High-Level Architecture service management for the interoperation of federations

Min-Wook Yoo¹, Changbeom Choi² and Tag Gon Kim¹,*

Abstract
In High-Level Architecture, federates are integrated into a federation in single Run-Time Infrastructure middleware, which handles High-Level Architecture services for interoperation. The interoperation of federations is required in order to combine systems simulated in existing federations. Unlike a single federation, the interoperation of federations provides the data security and makes it possible to interoperate federates developed in different Run-Time Infrastructures. Previous research has used a proxy method to interoperate federations without modifying the Run-Time Infrastructure. Based on the proxy method, this paper proposes a form of High-Level Architecture service management for the interoperation of federations. Federations construct a two-level hierarchical federation that uses proxies to interoperate different levels and that defines the data structure for interoperation and data security. For each High-Level Architecture service protocol, we propose the algorithms of the proxy and show how the service protocol works in the two-level hierarchical federations. In addition, the proxy handles the race condition, which may occur in interoperation of federations, and describes how to solve it. The algorithms are verified theoretically, and a case study shows the practical application for the interoperation of federations.

Keywords
Interoperation of federations, hierarchical federation, High-Level Architecture/Run-Time Infrastructure, proxy

1. Introduction
Simulation is an important tool and is used to resolve problems that are difficult to solve in various domains. In particular, the military invests a large amount of money in simulation and uses it in personnel training, the designing and testing of equipment, and analysis of both past and future actions.¹ Given the number and complexity of these military simulations, developers should provide simulation software that meets stakeholders’ requirements, with characteristics such as being high-fidelity, real-time, physically distributed, and supportive of mission-critical simulation domain, while minimizing redundancy of effort across applications and maximizing the flexibility of the software to be used for new, possibly unanticipated tasks.² Under these conditions, the Defense Modeling and Simulation Office (DMSO) of the United States first proposed High-Level Architecture (HLA) for simulations and developed the Run-Time Infrastructure (RTI) software prototype in 1999. HLA was defined under IEEE Standard 1516 in 2000 and revised in 2010.³–⁶

HLA is architecture for the interoperation of simulations so that the developer can perform a simulation with heterogeneous simulators. An HLA-based simulation is called a federation, which consists of multiple HLA-compliant applications called federates. The purpose of HLA is to support the coordination of different simulations with the goal of simplifying the integration task, encouraging modularization of simulations, and increasing quality of simulations and potential for reuse. In order to handle data exchanges among simulation applications, the HLA requires the object models for federations and federates. A federation is made with a Federation Object Model (FOM), which includes shared objects among joined federates, and each federate has a Simulation Object Model

¹Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea
²School of Creative Convergence Education, Handong Global University, Republic of Korea
*SCS member

Corresponding author:
Min-Wook Yoo, Department of Electrical Engineering, KAIST 291 Daehak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea.
Email: youmw@kaist.ac.kr
(SOM), which describes the shared objects used for a single federate. In general, a FOM is made from the SOM of joined federates, and the only data of objects included in the FOM can be exchanged among federates. On the other hand, all federates can acquire the object information included in the FOM if they want to receive it.

The RTI is a middleware that provides a set of software services that are necessary to support federates to coordinate their operation and data exchange during simulation based on the FOM. In order to support federate development, the RTI provides a programming library and an Application Programming Interface (API) based on the interface specification of the HLA. Therefore, when a developer creates HLA-compliant simulators, the RTI handles the service requests from the federates to interoperate with other federates. Unlike HLA, the RTI is not a standard, but an implementation of the HLA interface specification. As a result, RTI products from different vendors or different versions such as HLA 1.3 or IEEE 1516 are not operable with each other.

Owing to the characteristics of the HLA/RTI, large systems in various domains have been modeled as a system of systems and have been simulated as a federation. In other words, a large system can be modeled as a collection of several federates, which comprise a federation. Similarly, a federation may be a component of the bigger system, and reusing a federation as a component of large scale simulation is a challenging problem. For example, as illustrated in Figure 1, there are federations developed in different countries and they operate military training of their countries independently. In some cases, military trainings for combined forces of several countries are required. In this case, a new system, which consists of previous systems, should be made from existing federations.

In order to utilize a federation as a component of bigger simulation, a developer has two options: integration of federations and interoperation of federations. As shown in Figure 2, integrating federations requires disassembling each federation into federates and constructing a new federation. The integration of federations poses two problems. First, data security problems may occur among federates in the military domain. To interoperate with other federates, existing federations should combine their FOMs to make a superset FOM for the federations. As all federates can access the data of FOM, security could be considered compromised with all of the data confederated in a single FOM. In other words, a malicious federate can subscribe security objects to hijack the contents of the objects. In this case, object models and federates should be redesigned to have security mechanisms to protect the information; however, this will hinder the reusability of the federation. The other problem occurs when two federations are made in different RTIs. Ideally, a single federation is a sufficient environment for interoperation, and federates that are developed with HLA interfaces can be reused in any federation. However, in reality, federates of the integrated federation should be created based on identical development environments. In other words, federates that are developed in a certain RTI environment may not join the federation developed in a different RTI environment. Despite the Dynamic Link Compatible HLA API Standard, federations may be developed in several HLA standards – HLA 1.3, IEEE 1516-2000, IEEE 1516-2010 – and there have been various versions of RTIs for each.
In addition, each RTI vendor provides additional libraries to developers, so that a federate is developed with specific libraries from the RTI vendor: it cannot join a federation that was created in another RTI.

Interoperation of federations may solve aforementioned problems. As shown in Figure 3, each federation is separate and maintains its own FOMs during the interoperation. As a result, each federation can access the common objects of their FOMs, whereas it cannot access the other object of other federation. In addition, the interoperation of federations allows each federation to be created in different RTI environments.

Interoperation of federations can be accomplished by various methods. However, interoperation of federations must not violate or alter HLA standards. Most previous research has achieved interoperation of federations by
altering the standards or handling partial HLA services to support limited services to the federations in their applications. The point of this study is, therefore, to propose an environment that supports the interoperation of federations without altering HLA standards handling the entirety of HLA services.

This paper proposes federation architecture, algorithms, and HLA service protocols for the interoperation of federations. In the proposed architecture, a two-level hierarchical federation is chosen to minimize performance loss, and a proxy is utilized to interoperate federations of different levels. The structure of the federation consists of two levels. The federation that contains simulators is placed in the lower level, and the federation in the upper level manages the interoperation of the federations. The proposed architecture manages the data security of the confederation while reusing the existing data models of each joined federation.

Based on the architecture, we propose algorithms for the proxy, which consists of two agents. During the Federation Synchronization and the Unanimity protocol, a race condition may occur. Since the problem cannot occur in the single federation, we propose algorithms to solve the race condition problem in our protocols.

To provide HLA service protocols for confederation, we first classify HLA services into five distinct protocol types. Since HLA defines over 100 services, we show how the service types represent the HLA services, and how they work in the confederation architecture.

Moreover, the proposed algorithms are verified theoretically. For each service protocol, the results of the two environments — a single federation and a two-level hierarchical federation — are identical when a federate requests a HLA service. To show a practical application, a case study is performed in a two-level hierarchical federation. In the case study, an air-defense system is simulated in federations of different RTIs.

Because a proxy represents all federates of a federation, simulation performance can be decreased when the number of federates increases. Simulation performance of that case should be studied to apply the proposed work to practical applications. However, this paper focused on the correctness of HLA service management in a two-level hierarchical federation and we will handle the issue in future works.

This paper is organized as follows. Section 2 describes related works for the interoperation of HLA federations and Section 3 classifies the HLA service protocols in a federation. Section 4 defines the proposed architecture for interoperation of federations and presents the data structure. It also shows the algorithms of the proxy and the service protocols in the interoperation of federations. Section 5 presents verification of the algorithms and a case study in a two-level hierarchical federation. Section 6 concludes the paper.

2. Related work

RTI vendors partially support the interoperation of other RTIs. VT MÁK (Cambridge, MA, USA) provides the network toolkit for Distributed Interactive Simulation (DIS) and HLA, which includes previous HLA standards such as HLA 1.3 and IEEE 1516-2000. Raytheon (Waltham, MA, USA) supports the interoperation of federates of HLA 1.3 and IEEE 1516. The federates of HLA 1.3 can join the federation created by the federates of IEEE 1516. Pitch Technologies (Linköping, Sweden) provides the library, which allows IEEE 1516-2000 federates to be able to use in IEEE 1516-2010. However, the interoperation is restricted to RTIs of the same vendors or provides limited services.

The interoperation of federations arose in the late 1990s. The basic methods for interoperation were summarized by Myjak in 1999. Federation gateway provides access between federates of different federations. It requires special federates to be developed while considering the other federations. Federation proxy uses a proxy to represent the behavior of a federation. The proxy does not simulate a system and only delivers the behavior of a federation. Federation broker and protocol require RTIs to support the interoperation of federations. IEEE 1516-2010, the up-to-date standard of HLA, does not specify the function of federation broker or protocol. These methods can be applied only for the extended RTI. A hierarchical RTI was proposed and implemented to support the interoperation by RTI. The research extended the interfaces or FOMs in order to construct the hierarchical structures.

In a general federation environment, a federate considers only the other federates of the same federation and a federation is linked with a single RTI. Therefore, the federation gateway and the extension of RTI are hard to apply to the current environments, but the proxy method can be applied to those environments because it does not require the modification of the RTI and the federates. However, RTI treats a proxy like a ‘normal’ federate and does not provide additional services for it. Because of this treatment, problems can occur when the proxy represents behaviors of a federation.

Problems of the proxy method are summarized by Dingel et al. The problems occur because a proxy cannot acquire sufficient information of the joined federation. Previous research has proposed how to acquire the required information. RTI can provide additional interfaces for the proxy to acquire the necessary information, but it encounters the same problems of federation broker and protocol. The additional interfaces cannot be applied to existing environments. The Management Object Model (MOM) service can provide the required information to the proxy. RTI provides the service invocation of the federate using the MOM service. Though each federate initiates its
state to report service invocation, the initiation may be
accomplished with RTI Initialization Data (RID) files to
provide all information needed to solve the problems.
However, the MOM service provides the reports of all ser-
vice and cannot report partial services. This causes perfor-
mance loss in reporting federates and a proxy, though the
proxy requires much less information. To reduce perfor-
mance loss, the proxy can acquire information through
other methods. Federates can report service invocation
through interactions21 or Agent–User Protocol.36
Reporting through interactions requires the modification
of existing FOMs, and Agent–User Protocols can be applied
with modification of FOMs. These methods require feder-
ates to be modified, but the modifications are limited to
only partial services. Two or more solutions can be applied
at once. In the same federation, heavy-loaded federates can
use Agent–User protocol to reduce performance loss, and
light-loaded federates can use MOM services as well. Our
work assumes that problems of the proxy are solved by
those solutions.

A proxy is used to represent a federation. To interope-
rate three or more federations, the architecture for proxies
and federations should be defined. There is no limitation
for the architecture,14 but cyclic architecture is not recom-
ended because it may cause the duplication and circula-
tion of the service. Hierarchical structure is a traditional
structure for composing a system of systems. Many previ-
ous research studies have been constructed on the hier-
archical federation. Several studies have handled the HLA
services in hierarchical federation,21,22,37 and others have
studied the shared data among federations.17,18 Grid ser-
ices have also been applied for multiple federations.38–41
Proxies participate in the grids and interact with others
using grid services, providing advantages for management
of distributed resources.

3. HLA service protocols

IEEE 1516.1 defines the HLA functional interfaces
between an RTI and federates. Federates request services
to an RTI, and the RTI provides callback services using
the interfaces. The specification describes the behaviors of
the federate and RTI when the services are requested.
There are more than 100 service interfaces, and the ser-
vice are provided by RTI according to their specific prot-
cocols. Each interface has various names or arguments, but
their service patterns can be classified into much smaller
types. We classify the service protocols into five
types. Service protocols are classified with service patterns
and each service protocol does not affect the others
directly.

Table 1 classifies the HLA services in accordance with
the protocol types. The related interfaces are described by
the numbers, which are recorded in IEEE 1516.1-2010.5

<table>
<thead>
<tr>
<th>Protocol type</th>
<th>Services</th>
<th>Related interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolation</td>
<td>Federation initialization/termination</td>
<td>4.2<del>4.6, 4.9</del>4.10</td>
</tr>
<tr>
<td></td>
<td>Publication/subscription change</td>
<td>5.2<del>5.9, 6.16, 9.8</del>9.11</td>
</tr>
<tr>
<td></td>
<td>Information query</td>
<td>4.7<del>4.8, 4.22</del>4.23, 4.31<del>4.32, 6.25</del>6.26, 6.29<del>6.30, 7.17</del>7.19, 8.16<del>8.18, 8.20, 10.2</del>10.30</td>
</tr>
<tr>
<td></td>
<td>Object state change</td>
<td>6.2<del>6.7, 6.23</del>6.24, 6.27<del>6.28, 9.6</del>9.7</td>
</tr>
<tr>
<td></td>
<td>Time state change</td>
<td>8.2<del>8.24, 8.14</del>8.15, 8.19</td>
</tr>
<tr>
<td></td>
<td>Region management</td>
<td>9.2~9.4</td>
</tr>
<tr>
<td></td>
<td>Supporting services</td>
<td>10.31~10.44</td>
</tr>
<tr>
<td></td>
<td>Object switch notice</td>
<td>5.10<del>5.13, 6.17</del>6.18, 6.21~6.22</td>
</tr>
<tr>
<td></td>
<td>Object update</td>
<td>6.10<del>6.11, 6.19</del>6.20</td>
</tr>
<tr>
<td></td>
<td>Interaction delivery</td>
<td>6.12~6.13</td>
</tr>
<tr>
<td></td>
<td>Ownership transfer</td>
<td>7.2~7.16</td>
</tr>
<tr>
<td></td>
<td>Time advance request/grant</td>
<td>8.8<del>8.13, 8.21</del>8.22</td>
</tr>
<tr>
<td></td>
<td>Federation synchronization by a point</td>
<td>4.11~4.15</td>
</tr>
<tr>
<td></td>
<td>Unanimity</td>
<td>Federation Save 4.16<del>4.21, Federation Restore 4.24</del>4.30</td>
</tr>
</tbody>
</table>

3.1. Isolation protocol

Figure 4 shows the Isolation protocol type. In this
protocol, a federate requests a service and the RTI may or
may not respond. When the response or the grant is
required, the RTI provides the callback service to the
requesting federate. Otherwise, the service request is
applied immediately and the RTI does not provide the
callback service. Compared with other types, the service is
processed only between the requesting federate and the
RTI. The service request of a federate does not affect
the other federates directly, whether RTI provides the callback
or not.
3.2. Information Routing protocol

Figure 5 shows the Information Routing protocol type. In this protocol, the RTI delivers information to a part of federates when a federate has sent it to the RTI. Requesting federates do not specify the receiving federates, which are determined by the federates’ subscription states. All services of this protocol are related with shared objects. Each federate initiates and changes the publication/subscription states by the declaration management services, which belong to the Isolation protocol. The RTI routes the information to the federates using the states of federates.

3.3. Time Synchronization protocol

Figure 6 shows the Time Synchronization protocol type. Federates request time advances and the RTI grants them. Each federate cannot request a new time advance until the RTI grants the previous request. The RTI grants a time advance of federates when it is guaranteed that no messages of a smaller time stamp will arrive later. For time synchronization, each federate defines its time states related with the exchange of Time-Stamp Order (TSO) messages. Each federate has a limit time that is determined by the other federates, and it cannot advance the time past the limit. RTI grants the time advance that is faster than the limit time. Because the limit is affected by the other federates, it may be changed when a federate requests a time advance. When a federate requests a time advance, the RTI checks the limits and requested times of all federates and determine whether to grant it. The grant of a federate does not affect the other federates and two or more federates can be granted at once.

3.4. Federation Synchronization protocol

Figure 7 shows the Federation Synchronization protocol type. Synchronization is accomplished by the synchronization point. Federates define the point related with the specific job or state before constructing a federation. Synchronization service begins when a federate registers a point and the RTI announces it to all federates. The announced federate handles the jobs related with the point and achieves the point when it completes the job or becomes the specific state. All federates must report the achievement to the RTI after achieving the point. Failure of the achievement is not allowed. After all federates achieve the point, RTI notices that the federation is synchronized with the point. Two or more synchronization points can be processed in parallel but they should use different points. The registration of the processing point is rejected by the RTI.

3.5. Unanimity protocol

Figure 8 describes the Unanimity protocol type. This protocol is similar with Federation Synchronization. Compared with Federation Synchronization, Unanimity’s processing jobs are defined in HLA — as ‘Federation Save’ or ‘Federation Restore’— and there are two cases after the
service initiation. In the service success case, all federates report ‘Complete’, which means the service success of the federate. After all federates report ‘Complete’, RTI announces the service success to them. In the service failure case, one or more federates report ‘NotComplete’ which means the service failure of the federate. When a federate reports ‘NotComplete’, RTI announces the failure even if the other federates report ‘Complete’ or have not reported yet.

Only one service in Unanimity can be processed at once. Though jobs are requested with different labels, two or more requests cannot be processed in parallel. In addition, ‘Federation Save’ cannot be processed while ‘Federation Restore’ is processing and vice versa.

4. HLA service management for interoperation of federations

To interoperate multiple federations, a hierarchical federation can be constructed. The hierarchical structure originally does not have restrictions but we limit the hierarchy levels. Because the services may be propagated to all federations, the hierarchy levels can affect the simulation performance. We use the hierarchical structure only for interoperation and the structure is not used to represent the simulated system. Therefore, we minimize the hierarchy level.

Figure 9 shows the proposed confederation architecture. Federations construct a two-level hierarchical federation, and proxies are used to interoperate federations of different levels. The hierarchy is limited to two levels to minimize performance loss. The upper-level federation is made to interoperate the federations and is called the Agent Federation. Federations that want to interoperate are

![Figure 7. Federation Synchronization protocol type. RTI: Run-Time Infrastructure.](image)

![Figure 8. Unanimity protocol type. RTI: Run-Time Infrastructure.](image)
placed in the lower-level federations and are called user federations.

Each user federation has a proxy to interoperate with the Agent Federation. A proxy consists of two agents – AgentL and AgentU. AgentL joins a user federation and AgentU joins the Agent Federation. AgentU represents its user federation in the Agent Federation and AgentL represents the Agent Federation, which includes the information of the other user federations, in the user federation. User federations are interoperated with the others by inter-operating with the Agent Federation.

Figure 10 presents the overall behavior of a two-level hierarchical federation. In each user federation, federates simulate their system and request HLA services when they interact with the other federates. In the Agent Federation, each proxy presents behaviors of its user federation, and RTI handles the interactions of proxies.

When a federate requests a service in a user federation, RTI handles the request and the service may be propagated to the other federates through the Agent Federation. A service request of a federate is handled by following process.

**Step 1:** If the service will affect the other federations, the proxy acquires the related information using several methods like callback services. When the service does not affect the other federations, the proxy does not acquire the information, and the service is processed only in the user federation.

**Step 2:** After acquiring the information, the proxy requests the appropriate service in the Agent Federation to represent its user federation. In the internal process of the proxy, AgentL delivers the information to AgentU, and AgentU requests a service using the information.
Step 3: In the Agent Federation, RTI handles the service requests which include the behaviors of user federations. As a result of the service handling, each proxy acquires the information, which is delivered from the other federations and affects its user federation.

Step 4: Finally, each proxy requests a service in its user federation and the RTI handles the services. As a result, federates of the user federation are affected by federates of the other user federations.

4.1. Data structure

This section describes the data structure of a two-level hierarchical federation and object models for interoperability of federations. They are made from object models of existing federations; this section defines new object models from those object models. In HLA, object models are described by class and each class has attributes.

To describe object models, this section uses following operators. $A \cup B$ is the object model including the objects, which are commonly included in A and B. When the attributes of an object are different in A and B, the object of $A \cap B$ only includes the common attributes of them. $A \subseteq B$ means all objects of A and their attributes are included in B. A-B is the object model including the objects of A, which exclude attributes of B.

Figure 11 shows the data structure for a two-level hierarchical federation. Each federation has a FOM, and each proxy has a SOM. Initially, all user federations that want to interoperate have their own FOMs by HLA rules and they are used in the hierarchical federation without modification. The SOMs of proxies and a FOM of the Agent Federation will be defined for interoperation. They can be made from the FOMs of the user federations. Each SOM of a proxy is defined from its user federation and the FOM of the Agent Federation is made from the SOMs.

HLA rules describe that federates should have a SOM and can exchange the objects specified in their SOMs. In the Agent Federation, the proxy exchanges the objects that are exchangeable in its user federation. The objects are included in its FOM by HLA rules. Therefore, the SOM of a proxy should be a subset of its federation’s FOM.

$$SOM_A \subseteq FOM_A, \quad SOM_B \subseteq FOM_B, \quad SOM_C \subseteq FOM_C, \ldots$$

A proxy represents all of the other user federations in its user federation. In a user federation, the proxy exchanges the objects that are exchangeable in the other federations. The object should be included in at least one of the other FOMs.

$$SOM_A \subseteq (FOM_B \cup FOM_C \cup \ldots),$$
$$SOM_B \subseteq (FOM_A \cup FOM_C \cup \ldots),$$
$$SOM_C \subseteq (FOM_A \cup FOM_B \cup \ldots), \ldots$$

A proxy can exchange the objects that are exchangeable in both federations. The objects, which are exchangeable only in one federation, are invalid in the other federation and need not be delivered. Therefore, the SOM of a proxy should be the intersection of two sets.

$$SOM_A \subseteq FOM_A \cap (FOM_B \cup FOM_C \cup \ldots),$$
$$SOM_B \subseteq FOM_B \cap (FOM_A \cup FOM_C \cup \ldots), \ldots$$

Each user federation can define the hidden objects that are not shared with the other federations. Objects may be present that may also be included in the other federations, but a user federation does not want to share them. The hidden objects are generally related with security and are exchangeable only with federates of specific groups (groups are represented by user federations in this environment). The hidden objects should be excluded from the SOM. Finally, we can define the SOMs of proxies as the following equation with the Hidden Object Model (HOM), which includes all hidden objects.

$$SOM_A = (FOM_A - HOM_A) \cap (FOM_B \cup FOM_C \cup \ldots)$$
After all SOMs of the proxies are defined, the Agent Federation can define its FOM. The Agent Federation’s FOM should include the exchangeable objects among the proxies and the SOMs are defined to include the exchangeable objects for their proxies. Therefore, the FOM is the union of the SOMs.

\[ \text{FOM}_{\text{Agent}} = \text{SOM}_A \cup \text{SOM}_B \cup \text{SOM}_C \cup \ldots \]

4.2. Isolation protocol

In the Isolation protocol type, a service request of a federate does not affect the other federates. Most services of the Isolation type need not be represented in the other federations. The services either only affect the requesting federates or just provide the information to it. Therefore, the proxy does nothing for those service requests.

State change services change the states of the requesting federates. Although the service does not affect the other federates directly, some behaviors of other services are determined by the states. To represent a federation, a proxy’s states should represent those of its member federates. When a federate changes states, the proxy may acquire the information. However, the Isolation service does not affect the other federates and the proxy cannot know the request. The proxy can solve the problem without acquisition of the states. The proxy initiates its states to represent all cases of the federates and does not change during simulation. The states that affect the other federates, are related with the shared objects and time synchronization. We describe the states of the proxy in Information Routing and Time Synchronization as well as how to represent a federation.

4.3. Information Routing protocol

Figure 12 shows algorithms for the Information Routing protocol type. Both agents work identically although the interface names are different. When receiving the information from the RTI, the agent delivers it to the other agent. When receiving the information from the other, the agent updates the object in the joined federation. The algorithm is simple, but it works correctly only when the routings to the agents are correct.

The services of Information Routing are related with shared objects. The routing is determined by the subscription of the shared objects. The RTI provides the object information only to the federates that subscribe the object class. To represent the other federation, AgentL should subscribe the objects that are exchangeable in the other federations and AgentU should publish them in Agent Federation. To represent a user federation, AgentU should subscribe the objects that are exchangeable in its user federation and AgentL should publish them in the user federation. As described in Section 4.1, those objects are included in the SOM of the proxy by definition. Therefore, AgentL and AgentU publish and subscribe all objects of their SOM in the joined federation. Although federates can change the publication/subscription in run-time, the agents need not change the publication/subscription. When all of the federates do not publish an object, they cannot provide the object information. Because their proxy does not provide the object data by itself, it does not provide the object information in the other federation, although it publishes the object. When all of the federates do not subscribe an object, they cannot receive the object information. The RTI does not deliver the object information to them, although the proxy subscribes it, and delivers it to the user federation. There may be a waste of data traffic to the federation, but the service works correctly.

The RTI provides data of all shared objects to the proxy by its subscription, and the proxy can update the shared objects by its publication. As described in the algorithm, all received objects are delivered to the other federation. They are propagated to all federations in which the objects are exchangeable. After the delivery of the proxies, the RTIs route the information to the federates in its federation. Data filtering can be accomplished by non-subscription of the hidden objects. The SOM of the proxy excludes the hidden objects and agents do not subscribe them. As a result, the RTI does not provide the object information to AgentL and the hidden objects are not delivered to the Agent Federation.

Figure 13(a) shows the Information Routing protocols for internal objects, which are not exchangeable in the other federation or are hidden objects. AgentL does not subscribe the internal objects and the RTI does not provide the objects to it. Therefore, the service protocol is processed only in a user federation.
Figure 13(b) shows the protocols for shared objects, which are exchangeable in the other federations. AgentL receives the information for shared objects that should be delivered to the other federation. AgentL delivers the object to AgentU, and AgentU updates the object in the Agent Federation. Because AgentU subscribes the object instead of its user federation, the RTI of the Agent Federation routes the information to AgentUs, which includes the updated objects in its SOM. Though the figure shows single delivery to a user federation, the information may be delivered to multiple user federations in which updated objects are exchangeable. Finally, all AgentLs update the object in their user federations and the RTIs route it to the federates.

4.4. Time Synchronization protocol

The states related with time synchronization are time regulating and constrained. Regulating federates can generate TSO messages and regulate the time advance of the other federates. They guarantee that they will not generate TSO messages until the specific time. The time is determined by their logical time and lookahead. Constrained federates can receive TSO messages with causal order and their time advances are regulated by regulating federates. They can advance to the guaranteed time that no TSO message is generated. The time is determined by the guaranteeing times of the regulating federates. Two states are independent. Regulating federates can be constrained and constrained federates can be regulating.

An agent should determine its time states to represent a federation. Because non-regulating and non-constrained federates do not affect the other federates, the agent only considers the time-regulating or constrained federates. The agent should regulate the other federate instead of its federation. A time-constrained agent does not affect the time advance of the other federates but is related with the delivery of TSO messages. A non-constrained agent cannot receive the TSO messages with a time stamp order and cannot deliver the message with the order. If there are a time-regulating (constrained) federate, the representing agent should be time regulating (constrained). Federates can change their states in run-time and the agents may change their states. However, there is no service that provides the time regulation/constrained states of the other federates. Therefore, an agent is always time regulating and constrained even if all federates are non-regulating and non-constrained. When all federates are non-regulating, the agent guarantees that it will not generate TSO messages until time infinity—same behavior with non-regulating. Although all federates are non-constrained, the agent delivers TSO messages with time stamp order. The deliveries to the federates are determined by the federates, and they are not affected, whether the agent is constrained or not.

Based on the states, the agent requests a time advance to represent a federation for time synchronization. The agent regulates the federates instead of the federation. Especially, the agent sets its guaranteeing time by requesting a time advance. Requesting Time Advance Request Available (TARA) to the specific time means that it will not generate TSO messages until the time. Because the agent does not generate the messages by itself, the time is determined by the other federates. The Least Incoming Time Stamp (LITS) is the earliest possible incoming TSO event time to the federate. Although the agent does not know about the time advance requests of the other federates, the LITS of the agent provides the earliest message time for the other federation. The LITS of the AgentL means the earliest possible generating TSO message time of AgentU and vice versa. Therefore, the agent always requests TARA to its LITS to represent the federation.

Figure 14 shows the algorithms for Time Synchronization. AgentL and AgentU work identically. An agent queries the LITS periodically and compares it with previous LITS. If the LITS has increased, the agent saves the value and delivers it to the other agent. When the agent receives the new time from the other, it requests TARA to the received time. If the agent is in time advancing state—the agent has requested a time advance but it
was not granted by RTI – it stores the time to ‘NextTime’. Although ‘NextTime’ already stores the previous value, the proxy need not preserve the value, and the new value is overwritten. When receiving the time advance grant from the RTI, it requests TARA to ‘NextTime’ if the time is valid.

In lines 7~10, the agent requests TARA to the current time if it was granted and did not request a new time advance. This part is used to solve deadlock problems that may occur with the specific condition. TARA to the current time does not affect the time advance of the other federates but is related with the delivery of TSO messages. Federates can receive TSO messages when they are in time advancing states. TSO messages that are generated after the grant of the proxy, cannot be delivered to the proxy until it requests a new time advance. Owing to the messages, the LITS of the proxy may not be changed even though federates request time advances. TARA to the current time makes it possible to receive the TSO messages.

Figure 15 shows the Time Synchronization protocol for a two-level hierarchical federation. AgentL queries the LITS periodically in the user federation and checks the value. When a federate requests a new time advance, its guaranteeing time is increased and the LITS of AgentL may be increased. When the LITS is increased, AgentL delivers the new value to AgentU. AgentU changes its guaranteeing time by requesting a time advance. In the Agent Federation, AgentUs requests a time advance when the LITS of AgentL is changed by the time advance request of the federates. AgentU also queries its LITS periodically. The LITS of the AgentU may be increased when the other AgentU requests a time advance. A time advance request of a federate may cause a time advance request of AgentU, and it may cause the time advance request of the other AgentUs.

4.5. Federation Synchronization protocol

Figure 16 shows the algorithms for Federation Synchronization for a two-level hierarchical federation. The registration process is the same in AgentL and AgentU. When the RTI announces the point, the agent sends the point to the other agent if the point is not registered by itself. The point that is registered by the agent has been propagated from the other federation and it need not be delivered to the federation. The received agent registers the same point in a joined federation. After the registration, the behaviors of AgentL and AgentU are different.

In a user federation, AgentL monitors the achievement of the user federates. We assume that AgentL can receive the achievement by MOM service or Agent–User protocol. When a federate achieves the point, AgentL checks the remaining federates. When all federates achieve the point, AgentL reports the achievement to AgentU and waits for the response of AgentU. When AgentU announces the synchronization of the Agent Federation, AgentL achieves the point in the user federation. As described above, all
federates except AgentL already achieved the point in the user federation. By reporting the achievement, all federates including AgentL achieve the point, and the RTI notices that all federates are synchronized.

After the RTI announces the synchronization point, AgentU waits until AgentL reports the achievement. When AgentL reports the achievement, AgentU achieves the point in Agent Federation. This means that all federates achieve the point in their user federation. After achieving the point, AgentU waits for the synchronization from the RTI. The RTI announces ‘Synchronized’ to AgentUs when all of them achieve the point. At this point, all user federates achieve the synchronization point and AgentUs announce the fact to AgentLs.

Figure 17 shows the registration protocol of Federation Synchronization in a two-level hierarchical federation. A synchronization point is registered by a federate and announced to the federates in a user federation. The proxy also receives the point and registers the same point in the Agent Federation. After the registration, the RTI

---

**Figure 16. Algorithms for the Federation Synchronization protocol. RTI: Run-Time Infrastructure.**

```plaintext
1. Announce(L):
2. if(RegisterFlag[L] = false)
3. RegisterToUser(L)
4. end if
5. end
6. RegisterToAgent(L):
7. Register(L);
8. RegisterFlag[L] = true;
9. end
10. AchievedToAgent(L):
11. Achieved(L);
12. end
13. Synchronized(L);
14. AchievedToUser(L);
15. end

n = the number of the joined federates
1. Announce(L):
2. Remained[L] = n;
3. if(RegisterFlag[L] = false)
4. RegisterToAgent(L);
5. end if
6. end
7. RegisterToUser(L):
8. Register(L);
9. RegisterFlag[L] = true;
10. end
11. UserAchieved(L):
12. Remained[L] = Remained[L]-1;
13. if(Remained[L] = 0)
14. AchievedToAgent(L);
15. RegisterFlag[L] = false;
16. end if
17. end
18. AchievedToUser(L)
19. Achieved(L);
20. end
```

**Figure 17. Registration protocol for Federation Synchronization. RTI: Run-Time Infrastructure.**
announces the point to all AgentUs. The AgentUs, except the registrant, deliver the point to the user federation, and the received AgentLs register the same point in the user federations. As a result, the synchronization point is registered in all user federations.

Figure 18 shows the achievement protocol of Federation Synchronization. Each federate can achieve the point after announcing the points, even though the point is not registered in the other federations. AgentL reports the achievement to AgentU when all federates report achievement. After receiving the achievement, AgentU achieves the point instead of its user federation in the Agent Federation. At this point, AgentL waits for the response of AgentU and AgentU waits for the achievement of the other AgentUs, which will achieve the point instead of their user federations. After all AgentUs achieve the synchronization point, the RTI announces ‘Synchronized’ to all AgentUs. AgentUs announce the synchronization of the Agent Federation to AgentLs, and AgentLs achieve the point in their user federations. As a result, all federates including AgentLs achieve the synchronization point in the user federations and the RTIs announces ‘Synchronized’ to all user federations. As a result, all federates are synchronized in a two-level hierarchical federation.

4.5.3. Race condition handling. In a federation, two or more synchronizations can be processed in parallel, but the same synchronization points are not allowed. When a federate tries to register the point that is already registered and not achieved yet, the RTI notices the failure of the registration. Though two or more federates may register the same point simultaneously, only one registration is accepted, and the others are rejected. Because a single RTI handles the registrations, it can determine whether to accept registrants according to its criterion. The criterion is not specified in HLA, and RTI generally accepts the point that is registered first.

Competition of the registrations also occurs in a two-level hierarchical federation. Competition in the same federation can be handled by the RTI, but registrations can occur in different federations. In this case, each registration is handled by different RTIs, and all of them can be accepted. Then, their proxies try to register the same point in the Agent Federation. Only one registration succeeds, and the others fail. The accepted proxy can register the point in its user federation, but the rejected proxy should handle the failure. To represent the failure, AgentLs should stop the synchronization in the user federations. However, Federation Synchronization does not allow the ‘NotAchieved’ for the registered point and the registered point cannot be canceled until all federates achieve the point.

Figure 19 describes the problem when the race condition occurs. The same points are registered in different federations, but only one registration should be accepted originally. The proxies try to register the points in the Agent Federation. As only one registration is accepted, a failed proxy receives the point of the other proxy, and it
should register the point in its user federation. However, the point is already registered by another federate, and its registration will fail.

Because the same points mean the same synchronization jobs, the rejected proxy need not stop the synchronization in its user federation. Though its registration fails, AgentU receives the same point that is registered by the other AgentU. The point means the same job with that of the point registered in its user federation. When a point is announced, the proxy originally registers the same point in its user federation, but it does not register the point in this case because the same point was already registered. In the proposed algorithm, AgentU sets RegisterFlag although it fails to register a point to represent this behavior. Compared with a single federation, two or more points are registered successfully but all of them mean the same jobs. As a result, only one synchronization point is processed and this is the same with the single federation, although the registrants may be two or more federates.

4.6. Unanimity protocol

Figure 20 shows the algorithms of the agents for Unanimity protocol. The service starts when the RTI notices the service initiation. The initiation process is similar to that of the Federation Synchronization and is identical for AgentL and AgentU. When the RTI initiates the service, the agent delivers the information of the service to the other agent if it is not requested by itself. Because the job requested by the agent means that it is requested from the other federation, it need not deliver the other federation. The agent requests the same service in the joined federation when it receives the service request from the other. After initiation, the behaviors of AgentL and AgentU are different.

After the service initiation, AgentL checks the related states of the joined federation. Federation save/restore belongs to this protocol, and the RTI provides the query services for those services. AgentL calls ‘Query Federation Save (Restore) Status’ periodically to check the current states. The RTI responds to AgentL by the callback ‘Federation Save (Restore) Status Response’. AgentL queries until the RTI notices the service failure or all of the federates are in ‘Federate Waiting for Federation to Save (Restore)’ state, which means that all federates are saved (restored) successfully. Each federate must report ‘Complete’ or ‘NotComplete’.

If all federates report ‘Complete’, they are in ‘Federate Waiting for Federation to Save’ state and AgentL reports the completion to AgentU. After reporting, AgentL waits for the response from AgentU. In this case, the service success of the user federation is determined by the report of AgentL and it reports the results of the Agent Federation. The service succeeds if AgentL reports ‘Complete’ and fails if AgentL reports ‘NotComplete’. If a federate reports ‘NotComplete’, the service fails and the RTI notifies the result to all federates. AgentL reports the failure to AgentU and the service is finished in the user federation.

AgentU waits for the result from AgentL after the service initiation. If AgentL reports the completion, AgentU reports ‘Complete’ instead of the user federation. When this case, AgentU waits for the response of the RTI; RTI notices the service success only when all AgentUs report ‘Complete’. When RTI notices the service result, it delivers the result to AgentL. As described above, this result determines the service success of the user federations. If AgentL reports the failure, AgentU reports ‘NotComplete’. In this case, the service will fail by AgentL and the RTI notifies the service failure to all AgentUs. The reported
AgentU does not need to send the result to the user federation because its federation already failed.

Figure 21 shows the initiation protocol of the Unanimity in a two-level hierarchical federation. Initiated service is propagated to the Agent Federation and the other federations through the proxies.

Figure 22 shows the reporting protocol for a service success case. AgentL queries the states periodically until all federates report ‘Complete’. AgentU reports ‘Complete’ in the Agent Federation when AgentL announces the result. Each AgentL works identically in the user federation and reports ‘Complete’ in the Agent Federation. When all AgentUs report ‘Complete’, the RTI announces the success to all AgentUs. As a result, all AgentUs report ‘Complete’ in their federations and the RTIs announce the service success to all federates.

Figure 23 shows the reporting protocol for a service failure case. Several user federations can report the completion. The federations work as a service success case, and its proxy reports ‘Complete’ in Agent Federation. At least one federate reports ‘NotComplete’ to the RTI in a user federation, which causes the service failure in the user federation, and the RTI notifies the failure to all federates. AgentL also receives the failure and announces it to AgentU. AgentU reports ‘NotComplete’ instead of the user federation and causes the service failure in the Agent Federation. As a result, AgentLs report the ‘NotComplete’ in user federations, and all user federations fail due to the failure of the AgentLs.

4.6.1. Race condition handling. Race condition problem also occurs for the Unanimity protocol. Compared with
Federation Synchronization, a service request competes with all of the other requests. ‘Save’ competes not only with the other ‘Save’ of the other labels but also with ‘Restore’. In a two-level hierarchical federation, two or more requests can be accepted in user federations and compete in the Agent Federation. Competition for the same job with the same labels can be handled like in Federation Synchronization. However, different jobs can compete in Unanimity. The proxy should handle this case differently.

Figure 24 describes the problem when the race condition occurs. Two requests are initiated at once in different federations, but only one request should be accepted originally. In the Agent Federation, $L1$ is accepted and $L2$ is rejected. In this case, $L1$ should be processed in all federations, but it cannot be initialized in the user federation, which has initialized $L2$.

When the competition occurs in the Agent Federation, one AgentU is accepted and the other AgentUs receive the rejections. The RTI notices the failure and initiates the accepted jobs (this order may be reversed). Figure 25 illustrates the behavior of the rejected AgentU. The rejected AgentU delivers the result and the accepted job to AgentL. AgentL cancels the job by reporting the ‘NotComplete’ and the processing job fails. The RTI announces the failure to all federates. AgentL requests a new job, which is accepted in the Agent Federation. As a result, the same job is initiated in all federations.
4.7 Interoperability for HLA standards

Federations can be constructed in three types of HLA standards – IEEE 1516-2010, IEEE 1516-2000 and HLA 1.3. This paper handles federations of IEEE 1516-2010, but it also can be applied to IEEE 1516-2000 and HLA 1.3. Federations of same HLA standards can be interoperated in a two-level hierarchical federation according to the proposed algorithms. Federations of different HLA standards should consider some services for interoperation. Most services had not been changed when HLA standard was revised but we should consider change of the some services.

IEEE 1516-2010 provides more services than IEEE 1516-2000. However, additional services like querying information belong to Isolation protocol type and not affect the other federates. They can be interoperated because service protocols are same and additional arguments like update rate affects only the requesting federate. IEEE 1516-2000 had changed the name of interfaces from

---

**Figure 23.** Protocol for service failure of Unanimity. RTI: Run-Time Infrastructure.

**Figure 24.** Race condition for Unanimity. RTI: Run-Time Infrastructure.
HLA 1.3 but most service protocols are same. Data types of arguments used in HLA services are defined in HLA DLC Standards. HLA 1.3 and IEEE 1516 define different types but they can be translated each other. Therefore, IEEE 1516 and HLA 1.3 can be interoperaed by translating interfaces and arguments for most services. However, they define the different protocols for negotiated ownership transfer. Ownership management services can be limited to use for interoperation of HLA 1.3 and IEEE 1516.

5. Service verification

This section verifies the service management of a two-level hierarchical federation. Section 5.1 verifies the algorithms described in Section 4. It shows that results of a two-level hierarchical federation are the same as those of a single federation. Section 5.2 shows a case study to show a practical application, which simulates two forces in different federations.

5.1. Algorithm verification

Services of Isolation are not propagated to the other federations and are processed only in a federation. The algorithm for Time Synchronization was proposed and verified in Yoo et al. Therefore, this section proves the services of the other protocols.

To prove the algorithms, we compare the results of a two-level hierarchical federation with those of a single federation. We assume that member federates are the same in two architectures, and federates are placed in multiple user federations for a two-level hierarchical federation. When a federate requests a service, RTI handles it and provides callback services to joined federates. This section will show that results of a service request are the same in two architectures.

5.1.1. Information Routing. To prove Information Routing, we will show that the federates which receive the updated object information are the same in a federation and in a two-level hierarchical federation. Hidden objects, which are not shared with other federations, are not considered in this section. In this case, the service is processed differently in two architectures, but this situation is intended by the user who constructs the hierarchical federation for data security. Therefore, we assume that all common objects of FOMs are included in the proxy’s SOM and that the following equation is satisfied:

\[ \text{SOM}_A = \text{FOM}_A \setminus (\text{FOM}_B \cup \text{FOM}_C \cup \ldots), \]
\[ \text{SOM}_B = \text{FOM}_B \setminus (\text{FOM}_A \cup \text{FOM}_C \cup \ldots), \ldots \]

When a federate – called ‘Sender’– updates an object, the RTI provides the information to the federates, which subscribe the object. Though ‘Sender’ subscribes the object, it does not receive the information that is updated by itself.

To verify the algorithm, the following sets are defined for an updated object. TargetSet_{User} includes the user federates that subscribe the object, and TargetSet_{Agent} includes the agents that subscribe object. We assume that there are \( n \) user federations, and UF_k includes user federates and AgentL_k of user federation \( k \).

The following assumptions are satisfied by HLA and subscription of proxies, which is described in Section 4.3.1. In a federation, RTI provides object information to the federates subscribing it. AgentL subscribes the exchangeable objects for the other user federations, and AgentU subscribes the exchangeable objects for its user federation.

**Assumptions**

- If \( (\text{UF}_k)^c \cap \text{TargetSet}_{\text{User}} \neq \emptyset \), then \( \text{AgentL}_k \in \text{TargetSet}_{\text{Agent}} \)
• If $\text{UF}_k \cap \text{TargetSet}_{\text{User}} \neq \emptyset$, then $\text{Agent}_{U_k} \in \text{TargetSet}_{\text{Agent}}$

• When a federate (called ‘Sender’) requests ‘Send’ to RTI, RTI provides ‘Receive’ to federates included in $(\text{TargetSet} \cap F) - \{\text{‘Sender’}\}$. (F is a set of member federates of the federation)

**Theorem**

If member federates are identical except agents in a federation and in a two-level hierarchical federation, federates receiving object information are the same in two architectures when a federate updates an object.

**Proof:**

Case 1. In a single federation

All federates join a federation and RTI handles requests from them. When ‘Sender’ requests ‘Send’ service to RTI, the RTI routes the information through ‘Receive’ callback service. Receiving federates are determined by federates’ subscription and will be the following federates by assumption.

$$\text{TargetSet}_{\text{User}} - \{\text{‘Sender’}\}$$

Case 2. In a two-level hierarchical federation

We assume that ‘Sender’ is placed in user federation $p$.

When ‘Sender’ requests ‘Send’ service, an RTI of user federation $p$ routes the information in the federation. In user federation $p$, following federates receive ‘Receive’ callback service from the RTI

$$(\text{TargetSet}_{\text{User}} \cap \text{UF}_p) - \{\text{‘Sender’}\}$$

The object information may or may not be propagated to other federations depending on the subscription of agents.

Case 2.1. If $(\text{UF}_p) \cap \text{TargetSet}_{\text{User}} = \emptyset$ (There is no subscribing federate in the user federations.)

In this case, $\text{Agent}_{L_p}$ does not belong to $\text{TargetSet}_{\text{Agent}}$ by assumption. Therefore, $\text{Agent}_{L_p}$ does not receive ‘Receive’, and the service is finished. Due to $(\text{UF}_p) \cap \text{TargetSet}_{\text{User}} = \emptyset$, the following equation is satisfied:

$$(\text{TargetSet}_{\text{User}} \cap \text{UF}_p) \cap \{\text{‘Sender’}\}$$

As a consequence, receiving federates are as follows, and they are the same with the set of a single federation

$$(\text{TargetSet}_{\text{User}} \cap \text{UF}_p) - \{\text{‘Sender’}\}$$

$$\text{TargetSet}_{\text{User}} - \{\text{‘Sender’}\}$$

Case 2.2. If $(\text{UF}_p) \cap \text{TargetSet}_{\text{User}} \neq \emptyset$ (There are subscribing federates in the other user federations.)

In this case, $\text{Agent}_{L_p}$ belongs to $\text{TargetSet}_{\text{Agent}}$ by assumption and receives ‘Receive’ from RTI in user federation $p$. $\text{Agent}_{L_p}$ delivers it to $\text{Agent}_{U_p}$ and $\text{Agent}_{U_p}$ sends ‘Send’ in Agent Federation. In Agent Federation, RTI routes the information to the $\text{Agent}_{Us}$. All $\text{Agent}_{U_k}$ which satisfies the condition – $(\text{TargetSet}_{\text{User}} \cap \text{UF}_k) \neq \emptyset$, will receive ‘Receive’ from RTI, and they deliver the information to $\text{Agent}_{L_k}$. In each user federation $k$, $\text{Agent}_{L_k}$ updates the object and following federates receive the callback from RTI.

$$\text{TargetSet}_{\text{User}} \cap \text{UF}_k, \text{ for } k = p$$

As a result, following federates receive the object information in a two-level hierarchical federation.

$$((\text{TargetSet}_{\text{User}} \cap \text{UF}_p) - \{\text{‘Sender’}\})$$

$$\cup (\bigcup_{k \neq p} \text{TargetSet}_{\text{User}} \cap \text{UF}_k)$$

$$= (\text{TargetSet}_{\text{User}} \cap \text{UF}_p) - \{\text{‘Sender’}\}$$

$$\cup (\text{TargetSet}_{\text{User}} \cap \text{UF}_p)$$

$$= (\text{TargetSet}_{\text{User}} \cap (\text{UF}_p \cup \text{UF}_p)) - \{\text{‘Sender’}\}$$

$$= \text{TargetSet}_{\text{User}} - \{\text{‘Sender’}\}$$

The receiving federates are the same as those of a single federation.

5.1.2. Federation Synchronization. To prove Federation Synchronization, we will show that all federates are synchronized certainly when a synchronization point is registered. This is clear in a federation. Therefore, this section shows that there is the same result in a two-level hierarchical federation.

The following assumptions are satisfied by HLA specifications and the algorithm of the proxy. Synchronization can be accomplished when all federates achieve the point, and HLA requires that all federates achieve the point after RTI announces a synchronization point.

**Assumptions**

- RTI announces the synchronization point to all joined federates when the point is registered.
- When RTI announces a synchronization point, all federates report ‘Achieved’ after a finite physical time.
- When all federates report ‘Achieved’, RTI notifies ‘Federation Synchronized’ to all joined federates.
- $\text{Agent}_{L_k}$ acquires the achievement of federates in its user federation $k$. 

$\text{TargetSet}_{\text{User}} \cap \text{UF}_k, \text{ for } k = p$
• Agent\textsubscript{U\textsubscript{k}} achieves the point when UF\textsubscript{k} - \{AgentL\textsubscript{k}\} achieve the point in user federation k

\textbf{Theorem}

When a federate registers a synchronization point, all federates receive the ‘Federation Synchronized’ from RTI in a two-level hierarchical federation after they achieve the point.

\textbf{Proof:}

We assume that there are n user federations, and the synchronization point is registered in user federation p.

When a federate registers a point in user federation p, RTI announces the point to all federates of UF\textsubscript{p}. After a finite physical time, all federates achieve the point except the AgentL\textsubscript{p}, but RTI will not announce the ‘Federation Synchronized’ until AgentL\textsubscript{p} achieves the point.

Because the UF\textsubscript{p} includes AgentL\textsubscript{p}, AgentL\textsubscript{p} delivers the point to AgentU\textsubscript{p} and it registers the same point in Agent Federation. RTI announces the point to all AgentU\textsubscript{s}. Each AgentU\textsubscript{k}(k \neq p) delivers the point to AgentL\textsubscript{k} and AgentL\textsubscript{k} registers the same point in user federation k. As a result, each RTI announces the point to all federates of UF\textsubscript{k}. By the assumption, all federates of user federation k will achieve the point after the finite time from the registration.

We assume that all user federates achieve the point after time t\textsubscript{k} in user federation k. At time t\textsubscript{k}, In other words, UF\textsubscript{k} - \{AgentL\textsubscript{k}\} achieves the point in user federation k. As a result, AgentL\textsubscript{k} announces the achievement to AgentU\textsubscript{k} and AgentU\textsubscript{k} achieves the point in Agent Federation.

At physical time t, \(\bigcup_{\text{all } k} \{\text{AgentU}\textsubscript{k}\} \) achieve the point in Agent Federation.

Let \(t_{\text{max}} = \text{Max}(t\textsubscript{k})\). After \(t_{\text{max}}\), all AgentU\textsubscript{s} achieve the point.

At this point, RTI announces ‘Federation Synchronized’ to AgentU\textsubscript{s} by assumption. Each AgentL\textsubscript{k} achieves the point in user federation k. In user federation k, following federates achieve the point. Because AgentU\textsubscript{k}’s condition for achieving the point is ‘UF\textsubscript{k} - \{AgentL\textsubscript{k}\} achieve the point’, the following federates achieve the point in user federation k after propagation delay.

\begin{align*}
\{\text{UF}\textsubscript{k} - \{\text{AgentL}\textsubscript{k}\}\} & \cup \{\text{AgentL}\textsubscript{k}\} = \text{UF}\textsubscript{k}, \quad \text{for all } k
\end{align*}

Because all federates of user federation k achieve the point, RTI announces the ‘Federation Synchronized’ to them. Consequently, these federates receive ‘Federation Synchronized’ in a two-level hierarchical federation.

\(\bigcup_{\text{all } k} \text{UF}\textsubscript{k} = \text{all federates}\)

5.1.3. \textit{Unanimity}. To prove \textit{Unanimity}, this section shows the results of \textit{Unanimity} job are the same in a federation and a two-level hierarchical federation. \textit{Unanimity} job succeeds only when all federates report ‘Complete’ and fails otherwise. This behavior is clear in a federation by HLA specification. Therefore, this section shows the behavior of a two-level hierarchical federation. The following assumptions are satisfied by HLA and the algorithm of the proxy.

\textbf{Assumptions}

• RTI initiates the job for all joined federates when a federate requests a \textit{Unanimity} job.
• All federates that receive ‘Initiate’ report ‘Complete’ or ‘NotComplete’ after a finite physical time.
• When all federates report ‘Complete’, RTI notifies ‘Succeeded’ to all federates. If a federate reports a ‘NotComplete’, RTI notifies the ‘Failed’ to all federates.
• AgentL\textsubscript{k} acquires the reporting states of the other federates using ‘Query Status’ Service in user federation k.
• AgentU\textsubscript{k} reports ‘Complete’ when UF\textsubscript{k} - \{AgentL\textsubscript{k}\} report ‘Complete’ in its user federation and reports ‘NotComplete’ when RTI notifies the failure of the job.

\textbf{Theorem}

When a federate requests a service of \textit{Unanimity}, all federates receive ‘Succeeded’ only when all federates reports ‘Complete’. If a federate reports ‘NotComplete’, all federates receive ‘Failed’.

\textbf{Proof:}

We assume that there are n user federations, and the service is requested in user federation p.

When a federate requests a service in user federation p, RTI announces the service initiation to all federates of UF\textsubscript{p}. Because UF\textsubscript{p} includes AgentL\textsubscript{p}, AgentL\textsubscript{p} announces the service to AgentU\textsubscript{p}, and it requests the same service in Agent Federation. Then, RTI initiates the service to all AgentUs. Each AgentU\textsubscript{k}(k \neq p) announces the service to AgentL\textsubscript{k} and AgentL\textsubscript{k} requests the service in user federation k. As a result, all federates receive the service initiation from RTI. After initiation, the results depend on service success.

\textbf{Case 1. Service success}

We assume that all federates of the user federation k report ‘Complete’ until physical time t\textsubscript{k}. At time t\textsubscript{k}, UF\textsubscript{k} - \{AgentL\textsubscript{k}\} have reported ‘Complete’ in user federation k. As a result, AgentL\textsubscript{k} reports the result to AgentU\textsubscript{k} and AgentU\textsubscript{k} reports ‘Complete’ in Agent Federation.

At time t, \(\bigcup_{\text{all } k} \text{UF}\textsubscript{k} - \{\text{AgentL}\textsubscript{k}\} \) report ‘Complete’ in Agent Federation.

Let \(t_{\text{max}} = \text{Max}(t\textsubscript{k})\). After time \(t_{\text{max}}\), all AgentU\textsubscript{s} report ‘Complete’.

\begin{align*}
\bigcup_{\text{all } k} \{\text{AgentU}\textsubscript{k}\} &= \text{all AgentU}
\end{align*}
Then, RTI announces ‘Succeeded’ to AgentUs. Each AgentLk reports ‘Complete’ in user federation k. In user federation k, following federates report ‘Complete’. Because AgentUk’s condition for reporting ‘Complete’ is UFk-\{AgentLk\} report ‘Complete’, the following federates achieve the point in user federation k

\[ UF_k - \{AgentL_k\} \cup \{AgentL_k\} = UF_k \]

Because all federates of user federation k achieve the point, RTI announces ‘Succeeded’ to them. Consequently, these federates receive ‘Succeeded’ in a two-level hierarchical federation.

\[ \cup_{\text{all} k} UF_k = \text{all federates} \]

Case 2. Service failure

We assume that all federates of the user federation k report ‘Complete’ until physical time \( t_k \) and a federate reports ‘NotComplete’ in user federation q at time \( t_q \). At this point, RTI announces ‘Failed’ in user federation q and UFq receives ‘Failed’.

At time \( t_q \), \( \cup_{\text{all} k \leq t_q} \{AgentU_k\} \) report ‘Complete’ in Agent Federation and AgentUq reports ‘NotComplete’. After the reporting, RTI announces ‘Failed’ to all AgentUs by the assumption. Each AgentLk (k\( \neq q \)) reports ‘NotComplete’ in user federation k. In user federation k, RTI notifies ‘Failed’ to all federates. At this point, following federates receive ‘Failed’.

\[ \cup_{k \neq q} UF_k = \text{all federates} \]

### 5.2. Case study: air-defense simulation

To show a practical application of a two-level hierarchical federation, this section shows the simulation, that is performed by interoperating different federations. A proxy is implemented for the two-level hierarchical federation and used to interoperate federations. The proposed work is applied to the air defense simulation.

We assume that the target systems are developed in different federations. An air-defense system, which consists of radars, missile launchers and the Command and Control (C2) system, is developed in a federation of Pitch pRTI Evolved 4.2.5. Bombers, which attack the base of the air-defense system, are developed in the other federation of MÁK RTI 3.4. Figure 26 shows the simulation environments of the air defense system. The federations are created in each RTI and Agent Federation is operated with Pitch pRTI.

Federates are developed as HLA-compliant simulators and HLA services are used to perform simulations in a two-level hierarchical federation. Table 2 summarizes services used for simulation and their protocol types.

Object models are classified into two types. Common objects are exchangeable between federations and belong to SOM of proxies. The positions of objects, engagement and damage information belongs to common objects and shared in Agent Federation by proxies. Hidden objects are exchangeable only in a user federation and are excluded in SOM of proxies. The air-defense system exchanges the internal information, such as the detection target and attack orders in its federation. They belongs to hidden objects. In FOMs and SOMs, object positions are defined by object
classes, and the other information is defined by interaction class.

Following behaviors are defined to check the operation of HLA services. To check the initialization of the federates, simulation starts after federates are synchronized by a synchronization point. All simulators operates with the logical time, and their times are synchronized in the two-level hierarchical federation. Each federates requests **Time Advance Request Available** service to request a time advances, and waits until RTI grants the request. In addition, federates provide functions for simulation save/restore.

Figure 27 shows the simulation of the system. SIMDIS is used to display the simulation. The left federation is made in Pitch pRTI and simulates the air-defense system. The right federation is made in MÄK RTI and simulates the bombers.

6. Conclusion

Interoperation of federations has been researched due to the limitation of data security and interoperations of other RTIs. HLA specifications define only interoperation of federates in a single federation. However, data security and interoperation of other RTIs cannot be achieved in a single federation. The proxy method was proposed for the interoperation of federations, but this presents problems because HLA does not specify the interoperation of federations. Previous research has examined a part of services and focused on the problems of the proxy method. Because HLA defines more than 100 services, it is hard to define the behavior of the proxy. This paper classifies the services into five protocol types and proposes an HLA service management for the interoperation of federations based on the protocols.

In the proposed work, the two-level hierarchical federation is used for the interoperation of multiple federations. The target federations are placed in the lower level and are called user federations. The Agent Federation, which is the upper-level federation, consists of the agents representing a user federation. In the two-level hierarchical federation, the simulation of systems occurs in the user federations, and the Agent Federation interoperates them. The data structure is defined based on FOMs of existing federations and supports data security, which is achieved by defining hidden objects of each federation. For each service protocol, we define the algorithms of the agents. We show how the protocol is modified in the two-level hierarchical federation. Race condition may occur in **Federation Synchronization** and **Unanimity** protocols. The proxy handles the problems to show the same results of a single federation and describes the protocol when the race condition occurs. The algorithms and solutions of problems make it possible to support HLA services in a two-level hierarchical federation.

The algorithms are verified by theoretical approaches and show the application for the real simulation by a case study. The case study is applied to the air-defense system, which is developed in different RTIs. They are interoperated in a two-level hierarchical federation and simulate the system successfully.

The proposed work will provide extended interoperability and reusability of HLA. Interoperation can be applied to existing federations and additionally secures accessible data. This work can be used for the framework of the federation developments and management of the distributed simulations in HLA.

For further research, simulation performance in the two-level hierarchical federation can be issued to apply the proposed works to practical applications. As the number of federates are increased, data traffic of the federations may be increased significantly. In addition, services should be propagated to the other federations through proxies and the Agent Federation. The propagation delay and bottlenecks of data traffics may affect the simulation performance in the two-level hierarchical federation. The performances of time management and object exchange, which are frequently used, should be evaluated through experiments in the two-level hierarchical federation.

Funding

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD140022PD, Korea.

References

3. DMSO. *HLA RTI 1.3-Next generation programmer’s guide version 5*, 1999.
7. Kim BS, Chang BC and Tag GK. Multifaceted modeling and simulation framework for system of systems using HLA/RTI. In: *2013 Spring simulation multi-conference, 16th
communications and networking symposium (CNS), San Diego, CA, USA, 2013.
**Author biographies**

**Min-Wook Yoo** received his BS and MS in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2007 and 2009, respectively. He is currently a PhD candidate at the Department of Electrical Engineering at KAIST. His research interests include interoperation of federations for HLA/RTI, distributed system in HLA/RTI and methodology for modeling and simulation of discrete event systems.

**Changbeom Choi** received his BS in computer engineering from Kyung Hee University and his MS in computer science from the Korea Advanced Institute of Science and Technology (KAIST) in 2005 and 2007, respectively. He received his PhD in electrical engineering with specialization in systems modeling and simulation from the KAIST, 2014. He has been an assistant professor at the School of Creative Convergence Education in Handong Global University from 2014. His research interests include discrete event systems modeling & simulation, verification, validation, and accreditation (VV&A), defense modeling & simulation, and agent-based simulation.

**Tag Gon Kim** received his PhD in computer engineering with specialization in systems modeling and simulation from the University of Arizona (Tucson, Arizona) in 1988. He was an assistant professor of electrical and computer engineering at the University of Kansas (Lawrence, Kansas) from 1989 to 1991. He joined the Electrical Engineering Department, KAIST (Daejeon, Korea) in fall 1991, and he has been a full professor at the EECS Department since fall 1998. He was the president of The Korea Society for Simulation (KSS). He was the editor-in-chief for *Simulation: Transactions for Society for Computer Modeling and Simulation International* (SCS), and he is a coauthor of the textbook *Theory of Modeling and Simulation* (Academic Press, 2000). He has published about 200 papers on modeling and simulation theory and practice in international journals and conference proceedings. He is very active in defense modeling and simulation in Korea. He was/is a consultant for defense modeling and simulation technology at various Korean government organizations, including the Ministry of Defense, Defense Agency for Technology and Quality (DTAQ), Korea Institute for Defense Analysis (KIDA), and Agency for Defense Development (ADD). He is a Fellow of the SCS and a Senior Member of the IEEE.