Hierarchical HLA: Mapping Hierarchical Model Structure into Hierarchical Federation

Jae-Hyun Kim* and Tag Gon Kim**
Department of EECS
KAIST
373-1 Kusong-dong, Yusong-gu
Daejeon, Korea 305-701
Tel: +82-42-869-3454
Fax: +82-42-869-8054
jhkim@smslab.kaist.ac.kr*, tkim@ee.kaist.ac.kr**

Keywords: Hierarchical HLA, Hierarchical Federation

The High Level Architecture (HLA) defines a federation as a group of simulators. The HLA does not define relations between federations. This makes any complicated structure of a system flat. The absence of hierarchy may bring several issues related to data exchange, security and reusability.

The paper proposes the Hierarchical HLA that defines hierarchical relationship between federations. The relationship is inspired by Discrete Event System Specification (DEVS) formalism, which provides hierarchical and modular specification of discrete event systems. A federation takes part in a larger federation on behalf of component federates. A federation filters data exchange with the higher federation according to the Simulation Object Model (SOM). In order to control hierarchical federations, extended HLA services or explicit Federation Execution Processes are introduced in the paper.

1 INTRODUCTION

Complexity of a system is one major reason that necessitates hierarchical and modular modeling methodologies. Without these properties, it is impossible to do detailed modeling on a complicated system. Hierarchy of models makes the modelers possible to do divide-and-conquer approach. The whole system can be divided into several interconnected components and the components also into smaller ones. Each component is mapped into a model in the hierarchy. Modularity of models makes the increase of reusability of models. The Discrete Event System Specification (DEVS) formalism [1] is a modeling and simulation methodology that provides hierarchical and modular characteristics.

The huge scale of models often needs more than one computer to simulate the entire model. The High Level Architecture (HLA) is IEEE standard 1516 for interoperation of heterogeneous simulations that are developed using different algorithms, languages, and platforms [2-4].

The current HLA only supports one level of federation that is a virtual group of simulators (or federates). The HLA has no definition about relations of federations. There is no way for a federation to communicate with another one. More specifically, a joined federate in a federation cannot interact with other joined federates in another federation through RTI although two federations are running at the same time. To do so, all federates should belong to the same federation. Therefore, the hierarchical model structure should be mapped into federates in one federation. This mapping probably ruins the original hierarchy and raises several issues related to data exchange, security and reusability.

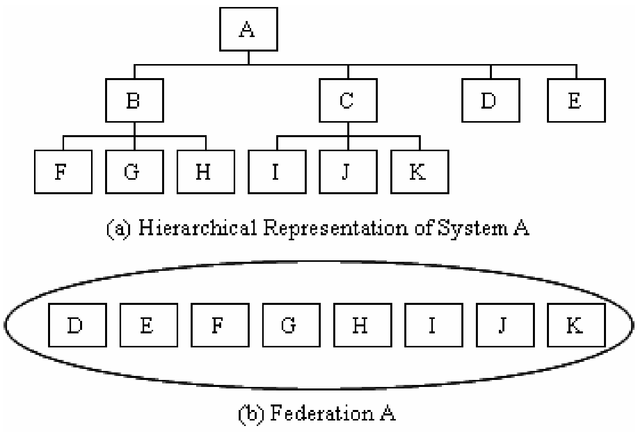

Figure 1 A Target System and its Federation

Figure 1 shows a hierarchical representation of an example system A and its possible implementation of federation A. Let us assume that the system A is composed of four components – B, C, D, and E. The component B and C is divided into subcomponent F, G, H and I, J, K, respectively. In this case, components from D to K have behavioral information while the component A, B, and C have structural information of the system.

When we try to make a federation in order to simulate the system A, we have to put all simulators in one federation. Each component from D to K can be implemented as a separate federate. The structural information in component A, B, and C can be used to build the FOM for the federation A.

This mapping may lead unexpected data flow in the federation. Let us assume that some data of type b1 is used inside of both components B and C, but there is no exchange of b1 between B and C. However, the federation A is not able to prevent data flow of b1 between {F, G, H} and {I, J, K} because each federate subscribes the same data type. The components need to check b1 whether it is from the right source. This extra flow possibly lowers the performance of the whole federation.

The problem can be simply avoided by renaming of data type b1 when the federation is designed. However, renaming requires modification of federate codes, which degrades the reusability of the federate.

This kind of data exchange may raise security issues if the component B and C are managed by different organizations or nations. In addition, any federate that is interested in data type b1 can receive the data without any restriction since there is no way to prohibit subscription of specific data types. This security problem is called Information Hiding Problem [6][7].

Another security issue is that all data types should be declared in the FOM in order to construct a federation. Even if there are some data types that are used only inside component B, those types should be included in the FOM. Therefore, it is impossible to keep internal data secured. Any data should be notified its existence and accessible to anyone.

There have been many researches that bring hierarchy among federations or enable inter-federation communication by introducing additional federates as gateways, bridges, proxies, or brokers [6-11]. However, it is only temporary solutions since the interconnecting federates have limited accessibility on internal RTI states. The HLA itself should be revised in order to support hierarchical federations.

Section 2 briefly reviews HLA and DEVS formalisms. Section 3 describes the Hierarchical HLA and the implementation issues of hierarchical RTI are addressed in the Section 4. Section 5 concludes the paper.

2 HIGH LEVEL ARCHITECTURE & DEVS FORMALISM: BRIEF REVIEW

2.1 High Level Architecture

The HLA is a specification for interoperation among distributed heterogeneous simulations. The HLA specification is approved as IEEE std 1516 in 2000. The standard consists of two parts; Federate Interface Specification and Object Model Template (OMT).

Federate Interface Specification defines essential services of 6 categories. These services are also listed as a form of Application Programming Interfaces (APIs) written in various programming languages. The Run-Time Infrastructure (RTI) is an implementation of HLA Federate Interface Specification. The HLA only defines APIs and lefts detailed algorithms to RTI software developers.

OMT is a documentation format for shared objects and message types by which simulators exchange events. OMT is used to describe Federation Object Model (FOM) and Simulation Object Model (SOM). The FOM is a collection of all shared objects and interactions defined inside whole distributed simulation, i.e. federation. The SOM is a list of shared objects and interactions that a simulator, i.e. federate, sends to or receives from other simulators.

The HLA/RTI provides explicit time management services, while traditional parallel and distributed time synchronization algorithms manage the simulation time of a local process with only input events from other processes. The HLA/RTI has a list of time services that enable a federate to facilitate event-driven, time-stepped, or optimistic time synchronization methodology.

A federate is able to use two communication methodologies for exchanging data with other federate; shared objects and interactions. There exist, however, no physically shared objects inside a federation because each federate is a modular and distributed across the network. In a modular system, a component should notify the modification of its internal variable to be shared with other components whenever the variable is changed. To implement non-modular components in a modular system, in other hands, to emulate a shared object in distributed environment, the shared objects need to be declared, first, in the FOM and to be managed by lists of notification functions in the HLA.

Finally, the HLA supports control functions to organize a federation, miscellaneous functions for synchronization points, save and restoration, monitoring and management functionality by Management Object Model (MOM).

2.2 DEVS Formalism

The DEVS formalism specifies discrete event models in a hierarchical and modular form [1]. With this formalism, one can perform modeling more easily by decomposing a large system into smaller component models with coupling specification between them. There are two kinds of models: atomic model and coupled model.

An atomic model is the basic model and has specifications for the dynamics of the model. Formally, a 7-tuple specifies an atomic model M as follows:

$$M = < X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta >,$$

where

- X: a set of input events;
- Y: a set of output events;
- S: a set of sequential states;
- δ_{ext}: $Q \times X \rightarrow S$, an external transition function, where $Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}$ is the total state set of M;
- δ_{int}: $S \rightarrow S$, an internal transition function;
- λ: $S \rightarrow Y$, an output function;

ta: $S \rightarrow R^{+}_{0,\infty}$ (non-negative real number), time advance function.

A coupled model provides the method of assembly of several atomic and/or coupled models to build complex systems hierarchically. Formally, a coupled model is defined as follows:

$$DN = < X, Y, M, EIC, EOC, IC, SELECT >,$$

where

X: a set of input events;

Y: a set of output events;

M: a set of all component models;

$EIC \subseteq DN.X \times \cup M.X$: external input coupling;

$EOC \subseteq \cup M.Y \times DN.Y$: external output coupling;

$IC \subseteq \cup M.Y \times \cup M.X$: internal coupling;

SELECT: $2^M - \emptyset \rightarrow M$: tie-breaking selector.

An overall system consists of a set of component models, either atomic or coupled, thus being in hierarchical structure. Each DEVS model, either atomic or coupled model, has correspondence to an object in a real-world system to be modeled. Within the DEVS framework, model design may be performed in a top-down fashion; model implementation in a bottom-up manner.

3 HIERARCHICAL HLA

3.1 Design Principles

This paper proposes the Hierarchical HLA in order to solve problems discussed in the previous section. The Hierarchical HLA is the extension of the HLA by including hierarchical relations between each federation

The design of the hierarchical federation is inspired by the DEVS formalism, which provides modular and hierarchical specification of discrete event systems.

In the DEVS formalism, a coupled model is composed of multiple models. Each component model can be decomposed into several subcomponents and this decomposition is continued recursively to some finite level. The indivisible component is specified as an atomic model.

Similarly, a federation is composed of not only a set of federates but also other federations. A federation is analogous to a coupled model, while a federate is to an atomic model.

System theory is *closed under composition* in that the structure and behavior of a composition of systems can be expressed in the original system theory terms. Closure under composition guarantees that such a composition results in a system with well-defined structure and behavior. *Modular* systems have recognized input and output ports through which all interaction with the environment occurs [1].

The closure under composition property can be applied to the hierarchical HLA. The hierarchical federation will have well-defined structure and behavior if the composition of systems, i.e., a federation, can be expressed as a federate. In other words, the federation should behave as same as other sibling federates when a federation is a member of a bigger federation.

Modularity is obtained when internal states of a system are hidden and only an interface to the environment is visible. A component federation becomes modular if internal federates are hidden to outer federations.

From these perspectives, an interface of a federation to the upper federation should be defined consistently with that of a federate. In this context, we call a component federation a *representative* federate. The component federation represents its member federates on the upper federation.

In the previous example in Figure 1, the hierarchical federations will be identical to the hierarchical representation of the system. The topmost federation A is composed of two federations B and C, and two federates D and E. B has federates F, G, H and C includes I, J, and K. The federation B and C is a representative federate in the federation A.

3.2 Data Management

This section proposes data management methodology for a federation as a representative federate.

A federate publishes objects and interactions that it generates and subscribes what it is willing to receive. The objects and interactions should be documented in SOM with OMT. Consistent with this interface, a federation needs to have its own way to declare what to send and receive.

The SOM for a federation is introduced. Similar to that of a federate, the SOM for a federation has publication and subscription information of objects and interactions to the outer federation. According to the SOM, the federation automatically filters data through the border of the federation.

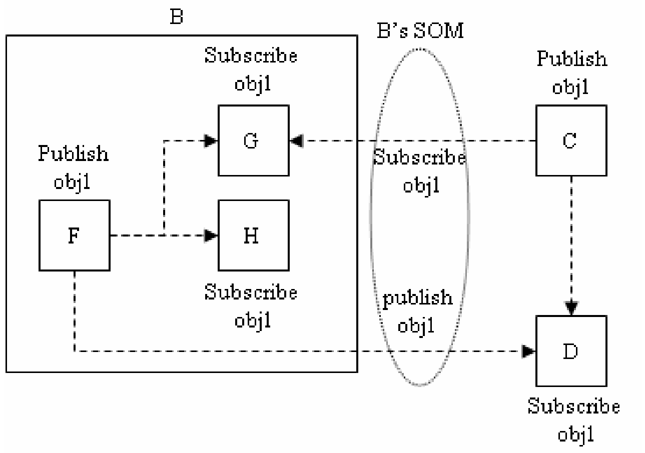

Figure 2 Federation B's SOM

Figure 2 shows an example of a SOM for federation B. The dotted arrow indicates automatic couplings by RTI according to publication-subscription relations. The obj1 type can be delivered from C to G and from F to D since the obj1 is declared in B's SOM. If obj1 is not declared in B's SOM, only C and D are able to exchange obj1.

The methods that actually make federations publish and subscribe its SOM will de discussed in Section 4.

3.3 Time Management

This section proposes time management methodology for a federation as a representative federate. The federate time of the representative federate will be defined as follows:

$$\text{federation time} = \min(\text{federate time}_i + \text{lookahead}_i)$$

The summation of the federate time and lookahead is the minimum time with that the federate can generate an event. Therefore the federation time is the minimum time when a federation is able to produce an event. This time can be used in Lower Bound Time Stamp (LBTS) calculation of the federation.

3.4 Federation Management

This section proposes how to handle federation management services in the hierarchical federations.

There are two types of federation management services – consensus and initiation services. A consensus service requires all federate to call the service in order to initiate callback functions in all federates. For example, Federation Synchronized callback is called by RTI only after all federates invokes Synchronization Point Achieved services. An initiation service is that a request of the service by a federate initiates callback functions in all federates. For example, if a federate calls Request Federation Save service, the Initiate Federate Save callback is invoked in all federates.

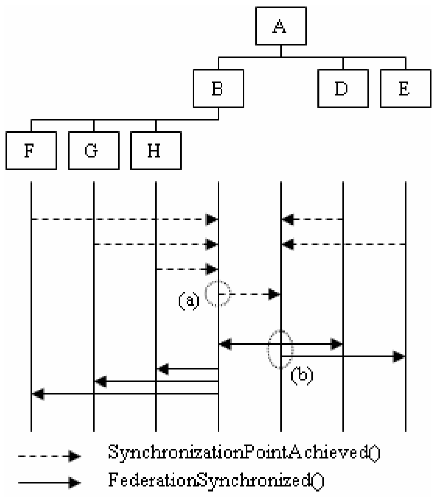

Figure 3 Consensus Service

Figure 3 shows an example sequence of RTI consensus services in the hierarchical structure. In this case, a federation acts differently according to its position in hierarchy. In the figure, point (a) and (b) describes this difference. If a federation exists in the middle of a hierarchy tree, the federation forwards a consensus service to higher federation instead of replying callback functions.

The federation B calls Synchronization Point Achieved service on behalf of member federates when it receives same requests from all members.

The federation B invokes Federation Synchronized service to all its member federates when it receives the same service from federation A.

Figure 4 shows an example sequence of RTI initiation service in the hierarchical structure. Similar to the consensus services, a federation in the middle of hierarchy forwards the initiation service to the higher federation, and the federation invokes callback functions after it receives the callback function from the higher federation.

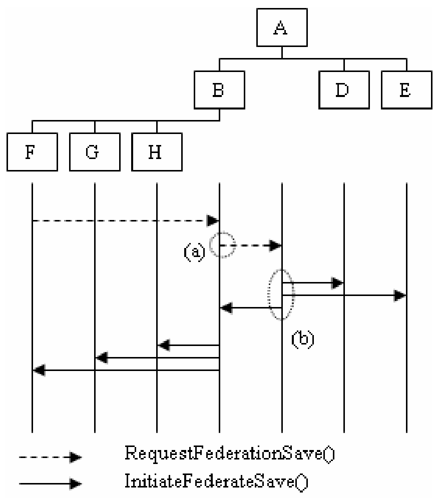

Figure 4 Initiation Service

4 Implementation of Hierarchical Run-Time Infrastructure

The design and implementation of RTI is not specified in the HLA standards. One big design issue can be the existence of Federation Execution Processes. The Federation Execution Process is a process that handles federation-related services. The process keeps information about member federates and helps their synchronization and data exchange. Figure 5 shows two possible options.

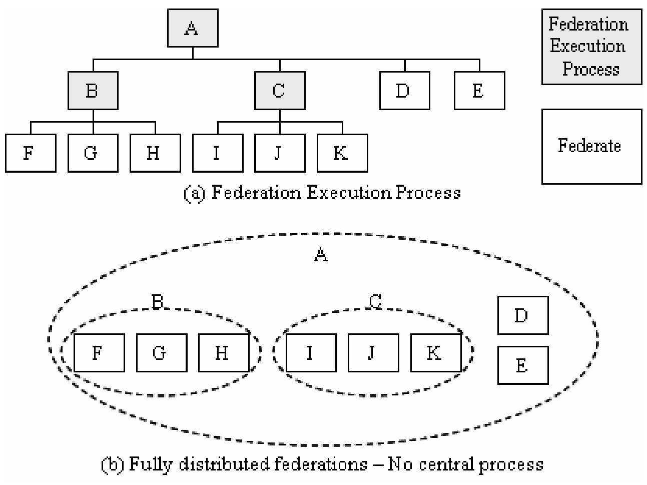
A
B
C
D
E
F
G
H
I
J
K
Federation Execution Process
Federate
(a) Federation Execution Process
(b) Fully distributed federations – No central process

Figure 5 Implementation of Hierarchical RTI

If RTI chooses architecture with Federation Execution Processes, the functionality and relations between processes are quite straightforward. One big possible drawback can be the number of Federation Execution Processes. As the hierarchy goes deeper, the whole system requires a lot of extra processes except federates.

It is possible to form fully distributed federations without the Federation Execution Process. However, the functionality for a federation should be distributed over federates and the design of RTI library becomes complicated.

There are two methods for developers to control federations. One is to manipulate explicitly the Federation Execution Process. The other is to use extended APIs for controlling hierarchical federations when implementing a federate. If RTI is implemented without Federation Execution Processes, the second method is the only option. However, the API method has risk to ruin modularity and reusability. If a federate manipulates federations, the federate is tightly coupled with the federation and the federate can not be used in other federations. Therefore it is desired for RTI to provide Federation Execution Processes that can be configured by users.

In this section, we propose methods for federation, data, and time managements in both ways.

4.1 Federation Management

Let us take look at extended HLA services that is used to construct hierarchical federations. The extended parts are written in italic.

Create Federation Execution
Supplied arguments:
a) Federation execution name
b) FOM document designator
c) SOM document designator

Join Federation Execution
Supplied arguments:
a) Federate type *or Federation execution name (child)*
b) Federation execution name
Returned arguments:
a) Joined federate designator

Resign Federation Execution
Supplied arguments:
a) Directive
b) Federation execution name (optional)

Join and Resign Federation Execution services are designed for a federate to join and resign its joined federation to upper federation.

Create Federation Execution service has new argument of a SOM document designator. When creating a federation, a SOM file that contains publication and subscription information of the federation is associated to the federation.

If RTI supports Federation Execution Processes and the Federation Process is created by extended APIs, it is usually impossible for Federation Processes to select a platform to run. RTI creates the Federation Process at the same location with the RTI process. Therefore, this can cause resource bottleneck if there exist many federations in the hierarchy.

One way to avoid the problem is to manually set up the Federation Execution Processes. Users start the Federation Execution Processes with arguments of the SOM file and the name of parent federation to join.

4.2 Data Management

If RTI operates in fully distributed way, extended services can be used to control publication and subscription information of the federation. For example, an additional argument of federation execution name in the following service indicates that publication of object class attributes is for the federation. Similarly, other related services can be extended by adding an additional argument.

Publish Object Class Attributes
Supplied arguments:
a) Object class designator
b) Set of attribute designators
c) Federation execution name (optional)

If RTI supports Federation Execution Processes, the process automatically publishes and subscribes objects and interactions according to SOM file that is associated when the process starts.

There is a difference between a federate and a federation with regard to the way of treating SOM files. A federate does not explicitly use a SOM file. Instead, a federation is supplied with a FOM file when it is created. However, if a federate uses a SOM file to initialize publication and subscription information, the interfaces of a federate and a federation become identical. This process is called SOM-based Join Process and proposed in the previous work [5].

4.3 Time Management

Unlike the federation management or data management, hierarchy among federates may not affect time management algorithms.

The developers of RTI are able to apply any asynchronous time management algorithms for parallel and distributed discrete event systems. Or the developers can use synchronous time management algorithms using hierarchical tree structure of Federation Execution Processes and federates.

5 CONCLUSION

The paper proposed Hierarchical HLA which defines hierarchical relations between federations. By preserving hierarchical structure of the system in the federations, security, reusability and performance issues can be solved.

We are currently working on development of a prototype hierarchical RTI. We have plans to research the effects of the hierarchical structure on the performance of simulation.

6 References

[1] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim, *Theory of Modeling and Simulation*, Academic Press, 2000

[2] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules, Std 1516, 2000.

[3] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification, Std 1516.1, 2000.

[4] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT), Std 1516.2, 2000.

[5] Jae-Hyun Kim and Tag Gon Kim, "Federate-Level Reusability: Joining a Federation with SOM Document Data," *Proceedings of the Spring Simulation Interoperability Workshop*, 04S-SIW-141, 2004

[6] W. Cai, S.J. Turner and B.P. Gan, "Hierarchical Federations: An Architecture for Information Hiding," *Proceedings of the 15th Workshop on Parallel and Distributed Simulation*, pp. 67-74, 2001.

[7] M. D. Myjak and S. T. Sharp, "Implementations of Hierarchical Federations," *Proceedings of the Fall Simulation Interoperability Workshop*, 99F-SIW-180, 1999.

[8] A. Cramp, J. Best, and M. Oudshoorn, "Time Management in Hierarchical Federation Communities," *Proceedings of the Fall Simulation Interoperability Workshop*, 02F-SIW-031, 2002

[9] K. Aoyama, S. Ninomiya, Y. Takeuchi, S. Miyajima, and A. Tsutai, "Hierarchical Multi-Federation Structure of the Sensor Data Fusion Simulation in JUSA," *Proceedings of the Fall Simulation Interoperability Workshop*, 98F-SIW-045, 1998

[10] J. Dingel, D. Garlan, and C. Damon, "Bridging the HLA: Problems and Solutions," *Proceedings of the 6th IEEE International Workshop on Distributed Simulation and Real-Time Application*, 2002

[11] L. Granowetter, "RTI Interoperability Issues – API Standards, Wire Standards, and RTI Bridges," *Proceedings of the European Simulation Interoperability Workshop*, 03E-SIW-077, 2003.