edges may originate and terminate on the same node.) Each AC is encapsulated and has a set of variables (including a (local) simulation clock) that are local to that AC. Associated with each edge are 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.

3.1.1 Edge Conditions

The condition attribute on CFG edges can be classified into one of three types: "time delay", "non-empty input port", and "boolean expression". Edges with a time delay condition, called "TimeEdges", have an associated time delay function. A TimeEdge's condition becomes **true** after a simulation time delay specified by its associated time delay function. This time delay function may return any non-negative value (including zero, which indicates that the condition is **true** immediately). Edges with a non-empty input port condition, called "PortEdges", are associated with an input port of the AC. A PortEdge is **true** if there exists an unreceived message waiting on the associated input port and **false** otherwise. Edges with a boolean expression condition type, called "BoolEdges", have an associated boolean expression. BoolEdges are **true** if the boolean expression (which may reference only local variables of the AC) evaluates to **true**, and **false** otherwise. We also define a subtype of BoolEdge, called a "TrueEdge", to be a BoolEdge whose condition is defined to always evaluate to **true**.

We use the graphical notation shown in Figure 6 to distinguish the different types of edges and their associated condition and event attributes. A TrueEdge is represented as a BoolEdge with a capital "T" near the edge type symbol. (Functions are indicated by adding a "()" suffix to the function name.) Port identifiers are not functions and thus do not have this suffix.

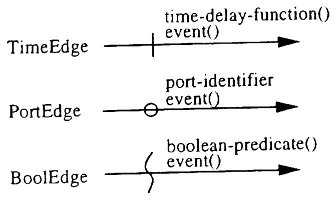

Figure 6: Edge Notation

3.1.2 Events

The event attribute associated with each edge specifies the action that is taken whenever that edge is traversed during the course of simulation execution. The action taken may include changing the values of the AC's local variables, sending messages to output ports of the AC, or receiving a message from an associated input port. Any event may send messages to one or more output ports, however only events that are associated with PortEdges may receive messages from input ports. An event associated with a PortEdge receives only one message during each execution (traversal of the PortEdge), and the message received is from the input port which is associated with that PortEdge. If no action is to be taken during the traversal of a specific edge, we say that the event associated with that edge is the "null event" (commonly represented as "e-null()" or "e_{null}").

3.1.3 Operational Semantics

Each AC is an independent encapsulated entity with its own thread of control. Each AC has what is referred to as a "Point of Control" (POC). The POC for an AC always resides at a control state. The control state at which the POC currently resides is called the "current" control state. The conceptual operation of an AC is as follows. The simulation execution algorithm examines all edges leaving the current control state and selects the edge whose condition attribute will first (at the earliest point from the current simulation time) assume the value **true**. If this selection process returns more than one edge, then the edge with the highest priority is selected for traversal. To execute its next simulation event, the AC advances its local simulation clock, if necessary, to the time at which the selected edge's condition attribute becomes **true**. The AC's POC then traverses the selected edge to the control state which the selected edge terminates on, executing the associated event routine during the edge traversal. The control state that the POC arrives on during the edge traversal becomes the new current control state and the process of selecting an outbound edge begins again. This operation is repeated for each AC until the simulation terminates. (Note that an edge traversal over an edge with the null event may advance the AC's local clock. This is because the clock advancement occurs prior to and independent of the event routine associated with the edge.)

3.1.4 Control Flow Graph Example

To demonstrate how a CFG is used to model the behavior of an AC, we show how to model the behavior of a simple queuing server. This server handles two classes of jobs using a "priority preempt/resume" job selection discipline. Each job is either a "high priority" job, or a "low priority" job. Jobs within each class are processed on a First Come First Serve (FCFS) basis. The server always works on a high priority job if one is available and high priority jobs are always run to completion once they start service. If the server is busy with a low priority job when a high priority job arrives, the low priority job is preempted (work on it is suspended) and the server then begins working on the high priority job. The server processes high priority jobs until there are no more high priority jobs to work on. Work on a suspended low priority job is then resumed where it left off. The external view of the AC type is shown in Figure 7.

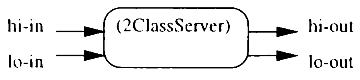

Figure 7: "2ClassServer" Type AC External View

Since the AC we are modeling in this example is a server that handles two classes of jobs, we assign this AC the type name: "2ClassServer". A "2ClassServer" AC has two input ports "hi-in" and "lo-in", and two output ports "hi-out" and "lo-out". In an AC of type "2ClassServer", each "job" arrival or departure is represented by a message. Thus a job arrival or departure is synonymous with (and represented via) a message arrival or departure respectively in our "2ClassServer". The priority of a job arriving at the server is determined by the port on which it arrives. Thus high priority jobs arrive on input port "hi-in" and low priority jobs arrive on input port "lo-in". As jobs finish service, they are sent out (as messages); high priority jobs on "hi-out" and low priority jobs on "lo-out".

We will model the behavior of the "2ClassServer" AC using a CFG with four control states. We name these four control states: "I", "BL", "BH", and "P" which stand for "Idle", "Busy-Low", "Busy-High", and "Preempt", respectively, as shown in Figure 8. When the POC is at control state "I" the server is idle. When the control state is at "BL" the server is working on a low priority job (no high priority jobs are available). When the POC is at "BH" or "P" the server is working on a high priority job. If the POC is at "P" there is also a "suspended" low priority job which will be "resumed" when there are no more high priority jobs. Three PortEdges and three TimeEdges show the possible control state transitions. An overview of the behavior of this CFG is given next.

The POC will remain at control state "I" until a job is available. If a high priority job is available (there is an unreceived message on input port "hi-in") the POC will traverse the edge to "BH", executing the event "start-hi()" during the traversal. If a low priority job is available (there is an unreceived message on input port "lo-in") the POC will traverse the edge to "BL", executing the event "start-lo()" during the traversal. (A PortEdge event routine receives a single message from the associated input port in addition to any other action it may take.) If both types of jobs

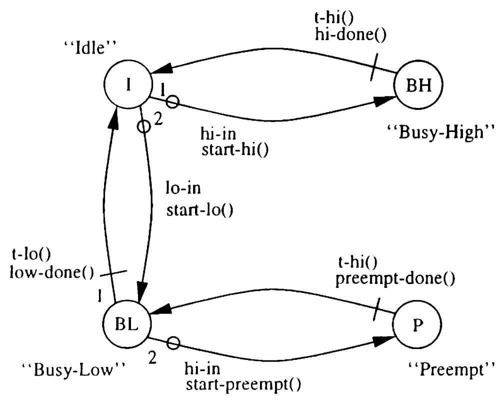

Figure 8: "2ClassServer" Control Flow Graph

are available, the POC will move to "BH" because the edge to "BH" has an edge priority of "1" which is higher than the edge to "BL" which has a priority of "2". (Lower numbers indicate higher priorities. Also note that there is no need to explicitly indicate a priority on edges where there is only one edge leaving a control state.)

When the POC enters either "BH" or "P", processing of a high priority job begins. Processing then continues for the duration specified by the time delay function "t-hi()" associated with the edges leaving "BH" and "P". After the specified time delay the POC moves again. The event routines associated with the edges leaving "BH" and "P" send a message to the "hi-out" output port indicating the completion of a high priority job. Even though the edges leaving "BH" and "P" share the same time delay function, they must have different event routines because the edge leaving "P" must also restore the state of the "preempted" low priority job that was saved previously by the "start-preempt()" event. The definition of the "t-hi()" time delay function is straightforward as high priority jobs always run to completion once started. The "t-lo()" time delay function on the TimeEdge from "BL" to "I" is more complex because it is based on the "remaining" processing time required by a low priority job which may have been preempted one or more times before completing service.

3.2 Hierarchical Control Flow Graphs

Behavior specification using CFG's is straightforward for the modeling of simple systems but can become complex when modeling the behavior of larger and more complex AC's. As the complexity of an AC's behavior increases it becomes more difficult to model that behavior using CFG's due to two primary factors. The first is the possible explosion of the number of control states and edges required to model a complex behavior, and the second is that all names within an AC are in the same name space. (The name space concern of CFG's is analogous to a programming language in which all variables are global variables. Having a single crowded name space can be an inconvenience to the modeler and can lead to difficult to detect errors in a CFG specification.)

Hierarchical Control Flow Graphs are a hierarchical extension of CFG's which address these issues by allowing the modeler to partition an AC's behavior specification into a disjoint set of encapsulated partial behavior specifications called Macro Control States (MCS's). These MCS's provide a fine granularity reuse capability for HCFG Models by supporting model element reuse at the sub-AC level.

3.2.1 Macro Control States

An AC's behavioral state space can be recursively partitioned (into MCS's) until the remaining partial behavior specifications can easily be modeled using simple CFG's. This recursive partitioning of the behavior specification forms a tree structure called an HCFG tree in which the nodes of the tree represent MCS's. The top level MCS of the HCFG tree encloses the entire behavior specification for the AC type. A CFG is simply a special type of an HCFG in which the behavior specification of the AC is contained entirely within the top level MCS; i.e., the top level MCS has no child MCS's and thus does not partition the AC's behavioral state space). HCFG's can be algorithmically mapped into CFG's (Fritz and Sargent 1993) prior to model execution in order to utilize the existing algorithms for CFG Models.

A MCS (pronounced "max" as in "maximum") encapsulates one or more control states and/or child MCS's and their associated edges. (A MCS either contains another MCS in its entirety, or not at all.) Edges over which the POC (Point of Control) flows enter and leave MCS's via "pins". Pins pierce the encapsulation boundary of MCS's in a manner analogous to how ports pierce the encapsulation boundary of components in the HIG (i.e., pins names are the same on both sides of the encapsulation boundary).

A MCS is similar to an encapsulated subgraph of a CFG, but with the following extensions and restrictions. Because a MCS is an encapsulated entity, the outside of an MCS is not visible from the inside of the MCS, and the converse also holds (with the exception of the pins). The encapsulation boundaries formed by MCS's also partition the function and variable name space within the AC's behavior specification. A MCS (i.e., the conditions and events of the MCS) may not access functions or variables defined within other MCS's in the HCFG, with the possible exception of those belonging to its parent MCS. A MCS may access only those functions and variables of its parent MCS to which it has been given explicit permission to access.

Edges leaving control states in a MCS are identical to those in CFG's. Edges in MCS's terminate on one of the MCS's external output pins, one of the input pins of a child MCS, or on a control state within the MCS. There is no limit on the number of edges which terminate on a pin, however, *exactly* one edge originates from each pin. Edges originating from pins do not have any attributes, that is, they do not have the three edge attributes (condition, priority, event) that edges originating from control states have.

MCS's have types (type names) and instances (instance names) in a manner analogous to components. We follow the same conventions with MCS's that we used for components (i.e., type names are enclosed in parentheses and start with an uppercase letter and instance names start with a lower case letter). A capability that MCS's have that components do not is that MCS's can be parameterized. A list of parameters may be passed to a MCS type upon "instantiation" (i.e., when an instance of a MCS is created from the MCS's type specification). The "explicit permission" to access information belonging to a MCS's parent is granted via these parameters.

3.2.2 Operational Semantics

The operation of an HCFG is an extension of the operation of a CFG. The POC for an AC resides at a control state called the current control state. This current control state is contained within a specific MCS, called the current MCS. Edges are selected in the same manner as in CFG's. The POC leaves the current control state over the selected edge and the action specified by the event routine associated with the traversed edge is carried out just as in CFG's. However, in an HCFG, the selected edge may terminate on either a control state within the same MCS, on an input pin of a child MCS, or on an output pin of the current MCS. If the selected edge terminates on a control state within the current MCS then the operation is identical to that of CFG's. If the selected edge terminates on an input pin of a child MCS, then the POC enters that child MCS through the input pin. If the selected edge terminates on an output pin of the current MCS, then the POC leaves the current MCS and enters the current MCS's parent MCS in the HCFG tree via the current MCS's output pin.

When the POC traverses an edge that terminates on a pin, the POC will then continue to traverse a sequence of directed edges, starting with the edge leaving the pin, until it eventually arrives at a control state. In contrast to control states, which may have an arbitrary number of outbound edges, each pin has exactly one outbound edge, thus no edge selection algorithm is required for edges leaving pins. Since edges which originate from pins do not have the set of three attributes (priority, condition, and event) that edges originating from control states do, there is never a condition test required before traversing an edge originating from a pin, and there is no associated event to be executed during the traversal of an edge originating from a pin.

In the following subsection we show a simple example of AC behavior modeling in HCFG Models using MCS's.

3.2.3 Simple MCS Example

We show a variation of the "2ClassServer" example from Subsection 3.1.4 to demonstrate the use of MCS's for partitioning behavior specifications. (The partitioning demonstrated in this example was selected for illustrative purposes only as this particular partitioning is not particularly interesting, useful or efficient.) To encapsulate the behavior of control state "BH" and its outbound TimeEdge from our original CFG (Figure 8) we first draw an encapsulation boundary around control state "BH" as shown in Figure 9(a). We then replace this encapsulation with a child MCS named "doHiJob" of type "(DoHiJob)" as shown in Figure 9(b). (We represent MCS's contained within other MCS's graphically as ellipses so that it is easy to distinguish between the MCS's and control states (which are represented as circles).) This "doHiJob" MCS has two pins: "in" and "out" which specify, respectively, the paths over which the POC can enter and leave the MCS.

The next step is to specify the internal view of a "DoHiJob" type MCS. A graphical representation of a "DoHiJob" MCS is shown in Figure 9(c). (The pins are shown as circles containing a "cross".) We see that a "DoHiJob" MCS has a single control state "S" which has one outbound TimeEdge. Note that the conditions and events shown in Figures 9(b) and 9(c) are contained in different (encapsulated) MCS's and thus are in different name spaces. If any condition or event functions in the "doHiJob" MCS need access to any functions or variables in its parent MCS this access must be explicitly granted via parameters passed to the "DoHiJob" type MCS when "doHiJob" is instantiated (created). This parameterization of MCS's greatly enhances their capability for reuse.

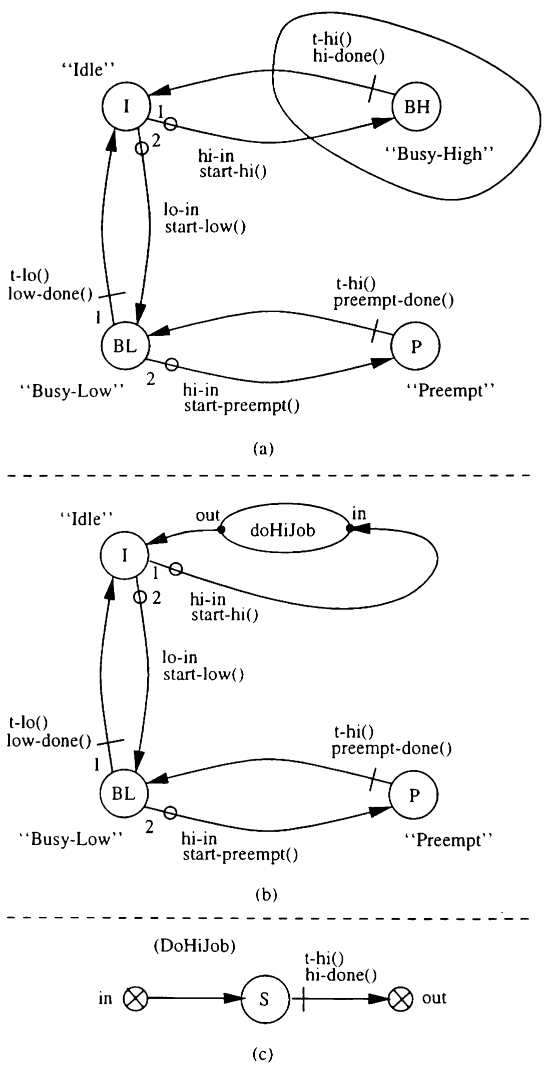

Figure 9: "2ClassServer" Using MCS's

4 EXECUTION OVERVIEW

In this section we provide a brief overview of model execution using the sequential synchronous simulation execution algorithm. (Other simulation execution algorithms are discussed in Cota and Sargent (1990b) and Fritz and Sargent (1993).)

Conceptually, an HCFG Model consists of a set of independent concurrently operating AC's. (Coupled components only specify message routing patterns.) The sequential synchronous simulation execution algorithm runs on a sequential computer and executes all events in strict time order. Each AC is assigned a unique priority that is used to break event time ties.

Each AC has either a pending edge or a conditionally pending edge which is selected from the set of edges leaving the AC's current control state by the simulation execution algorithm. A pending edge *is* the next edge that will be traversed for that AC. A conditionally pending edge may be preempted (and another edge selected) before the AC executes its next event due to the arrival of new messages. Each pending or conditionally pending edge has an associated next event time that defines the next event time for its AC. The AC with the earliest next event time is selected for execution, using AC priorities to break time ties. The selected AC advances its local simulation clock to the time of its next event and traverses the selected (pending or conditionally pending) edge in its HCFG, executing the event associated with the edge. Any messages transmitted by the selected AC during its event execution are propagated (as specified in the HIG) to the input port(s) of the receiving AC's. (A message arrival at an AC which has a conditionally pending edge may cause a change in that AC's next event time.) These steps are repeated until the simulation termination conditions for the model are met.

5 EXPERIMENTAL FRAME

HCFG Models support the use of experimental frames. The "Experimental Frame" concept separates a model's definition from the set of model parameters used for a specific execution run of the model. This allows a modeler to modify such items as the initial conditions, desired data collection, and termination conditions for a specific simulation run without modifying the model itself. For example, a user may wish to make several simulation runs for statistical analysis purposes using the same model while changing only the seeds used by the random number generators in the model between simulation runs.

An HCFG Model reads experimental frame information from an external source during a model initialization phase (prior to initiating model execution) and initializes its internal state accordingly.

6 SUMMARY

We presented the foundation upon which HCFG Models are based, discussed the motivations and objectives for the HCFG Model paradigm, and showed how those objectives were addressed. We then described the two complementary types of specification structures (components and interconnections, and behavior specifications) used in model specification. We also presented overviews of the execution of HCFG Models using the sequential synchronous algorithm and of the experimental frame concept supported by the HCFG Model paradigm.

REFERENCES

Cota, B., D. Fritz, and R. Sargent. 1994. Control flow graphs as a representation language. In *Proceedings of the 1994 Winter Simulation Conference*, J. Tew, S. Manivannan, D. Sadowski, and A. Seila (Eds.), pp. 555-559.

Cota, B. and R. Sargent. 1990a. Control flow graphs: A method of model representation for parallel discrete event simulation. CASE Center Technical Report 9026, Syracuse University.

Cota, B. and R. Sargent. 1990b. Simulation algorithms for control flow graphs. CASE Center Technical Report 9023, Syracuse University.

Cota, B. and R. Sargent. 1992. A modification of the process interaction world view. *ACM Transactions on Modeling and Computer Simulation 2*(2), 109–129.

Fritz, D. and R. Sargent. 1993. Hierarchical control flow graphs. CASE Center Technical Report 9323, Syracuse University.

Fujimoto, R. 1990. Parallel discrete event simulation. *Communications of the ACM 33*(10), 30–53.

Zeigler, B. 1976. *Theory of Modelling and Simulation.* Wiley.

AUTHOR BIOGRAPHIES

DOUGLAS G. FRITZ is a graduate student at Syracuse University working towards a Ph.D. in Computer Engineering. His research area is hierarchical modeling for discrete event simulation. He received M.S. degrees in Electrical Engineering and Computer Science from Syracuse University and a B.S. degree in Electrical Engineering from The Pennsylvania State University. He was formerly with IBM as a development engineer for high speed switching systems.

ROBERT G. SARGENT is a Professor at Syracuse University. He received his education at the University of Michigan and has published widely. Dr. Sargent has served his profession in numerous ways and has been awarded the TIMS College on Simulation Distinguished Service Award for long-standing exceptional service to the simulation community. His research interests include the methodology areas of modeling and discrete event simulation, model validation, and performance evaluation. Professor Sargent is listed in *Who's Who in America.*