Federate-Level Reusability: Joining a Federation with SOM Document Data

Jae-Hyun Kim
Tag Gon Kim
Systems Modeling Simulation Laboratory,
Korea Advanced Institute of Science and Technology (KAIST)
373-1 Kusung-Dong, Yusung-Ku, Daejeon, Korea
+82-41-869-5454
jhkim@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr

Keywords:
HLA, RTI, FOM, FDD, SOM, SDD, Federate-Level Reusability

ABSTRACT: *The current HLA specifies that all federates need to have the same copies of FOM Document Data(FDD) designated while creating a federation. This scheme simplifies the initialization phase of the federate since FDD has all required data. This restriction, however, has some problems. An information hiding problem is occurred when two or many federations from different organizations form a unified federation. The organizations need to expose all data within their federation even if they do not want to expose. Another problem happens when the FOM is modified. All federates in a federation ought to modify their FDD files although the modification is not directly related to their object model. Finally, a federate should maintain a set of FDD files of every federation with which it joins.*

This paper proposes a new scheme that eradicates these problems; a federate joins a federation with its SOM Document Data (SDD) file instead of FDD. Like a FDD file, SDD contains SOM data required at run-time. A federate and its SDD become a unit of reuse, since SDD is in fact an interface to the federate. This set, i.e., the federate program and its interface description, is able to join any federation without modification. Since it is unnecessary to know other data that is not related to the federate, the information hiding problem and other FOM maintenance problems will be disappeared. In order to support this scheme, the paper proposes modification to Join Federation Execution service and associated algorithms that process the join service with SDD.

1. Introduction

The component-based development methodology in software engineering has been widely applied in various software program companies. Using the methodology, a program is quickly assembled with various off-the-shelf components of proper cost. The program interacts with the components using their interfaces. A component and its interface are provided to the developers in the form of binary code as well as documents.

There is an analogy between distributed simulation based on the High Level Architecture (HLA) [1-3] and component-based development methodology.

The HLA is the specification for interoperation among distributed heterogeneous simulations. One of the major goals for the HLA is to increase reusability of simulators.

The unit of simulation execution in the HLA is a federation. A federation consists of several federates. Each federate interact and communicate with each other through RTI.

If a federation is considered as a software program, a federate can be treated as a component. A Simulation Object Model (SOM) of the federate performs as an interface to the federate because the SOM contains the information that which data types to exchange with RTI.

In the component-based development methodology, the unit of reusability is a binary component and its interface. Similarly, a federate and its SOM is the unit of reusability

in the HLA.

However, the HLA does not uses SOM information directly while executing a federation. Instead, the HLA requires Federation Object Model (FOM) of the federation and FOM Document Data (FDD) file.

This semantics is likely to cause some limitations. One of the most critical issues is an information hiding problem [4]. This problem is occurred when two or more organizations try to build a bigger federation by merging federations under their control. Not only information needs to exchange within federations but also internal information that organizations want to keep is needed to be exposed in order to build one whole federation.

In addition, there are maintenance problems. Because a federate needs different FOM/FDD to join different federations, a federate is required to have FOM/FDD. A single modification to the federate may trigger re-organization of all FOMs in which the federate participates. In contrast, a modification to FOM is possibly to affect unnecessary re-distribution of the FOM even if the change is not directly related to the federate.

The current semantics may restrict free reuse of federates that are available in the simulator-base. This paper proposes a new semantics to encourage reuse of the federate and eliminate information hiding problem.

2. Current FOM/FDD Semantics

2.1 FED/FDD

The FED and FDD are basically same. The FDD is a text file that contains data and information in a FOM document, as defined in IEEE Std 1516. The FED is a name of the same file under DMSO HLA v1.3 specification.

The FDD file is used by the Create Federation Execution service to initialize a federation. The Create Federation Execution service requires the FDD file designator as an argument. If the federate fails to locate the file or to parse the content, the service throws exceptions.

The format of the FDD/FED file may differ among various RTI. However, the file needs to contain at least object class, interaction class, attribute, parameter, dimension, transportation type, switches tables of the FOM document.

2.2 Semantics

The *Create Federation Execution*, *Join Federation Execution* services and their semantics demonstrate a little difference between DMSO HLA v1.3 specification and IEEE 1516 Standard.

Although IEEE 1516 is the revised version of HLA v1.3, HLA v1.3 and its implementation, the RTI 1.3 NG, are still popular. Therefore, we deal with both specifications.

The *Create Federation Execution* service in both specifications should be supplied with FED/FDD file designator. This shows that a federate that is responsible to create a federation should keep the FED/FDD file in the same computer.

The *Join Federation Execution* service and joining process show a little difference. In the DMSO HLA v1.3, the joining federate is notified with FED file name and it searches the file. During the joining process, the federate initializes with the data obtained from the FED file. If the federate has failed to find the FED file or to parse the content, the *Join Federation Execution* service throws appropriate exceptions. In other words, all federates that join to the federation should keep the same FED file as the file used to create the federation.

In the IEEE 1516, *Join Federation Execution* service no longer throws exceptions related to FDD files. The joining process does not require local FDD file. The initialization data extracted from FDD file may be transferred from the federate that create the federation. The algorithm used to create and initialize the federate may differ from RTI venders to venders.

The federate that is responsible to create a federation should keep FDD file in the same computer, although federates that only joins to the federation are not necessary to do so. Nevertheless, based on the experience of developing federates, it is common to call *Join Federation Execution* service after *Create Federation Execution* services in all federates. Because not all

federates participate in the simulation all the time during test phase, and any federate that starts first, should create the federation. Therefore many federates are designed to call both services for convenience. In summary, FDD file is normally distributed to all federates in many cases.

2.3 Issues

The requirement of FOM/FDD to execute a federation causes some problems. One of the most critical problems of current semantics is information hiding problem.

Information hiding problem is about security. During the process of merging two or more federations together, unnecessary information for interoperation – such as object/interaction class structures, attribute/parameters that brings secure data only for internal use – must be exposed to build a merged FOM. This is a quite critical problem when two or more nations are involved in military applications.

Recently, many solutions like RTI Gateway/Bridge [5] and Hierarchical Federation[6] have been proposed. These approaches are suggesting special federates or architecture that manages information flow among two or more federations interoperating.

However, limitation to access RTI internal data makes it hard to support all HLA services in bridged architecture [6].

Beside the information hiding problem, some maintenance problems happen.

In the HLA v1.3 specification, all federate programs should keep the same copy of FED file that is used to create the federation. Even in the IEEE 1516, many federate programs are keeping FED files for convenience of testing. During test phase, frequent modification of FOM may occur. One change in the FOM/FDD makes all other copies of the file replaced, even if the change is irreverent to most federates.

A collection of simulators may be called simulator-base like data-base or model-base. This simulator-base is a very useful tool that encourages reusability. This library should keep not only federate code and its SOM document but also FED files.

3. Join with SOM Document Data

This chapter introduces a new semantics for creating and joining federation that eliminates the problems listed in the chapter 2.3.

3.1 SOM Document Data

Why the current HLA uses FOM/FDD for initialization? Because FDD contains all information required to initialize a federation, and direct access to FDD is very simple and easy to implementation.

Recall the chapter 2.1. FDD should contain at least the following information [2].

- Object class structure table
- Interaction class structure table
- Attribute table (transportation, order, and available dimensions columns only)
- Parameter table(transportation, order, and available dimensions columns only)
- Dimension table
- Transportation type table
- Switches table

FDD has all object/interaction, dimension, transportation and all other associated information that all federates uses inside the federation. Therefore, it is extremely simple to assign globally unique handles to each object, interaction classes, attributes, and parameters, etc. It also knows the exact size of the each data structure that represents object and interaction class structures. The HLA service does not allow for federates to use other types not defined in the FDD.

Except for convenience purpose, FDD is unnecessary. There is another way to get all the information the federation needs. In other words, it is possible to retrieve information and construct FDD in other ways, not manually by developers.

Let’s introduce the SOM Document Data (SDD). The FOM and SOM have the same format of Object Model Template (OMT) [3]. The SDD is that data and information in a SOM documents, in the same format as FDD. The FDD represents all data types used in the

federation, while the SDD represents all data types used in the corresponding federate. Therefore, union of all data from SDDs of all federates will construct the same data structure that is retrieved from the FDD.

Each federate joins to the federation with its own SDD file. SDD only contains object/interaction classes that the federate publishes and subscribes. SDD only contains the dimensions and transportation types that the federate uses.

When another federate tries to join the federation, its SDD data will be merged with the current data structure that is managed by RTI. After all federates has finished to join the federation, RTI will have the same global data structure as that from FDD file.

The previous FDD is no longer necessary. Or the FDD will be supplied to the Create Federation Execution service with different meaning. FDD contains only data common to all federates – such as Management Object Model (MOM). It will be a good start point for further merging.

3.2 Join Federation Execution service

To support the SDD mechanism, Join Federation Execution service should be revised. The new Join Federation Execution service is suggested as follows:

Join Federation Execution

1. Supplied arguments
   (a) Federate type.
   (b) Federation execution name
   (c) SOM document designator
2. Return arguments
   (a) Joined federate designator.
3. Preconditions
   (a) The federation execution exists.
   (b) The federate is not joined to that federation execution.
   (c) Save not in progress.
   (d) Restore not in progress.
4. Postconditions
   (a) The joined federate is a member of the federation execution.
5. Exceptions
   (a) The federate is already joined to the federation execution.
   (b) The specified federation execution does not exist.
   (c) Could not locate SOM document indicated by supplied designator.
   (d) Invalid SOM document.
   (e) Inconsistent Object Model.
   (f) Save in progress.
   (g) Restore in progress.
   (h) RTI internal error.

One argument and associated exceptions are added. The SOM document designator indicates the SDD file path and name.

The *Join Federation Execution* service locates the supplied SDD file. If it has failed to find the file, the service throws the exception – *Could not locate SOM document indicated by supplied designator*.

After the service successfully loaded the SDD, it parses its contents. The SDD has the same format as FDD, the developers are able to reuse the parsing routine. If the SDD has invalid format, the service throws *Invalid SOM document* exception.

The SDD data structure will be transferred to RTI after parsing. RTI will add new information to its current data structure. If merging process fails, the service will throw *Inconsistent Object Model* exception.

3.3 Merging Process

The merging process of the SDD plays the key role in the SDD joining mechanism. This chapter shows each merging process of object model tables and its exceptional cases.

Following examples are taken from [3]. Let us assume that a *restaurant* federation consist of two federates – *hall* and *kitchen* federates. Table 1,Table 2, and Table 3 show the object class structure tables of each federate and the merged result. The number in parentheses means globally unique object class handles.

Table 1 Hall Federate Object Class Table

HLA Object Root (1)	Customer (2)		
	Bill (3)		
	Order (4)		
	Employee (5)	Greeter (6)	
		Waiter (7)	
		Cashier (8)	
	Food (9)	MainCourse (10)	
		Dirnk (11)	Soda (12)
		Soup (13)	Today's Soup (14)
		Entree (15)	Beef (16)

Table 2 Kitchen Federate Object Class Table

HLA Object Root	Order			
	Employee	Dishwasher		
		Cook		
	Food	MainCourse		
		Dirnk	Soda	
		Soup	Today's Soup	ClamChowder
				BeefBarley
		Entree	Beef	
			Chicken	
			Pasta	

Table 3 Merged Object Class Table

HLA Object Root (1)	Customer (2)			
	Bill (3)			
	Order (4)			
	Employee (5)	Greeter (6)		
		Waiter (7)		
		Cashier (8)		
		Dishwasher (17)		
		Cook (18)		
	Food (9)	MainCourse (10)		
		Dirnk (11)	Soda (12)	
		Soup (13)	Today's Soup (14)	***ClamChowder (19)***
				BeefBarley (20)
		Entree (15)	Beef (16)	
			Chicken (21)	
			Pasta (22)	

There is an empty table right after a federation is created. If *hall* federate joins to the federation first, its table becomes the initial table and object class handles are assigned. The algorithm to assign object class handles may differ from vendor to vendor.

When *kitchen* federate tries to join, RTI merges two object class tables. Beginning with the same root class – HLAobjectRoot – RTI adds newly found classes to the table. There is no exceptional case in merging process for object class structure table. After the merging is complete, RTI assigns object class handles to the newly added classes.

The merging of interaction class structure table is identical with that of object class structure.

The merging of attribute table is basically same as that of object/interaction class structure. During merging process, RTI adds newly found items to the table. If RTI detects that an attribute is already defined in the internal table, it checks other columns for consistency.

The rule of thumb for checking consistency is that the option is whether federation-wide basis or per-federation basis. If an option works as federation-wide basis, the option should be consistent through all joining federates. If not, the option is considered as the specific setting for that federate.

The options from (a) to (d) in the Table 6 show some cases that different options are set in SDD files.

(a) Conflict in Transportation: Transportation must be consistent in the federation, because federates are not able to communicate with different transportation type.
(b) Conflict in Order: Order is federation-wide basis. Although a federate can treat a time-stamped order message as a receive order message, the opposite case is impossible.
(c) Conflict in Dimension: The default dimension (noted by NA) is treated as the option is not specified. Therefore (c) case is not a conflict.
(d) Conflict in Dimension: If two different dimensions are specified, then RTI uses both dimensions. Therefore (d) case is not a conflict.

In summary, conflict in transportation and order column causes an exception, while different values in dimension column are added in the table.

The merging process of parameter table is similar with that of attribute table. However, there is one big difference that merging of parameter table does not allow new parameters in previously defined interaction class. An interaction is sent and received including all parameters defined in the table. Therefore two federates are not able to interact with interactions containing different set of parameters. If RTI detects new parameter for previously defined interaction class, it throws the exception.

The dimension table and transportation type table are simple listing of items. The merging processes are simply to add new items in the internal table.

For switch tables, the checking algorithm differs from item to item. *Auto Provide* and *Convey Region Designator Sets* switches are controlled on a federation-wide basis. Therefore mismatches of the switch setting causes the exception. Other advisory switches and *Service Reporting* switch are per-federate basis. RTI does not check these switch settings for consistency.

3.4 Pros and Cons

The proposed SDD semantics eliminate central integration of data and information. A federate is able to join any federation as long as the object model is consistent. There is no need to re-organize or re-distribute FOM/FDD when the composition of the federation has changed.

The unit of reusability becomes the federate program and its SDD file. Without modification, this set is able to join a federation. Information hiding problem disappears naturally, because there is no explicit data exposure. Organizations are able to exchange only data related to interoperation.

In addition, this semantics is backward compatible. If a federation is created with the FDD containing full data of the federation and no SDD is supplied with *Join Federation Execution* service, RTI will work exactly the same as current one.

Table 4 Hall Federate Attribute Table

Object	Attribute	Available Dimensions	Transportation	Order
HLAobject Root	HLAprivilege ToDeleteObject	NA	HLAreliable	TimeStamp
Employee	PayRate	NA	HLAreliable	TimeStamp
	YearsOfService	NA	HLAreliable	TimeStamp
Employee.Waiter	Efficiency	NA	HLAreliable	TimeStamp
	Cheerfulness	NA	HLAreliable	TimeStamp
Food.Drink	Number Cups	BarQuantity	HLAreliable	TimeStamp
Food.Soda	Flavor	SodaFlavor, BarQuantity	HLAreliable	TimeStamp

Table 5 Kitchen Federate Attribute Table

Object	Attribute	Available Dimensions	Transportation	Order
HLAobject Root	HLAprivilege ToDeleteObject	NA	HLAreliable	TimeStamp
Employee	PayRate	NA	HLAbestEffort	TimeStamp
	YearsOfService	NA	HLAreliable	Receive
	HomeNumber	NA	HLAreliable	TimeStamp
Employee.Cooker	Skill	NA	HLAreliable	TimeStamp
Food.Drink	Number Cups	NA	HLAreliable	TimeStamp
Food.Soda	Flavor	SodaType	HLAreliable	TimeStamp

Table 6 Merged Attribute Table

Object	Attribute	Available Dimensions	Transportation	Order
HLAobject Root	HLAprivilege ToDeleteObject	NA	HLAreliable	TimeStamp
Employee	PayRate	NA	**(a) HLAreliable /HLAbestEffort**	TimeStamp
	YearsOfService	NA	HLAreliable	**(b)TimeStamp /Receive**
	HomeNumber	***NA***	***HLAreliable***	***TimeStamp***
Employee.Waiter	Efficiency	NA	HLAreliable	TimeStamp
	Cheerfulness	NA	HLAreliable	TimeStamp
Employee.Cooker	***Skill***	***NA***	***HLAreliable***	***TimeStamp***
Food.Drink	Number Cups	**(c) BarQuantity /NA**	HLAreliable	TimeStamp
Food.Soda	Flavor	**(d) SodaFlavor, BarQuantity, SodaType**	HLAreliable	TimeStamp

The fact that the HLA specification should be modified to support the proposed semantics is one big demerit. We hope that RTI developers will support modified Join Federation Execution service, because it is very simple and has good benefits. Currently we are implementing Runtime Infrastructure with this feature.

4. Conclusion

We have described new HLA semantics related to create and join federation execution. Current semantics using FOM/FDD causes information hiding and maintenance problems so that it lowers the reusability of a federate.

A federation consists of a set of federates. The FOM contains all data and information used in a federation, while the SOM only describes data the federate is interested in. The data in the FOM document is union of data from SOMs of joining federates. In this point of view, a new semantic that a federate join the federation with SDD is proposed. Data that is required to initialize the federation is assembled and merged from SDD that joining federate supplies.

A federate and its SDD can form a plug-and-play style federation. The federate can join any federation, and its FOM is constructed automatically and internally by RTI. An information hiding problem and maintenance problems related to FOM/FDD are eliminated.

Although the HLA specification should be modified to support this feature, the proposed semantics is simple to implement and has good benefits. We hope many RTI developers adopt this feature.

5. References

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

[2] IEEE: "IEEE Standard for Modeling andSimulation (M&S) High LevelArchitecture (HLA).FederateIEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification," Std 1516.1, 2000.

[3] IEEE: "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification," Std 1516.2, 2000.

[4] 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.

[5] M.G. LI, W. CAI, B. Francis, M.L. LIU and D.S. TURNER: "An Event Based Design of a Gateway Federate" Proceedings of the 2003 Spring Simulation Interoperability Workshop, 03S-SIW-046, 2003.

[6] B. Breholee and P. Siron: "Design and Implementation of a HLA Inter-federation Bridge" 03E-SIW-054, 2003.

Author Biographies

JAE-HYUN KIM received a B.S. and M.S. in electrical engineering in 1998 and 2000, respectively, from Korea Advanced Institute of Science and Technology(KAIST), Daejeon, Korea. Currently, he is a Ph D. student at the Department of Electrical Engineering and Computer Science, KAIST. His research interests include methodology for modeling and simulation of discrete event systems and high level architecture.

TAG GON KIM received his Ph.D in computer engineering with specialization in methodology for systems modeling/simulation from University of Arizona, Tucson, AZ, 1988. He was a Full-time Instructor at Communication Engineering Department of Bukyung National University, Busan, Korea between 1980 and 1983, and an Assistant Professor at Electrical and Computer Engineering Department at University of Kansas, Lawrence, Kansas, U.S.A. from 1989 to 1991. He joined at Electrical Engineering Department of KAIST, Daejeon, Korea in Fall, 1991 as an Assistant Professor and has been a Full Professor since Fall, 1998. His research interests include methodological aspects of systems modeling simulation, analysis of computer/communication networks, and development of simulation environments. He is a senior member of IEEE and SCS and a member of ACM and Eta Kappa Nu.