TECHNICAL ARTICLE

SIMULATION 71:6, 7 378–387

ISSN 0037-5497/98
Printed in the United States of America

Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation

Judith S. Dahmann
Defense Modeling and Simulation Office
U. S. Department of Defense

Frederick Kuhl
The MITRE Corporation

Richard Weatherly
The MITRE Corporation

The High Level Architecture (HLA) is an architecture for reuse and interoperation of simulations. It is based on the premise that no simulation can satisfy all uses and users. An individual simulation or set of simulations developed for one purpose can be applied to another application under the HLA concept of the federation: a composable set of interacting simulations. The intent of the HLA is a structure which will support reuse of capabilities available in different simulations, ultimately reducing the cost and time required to create a synthetic environment for a new purpose, and the possibility of distributed collaborative development of complex simulation applications. The HLA is widely applicable across a full range of simulation application areas. The widely differing application areas indicate the variety of requirements that have been considered in development and evolution of the HLA. The HLA does not prescribe a specific implementation, nor does it mandate the use of any particular software or programming language. Over time, as technology advances, new and different implementations will be possible within the HLA framework. This paper describes the technical motivations for the HLA, the key elements of the architecture and how they are minimum and essential to the goal of reuse and interoperability.

Keywords: High level architecture, HLA, software architecture, standards, reuse, interoperability

1. Introduction

Setting standards for simulation involves a delicate balance between the need for clear definition and direction, and the desire to allow flexibility and extensibility. Determining what to standardize and what to leave to the user is key to a successful standard.

The High Level Architecture (HLA) for simulation interoperability is an example of minimal but sufficient standards for general application. HLA is an architecture for reuse and interoperation of simulations. It is based on the premise that no simulation can satisfy all uses and users. An individual simulation or set of simulations developed for one purpose can be applied to another application under the HLA concept of the federation: a composable set of interacting simulations. The intent of the HLA is a structure which will support reuse of capabilities available in different simulations, ultimately reducing the cost and time required to create a synthetic environment for a new purpose, and the possibility of distributed collaborative development of complex simulation applications.

The HLA is widely applicable across a full range of simulation application areas, including education and training, analysis, engineering and even entertainment, representing entities at many levels of resolution. These widely differing application areas indicate the variety of requirements that have been considered in development and evolution of the HLA. The HLA does not prescribe a specific implementation, nor does it mandate the use of any particular software or programming language. Over time, as technology advances, new and different implementations will be possible within

the HLA framework. This paper describes the technical motivations for the HLA, the key elements of the architecture and how they are minimum and essential to the goal of reuse and interoperability.

2. The High Level Architecture (HLA) Defined

Basic terms are used throughout this article and in other descriptions of the HLA:

- A *federation* is a set of simulations, a common federation object model, and supporting runtime infrastructure that are used together to form a larger model or simulation.
- A *federate* is a member of a federation, one point of attachment to the infrastructure. A federate could represent one platform, like a cockpit simulator. Or a federate could represent an aggregate simulation, like an entire national simulation of air traffic flow.
- A *federation execution* is a session of a federation executing together.

The HLA standard has three parts:

- HLA Rules
- Object Model Template
- Interface Specification

HLA Rules

The HLA Rules are principles and conventions which must be followed to achieve proper interaction of federates during a federation execution. These describe the responsibilities of federates and federation designers.

Object Model Template (OMT)

The OMT is the prescribed common method for describing the entities to be simulated and interactions between entities in a federation. The OMT is an example of a meta-model, i.e., a way to describe models of information. Each prospective federate has a *Simulation Object Model*, or SOM, that describes the data the federate can produce or consume. A given federation has a *Federation Object Model*, or FOM, that defines what part of the union of all the federates' SOMs will be used in the federation. The OMT is the meta-model for both SOMs and FOMs.

Interface Specification

This is the specification of the interface between federates and the *Runtime Infrastructure* (RTI). The RTI is software that allows a federation to execute together. It is the interface between the RTI and federates that is standardized; an implementation of the RTI could take a variety of forms.

An HLA Federation Combines Rules, Object Model, and Infrastructure

The three parts of the HLA standard are applied as follows:

- An HLA-compliant federate has a Simulation Object Model documented in accordance with the OMT. The federate complies with the HLA Rules for federates.
- An HLA federation consists of: (a) a Federation Object Model documented in accordance with the OMT; (b) a set of compliant federates; and (c) a compliant implementation of the RTI.

An HLA Federation Has Software Components: Federates and RTI

As the foregoing discussion shows, an HLA federation has several software components:

- Some number of federates (usually more than one)
- An implementation of the RTI

This is depicted in the "lollipop diagram" in Figure 1.

For the HLA, a federate is defined as having a single point of attachment to the RTI. A federate might consist of several processes, perhaps running on several computers. A federate might model a single entity, like

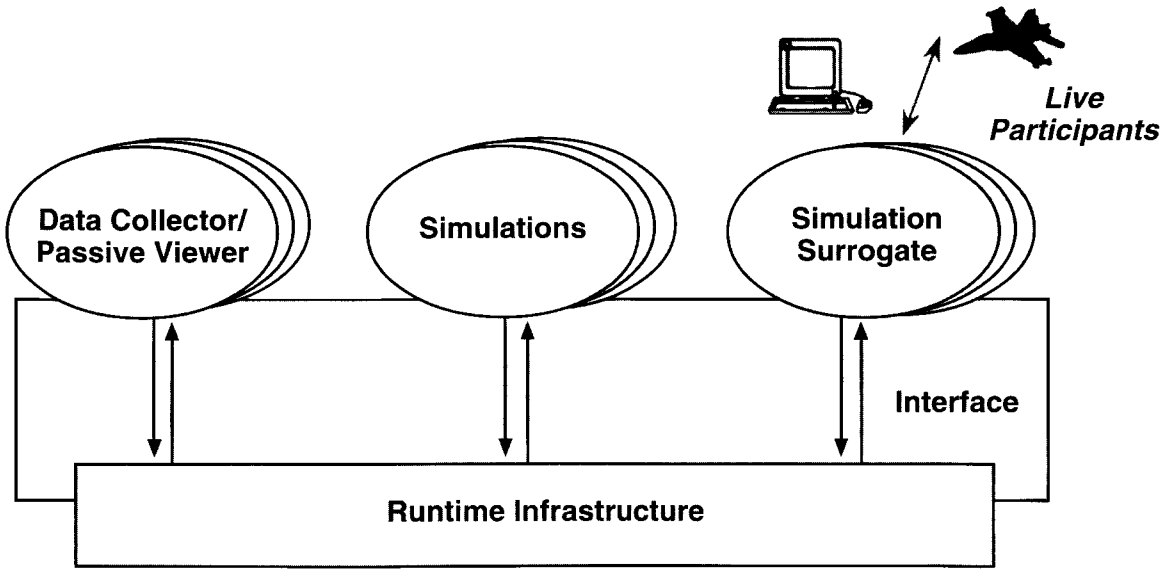


Figure 1. Software components of an HLA federation.

a vehicle, or a federate might model many entities, like all the components of an air traffic control system in a region. From the perspective of the HLA, a federate is defined by its single point of attachment to the RTI.

As shown in Figure 1, a federate might model some number of entities, or it might have a different purpose. It might be a collector and/or viewer of data, passively receiving data and generating none. Or a federate might act as a surrogate for human participants in a simulation. In this role, the federate might reflect the state of the larger simulation to the participant via some user interface, and might convey control inputs or decisions from the participant to the rest of the federation.

The RTI used by a federation may be implemented as many processes or as one. It may require many computers to execute, but conceptually it is one RTI.

The interface between each federate and its point of attachment to the RTI is defined by the HLA Interface Specification. The HLA Rules require that each federate interact with other federates through the services of the RTI, and not directly. The kinds of data to be exchanged through the RTI in a given federation execution are defined in the FOM for that federation execution.

3. The HLA Defines a Software Architecture

Goal of the HLA: Build Simulation Applications from Component Simulations

The HLA defines a software architecture. To understand that architecture, it's helpful to know what the HLA is intended to do. Its chief intent is to allow simulation applications to be created by combining simulations. In this way the HLA supports component-based simulation development, where the components are federates (not entities being simulated).

The design of the HLA is based on several premises or assumptions:

- No single, monolithic simulation can satisfy the needs of all users. Users differ in their interests and requirements for fidelity and detail.
- Simulation developers vary in their knowledge of domains to be simulated. No one set of developers is expert across all details, even in one domain.
- No one can anticipate all the uses of simulation and all the ways simulations could be usefully combined. Even if we were able to satisfy a comprehensive set of requirements in a domain of application (by building the monolithic simulation), the project would fail at the outset in anticipating the requirements for such a system. The world changes continually; nobody knows all the future uses for simulation, even in one domain.
- Future technology and tools must be incorporated. Were we able to build the comprehensive simulation, and our crystal ball functioned well enough for us to anticipate requirements, computer technology would change underneath us. If our crystal ball could show us the future of computer technology, we'd still face the problem of incorporating it as our customers demanded to benefit from its advances.

These observations led the HLA designers toward the following goals:

- It should be possible to decompose a large simulation problem into smaller parts. The smaller parts are easier to define, build correctly, and verify.
- It should be possible to combine the resulting software components into a larger simulation.
- It should be possible to combine those components with other, perhaps unanticipated, components to form a new simulation.
- Those functions that are generic to component-based simulation should be separated from specific simulations. The resulting generic infrastructure should be reusable from one simulation to the next.
- The interface between components and generic infrastructure should insulate the components from changes in the technology used to implement the infrastructure.

In this way the HLA is fundamentally an architecture to support component-based simulation, where, as we said, the components are individual simulations. The architecture also supports building simulations that are distributed across multiple computers, but that is a happy side effect of its support for components. Nothing in the architecture assumes or requires a distributed implementation.

An Architecture Involves Elements, Interactions and Patterns

Shaw and Garlan [1] define a software architecture as follows: Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on those patterns.

The three parts of the HLA standard map to elements, interactions, and patterns as follows:

- **Elements**. The Rules and the Interface Specification define the elements of an HLA federation to be the federates, an RTI, and a common object model.
- **Interactions**. The Rules and the Interface Specification define the interactions between federates and the RTI, and between federates (always mediated by the RTI). For a given federation, the federation's FOM defines the kinds of data carried by interactions between federates and the RTI. The structure of the Federation Object Model (the Object Model Template as a meta-model) is assumed as part of the Interface Specification.
- **Patterns**. The allowed patterns of composition of federates in the HLA are constrained to some extent by the Rules, and are defined in the Interface Specification, especially in the overview sections.

HLA Defines an Architecture, Not an Implementation

The HLA defines an architecture, not an implementation. In particular, the RTI is defined by its interface. Many different implementations of the RTI could meet the interface as specified. This is also true of federates. The Interface Specification defines not just the interface the RTI presents to federates, but also the interface federates present to the RTI. Every federate is yet another implementation of that interface.

The Interface Specification is abstract. The services (in both directions) are defined as procedure calls that take and return parameters, by pre- and post-conditions on the calls, and by exceptions. The definitions of the services make no reference to any programming language. Only after the definitions of the services does the Interface Specification define (in appendices) application programmer's interfaces to various programming languages.

Why define things so abstractly? One goal of the HLA is to standardize the approach to the persistent problems in federating simulations, and to avoid defining the HLA in terms of transitory technology. One may say with confidence that the evolution of programming languages will continue. And we haven't yet found the ultimate networking technology. The HLA will keep pace with new technology by defining APIs for new languages, and by incorporating new networking technology into implementations of the RTI.

Architecture Splits Functions Between Simulations and Runtime Infrastructure

The software elements of an HLA federation are: (1) an implementation of the RTI; and (2) some number of federates. The RTI is the software that allows a federation to execute together. The HLA partitions functionality in a federation into simulation-specific functions and infrastructure functions. The intent is that all behavior specific to a given model or simulation is in the federate that implements it, and that the infrastructure contains functions generic to coordination among simulations. Thus the same implementation of the RTI can be used to support many different simulation applications. And federate developers are freed from worrying about many infrastructure matters.

In Figure 2, you see the interface between the RTI on the bottom and the various kinds of federates above. Notice that the federates do not talk to each other directly. They are each connected to the RTI, and they communicate with each other using only services provided by the RTI. Notice also that the federates may be of various kinds: some are simulations in the usual sense; some may be merely passive viewers or loggers of the federation's progress; some may be surrogates for external systems or live participants in an exercise; some may be controllers and managers of the federated simulation and not simulations at all. Each federate uses those RTI services appropriate for its purpose; the RTI does not otherwise distinguish federates. Finally, notice that each federate has a single point of contact with the RTI. The single point of contact is the definition of a federate from the RTI's perspective. The federate might consist of multiple processes, perhaps running on several computers, but it maintains one connection to the RTI, and it communicates with other federates only through the RTI.

In Figure 2, we've shown the names of the interfaces between each federate and the RTI. The RTI offers to each federate an interface called *RTIambassador*. The federate invokes operations on that interface to request

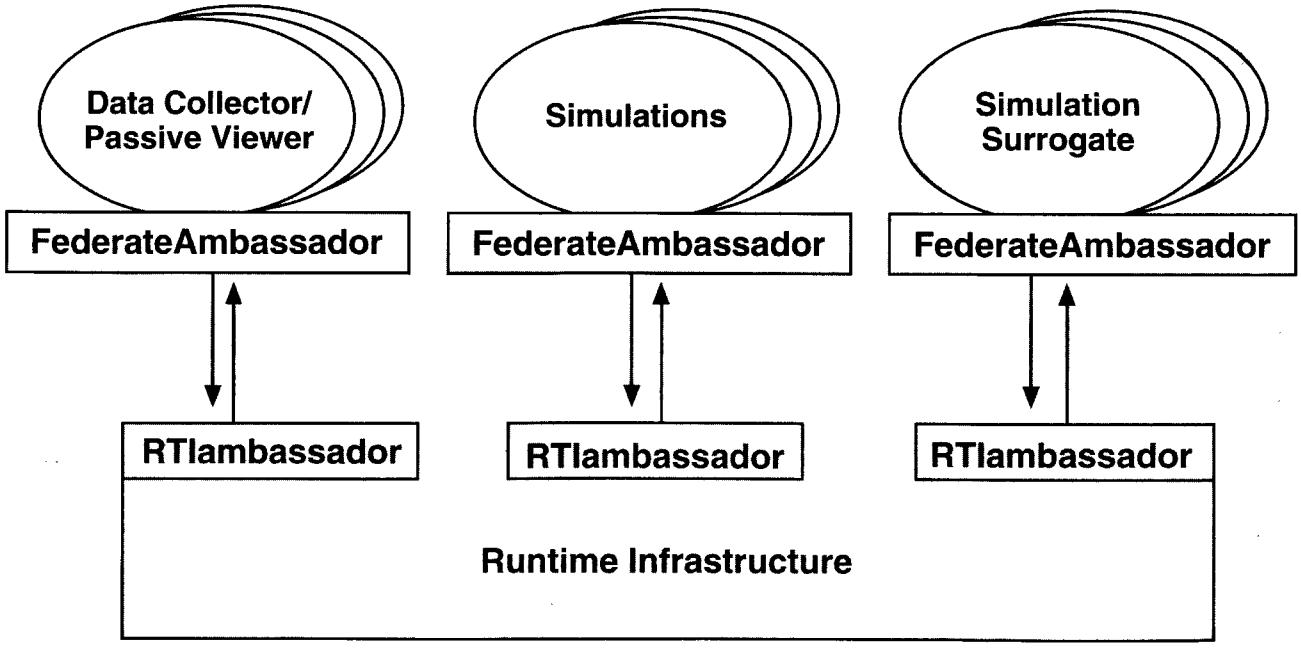


Figure 2. The interface between the RTI and the various kinds of federates.

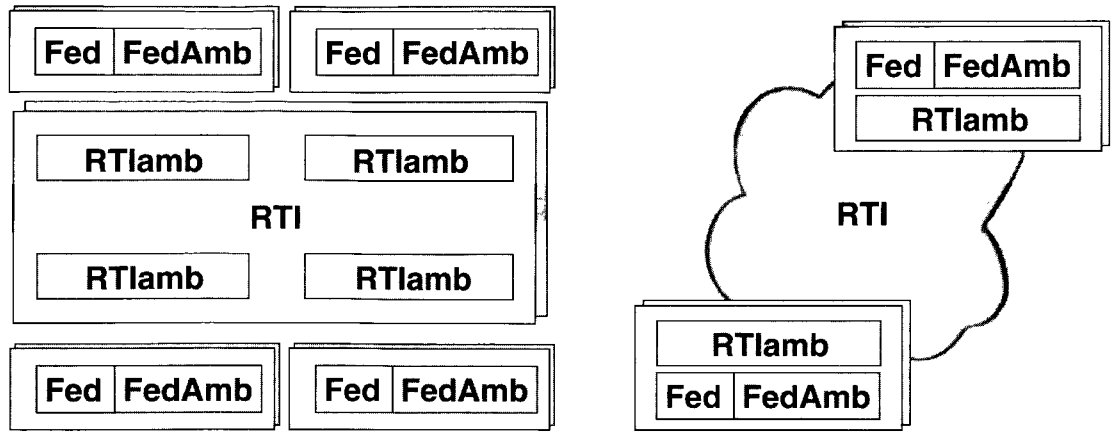


Figure 3. Two of the ways an RTI might be implemented.

services of the RTI, e.g., a request to update the value of an attribute of an object. (These are called federate-initiated services.) Each federate also presents to the RTI an interface called *FederateAmbassador*. The RTI invokes operations on that FederateAmbassador when it must call the federate, e.g., to pass to it a new value of an attribute. (These are called RTI-initiated services.) Thus some RTI services are defined as part of the RTI-ambassador interface, and some are defined as part of the FederateAmbassador interface.

These interfaces take slightly different forms in the APIs defined in the Interface Specification in various programming languages. In C++, RTIambassador is a class with methods corresponding to the federate-initiated services. The FederateAmbassador is an abstract class with abstract methods corresponding to the RTI-initiated services. As part of an RTI implementation, a federate developer receives an implementation of RTIambassador that would be linked into the federate process. The developer also receives the abstract class FederateAmbassador, and is responsible to implement a concrete class derived from it. In the Java API, RTI-ambassador and FederateAmbassador are Java interfaces. An implementation using CORBA defines the interfaces as interfaces in the Object Management Group's Interface Definition Language [2]. An implementation using Ada 95 defines the interfaces as limited private types.

A Federation Has Several Federates But One RTI

A federation contains several federates, but only one RTI. The RTI may take a variety of forms. It is important to remember that the approach taken by one's favorite implementation of the RTI isn't the only one allowed by the definition of the HLA.

Figure 3 presents two (of several) ways in which an RTI might be implemented. On the left, the RTI is a single process, distinct from the processes that contain the federates. Some form of inter-process communication is needed for federates to invoke services on the RTI, and the RTI on federates. At right, the RTI consists partly of local components bound into each federate's process, and partly of other processes or objects. Because each federate interacts with its local RTI component, the shape of the rest of the RTI is irrelevant to it. In addition to these options, one can envision a federation running on a shared-memory multi-processor computer where communication between each federate and the RTI occurs through shared memory regions.

4. HLA's Information Model: the Object Model Template

The Object Model Template prescribes the structure of Simulation and Federation Object Models. Each HLA-compliant simulation has an SOM that describes the data that the prospective federate can produce or consume in a federation. Each federation has an FOM that is the vocabulary of that specific federation. Each execution of a federation uses the federation's FOM. The FOM defines the names of things and occurrences that federates speak to each other about. The FOM does not describe things internal to a single federate, only things that are shared with other federates. Recall that the HLA requires that communication between federates must go through the RTI; the FOM then is the vocabulary of data exchanged through the RTI for an execution of the federation.

The FOM is a parameter to the RTI, in the sense that the FOM is supplied as data to the RTI at the beginning of an execution. The RTI does not change when the FOM is changed or the RTI is applied to another federation. This is a significant feature of the HLA: the designers of each federation are free to adopt their own model, without changes to the RTI or to the HLA standard.

The main components of the Object Model Template are the following:

- Object classes
- Interaction classes
- Routing spaces and their dimensions

Objects: Simulated Entities that Endure. *Objects* in the HLA refer to simulated entities that:

- Are of interest to more than one federate and thus handled by the RTI, and
- Persist or endure for some interval of simulated time.

The OMT defines classes of objects. Each class has a name. Each class defines a (possibly empty) set of named data called *attributes*. Federates create instances of these classes, each of which possesses a separate identity in the federation, and each of which has its distinct instances of its attributes. Federates evolve the state of an object instance in simulation time by supplying new values for its attributes.

Federates converse with the RTI in terms of objects, and, since federates converse with each other through the RTI, federates converse with each other in terms of objects. Each federate must make some translation from its internal notion of simulated entities (whether simulated by that federate or another) to HLA objects as specified in the FOM. If the federate was written with the intention of HLA-compliance, the translation may be very straightforward; if the federate is being adapted to the HLA, the translation may be more involved. The definition of the objects to be handled by the RTI in a federation is (that part of) the federation object model. The FOM represents the common, agreed vocabulary between members of a federation.

Object Classes Form Inheritance Hierarchies

Object classes form a hierarchy. Each object class has exactly one immediate ancestor or superclass. (Object classes form a single-inheritance tree.) The root class is called *ObjectRoot*. The *fully qualified name* of a class is formed by concatenating the class names from the root to the class, separating the names with periods. The fully qualified name of each class must be unique in the FOM.

Each class inherits all the attributes of its superclasses. The set of attributes declared for a class and those inherited from above is called the set of *available attributes*. The root class, ObjectRoot, has one attribute, called *privilegeToDeleteObject*. Consequently, every class has at least one available attribute.

All of the foregoing are illustrated in Figure 4. This figure uses the notation of the Unified Modeling Language [3].

Class and Instance Attributes

When attributes are the subject of an RTI operation, they are either the attributes of an object class, considered as a class, or they are the attributes of a specific object instance:

- The attributes of an object class, considered as a class, are called *class attributes*.
- The attributes of a specific object instance are called *instance attributes*.

Here are some examples. A federate wishing to subscribe to information regarding all instances of a certain object class subscribes to class attributes. A federate wishing to supply a new value for an attribute of a specific instance will update the appropriate instance attribute. The attributes that appear in the object class diagram above are class attributes.

The distinction, with respect to the RTI interface, is more conceptual (but important) than formal: class and instance attributes are referred to by the same identifiers (names or handles). However, RTI services that specify class attributes specify an object class and attribute identifiers, but never identify an object instance; and

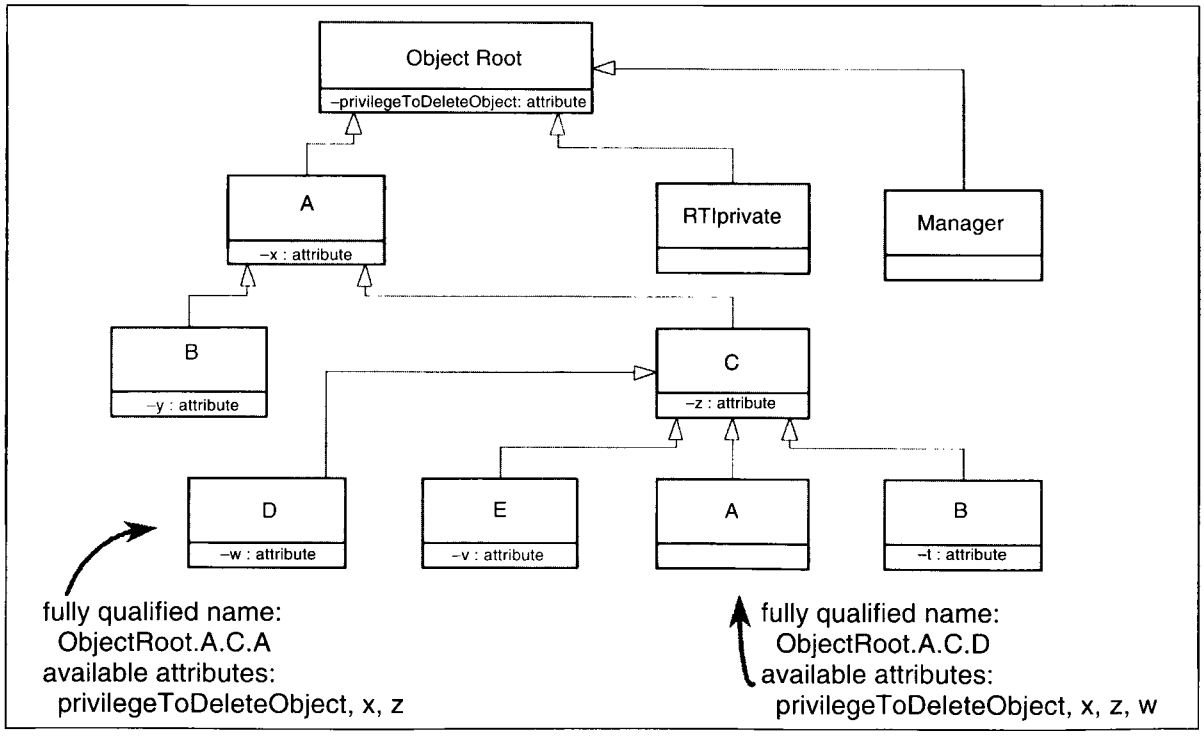


Figure 4. Inheritance hierarchies formed by object classes.

services that take instance attributes always specify an object class and instance and attribute identifiers.

HLA Objects Are Not Object-Oriented

The term "object" as it's used in the HLA may be confusing, because it means something somewhat different from "object" in the usual object-oriented sense. The terms are the same in the following ways:

- HLA objects are instances of object classes, with distinct identity.
- Object classes form a hierarchy, with defined attributes inherited from superclasses.

The terms are different in these ways:

- HLA objects have no behaviors associated with them.
- The "class attribute" notion does *not* correspond to "class variable" (Smalltalk) or "static member" (C++ and Java). In the HLA, a class attribute is an attribute of a class considered as a class and not with respect to an instance. There is no notion of a value for a class attribute.

With respect to implementations, use of the term "object" in the HLA:

- Does not imply that either the RTI or any of the federates must be implemented in an object-oriented language.
- Does not imply that HLA objects exist as object constructs in any API for an object-oriented language. In fact, HLA objects are not "language objects" in the C++ or Java APIs.

Interactions: Instantaneous Occurrences

Whereas objects represent simulated entities that endure for some interval of simulated time, interactions represent simulated occurrences. Interactions do not persist in simulated time: they occur at a point in simulated time. An FOM or SOM defines classes of interactions; when a federate sends an interaction, it is an interaction of a specific class. Some federates send an interaction; other federates receive the interaction. The interaction has no continued existence. Each interaction carries with it a set of parameters, named data similar to attributes.

As with object classes, interaction classes form a single-inheritance hierarchy. The root is called *InteractionRoot*. Each interaction class defines parameters that are sent with it. Each class inherits the parameters defined for all its superclasses. (InteractionRoot defines no parameters by default.) The fully qualified name of an interaction class is defined as for object classes. Each fully qualified name in an FOM must be unique.

Interactions Versus Objects

When to model with objects instead of interactions? In a sense, objects and interactions are interchangeable. Any federation model could be written entirely in terms of objects or entirely in terms of interactions. The change of state of an attribute of an object is an instantaneous occurrence (when the attribute's value is updated), so a designer could always achieve the effect of an interaction by creating attributes whose state changes are noticed and treated as interactions. Similarly, the evolution of the state of an object could be conveyed by a series of interactions that represent changes in the values of attributes. We assert the following general rules:

- If an entity is to be modeled that has persistent state, the entity should be represented as an object.
- If the modeling of an object gives rise to some event or occurrence that is not easily interpreted as a change of state of that object or of another, the occurrence should be modeled as an interaction.

Taken together, objects and interactions represent a very general scheme that can be applied flexibly within a federation design.

Spaces and Dimensions: Routing Data Efficiently

For our tour of the object model to be complete, we must mention routing spaces and dimensions. They are part of every FOM. They are part of a flexible mechanism that allows a federation designer substantial control over the routing of data from one federate to other interested federates. This mechanism is described further in the discussion of RTI data distribution management services.

5. HLA Rules

The HLA Rules are one of the three parts that constitute the HLA standard. They express design goals and constraints on HLA-compliant federates and federations. The benefit for the developer of considering the Rules here is that they summarize the way the HLA is intended to be used. The first five Rules deal with federations; the latter five with federates.

Federation Rules

Rule 1. Federations shall have an HLA federation object model (FOM), documented in accordance with the HLA Object Model Template (OMT).

As said before, the FOM is the agreed vocabulary of a federation. It describes the objects and interactions that one federate exposes to another in the federation. Agreeing on the FOM is an important step in the design of the federation. The OMT requires more information than the RTI needs to operate. The additional information in the template aims to ensure semantic consistency across the federation.

Rule 2. In a federation, all simulation-associated object instance representation shall be in the federates, not in the runtime infrastructure (RTI).

This expresses a crucial design goal of the HLA: the RTI services are generic to component-based simula-

tion and can be used without modification across a variety of domains of application. The same RTI services will support cockpit simulators, manufacturing scheduling simulations and urban traffic analysis tools.

This rule also gives rise to the following design constraint on the RTI: the RTI never stores the values of any attributes, except perhaps transiently. The RTI never functions as a database of present or previous attribute values. This constraint is sometimes oversimplified as "The RTI never keeps state." Any RTI implementation keeps a great deal of state associated with its services, but it is always state related to its services, and not values of instance attributes.

Rule 3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI.

The goal of this Rule is interoperable, reusable simulation components. The infrastructure cannot be bypassed by a compliant federate. This means that the RTI services define completely the interactions between federates. And compliant federates avoid more of the incompatibilities due to private interfaces.

Rule 4. During a federation execution, federates shall interact with the RTI in accordance with the HLA interface specification.

This Rule acknowledges the place of the Interface Specification in the HLA. Not only shall federates interact only through the RTI, their interactions shall observe the Interface Specification. This protects federates from peculiarities in RTI implementations, and makes it more likely that one RTI implementation can be replaced successfully with another.

Rule 5. During a federation execution, an instance attribute shall be owned by at most one federate at any time.

There is an important idea latent in this Rule: to own an instance attribute is to be responsible for updating it, and to be allowed to furnish new values. The Interface Specification makes it clear that, if a federate does not own an instance attribute, its attempt to update its value will be rejected by the RTI. The RTI must also enforce this Rule: at most one federate owns an instance attribute. Rule 8 alludes to the RTI's mechanisms that allow federates to transfer ownership of instance attributes from one federate to another.

Federate Rules

Rule 6. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA OMT.

For a federate, part of HLA compliance is that all its simulation functionality be documented in HLA terms, at least all the functionality that the federate might expose in any federation. That documentation is its SOM. The OMT establishes a minimum set of syntactic and semantic information needed to characterize a federate during the design of a federation. Because a federate is used in several federations, each FOM may include different subsets of that federate's SOM.

Rule 7. Federates shall be able to update and/or reflect any attributes, and send and/or receive interactions, as specified in their SOMs.

This Rule defines HLA compliance with respect to data: attributes and interactions mentioned in the SOM must be fully supported in RTI terms. A federate will initiate the appropriate behavior with the RTI and will respond to RTI-initiated services, with respect to each attribute or interaction in its SOM. A federation designer can expect correct behavior from the federate with respect to any data it offers in its SOM.

Rule 8. Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOMs.

Data in the SOM must be supported not only as to production and consumption, but the federate must also implement its part of the ownership transfer protocols defined in the Interface Specification. A federate may never be willing to transfer ownership of one of its instance attributes; if so, the federate must indicate that in its SOM and must respond correctly to requests to transfer.

Rule 9. Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of attributes, as specified in their SOMs.

The SOM indicates the conditions (passage of time, passage of a threshold in another variable) that cause the federate to update a given attribute. The federate should behave as advertised.

Rule 10. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

This Rule requires a federate to use some set of the time management functions of the RTI to manage its logical time coherently and to allow other federates to manage theirs. The federate may decide not to use any time management services. The RTI supports several different time management schemes and it allows a federation to include federates that vary in their approaches. However, each federate must operate coherently.

6. The RTI Offers Services in Six Areas

Our overview of the HLA concludes with a summary of the functions of the interface, i.e., the services the RTI offers to federates and vice versa. HLA services fall into six groups that are defined by similarity of interest. We'll indicate the interest or concern of each group.

How do federates join a federation?

The federation management services deal with the existence of a federation in two broad categories: (1) definition of a federation execution, as to existence and

membership; and (2) federation-wide operations. Under existence and management, we include services to create a federation execution, and for a federate to join the execution or resign from it. Every federate must join a federation execution, so no federate can completely ignore this group.

Federation-wide operations include the coordination of federation saves (checkpoints) and restores. There are also services to allow a federation to define a federation-wide synchronization point.

How does a federate declare its interest in simulation data?

The HLA is characterized by an implicit-invocation style of data exchange. Federates don't send data to other federates by name; they make it available to the federation, and the RTI ensures its delivery to interested parties. The declaration management services are the way federates declare their intent to produce (publish) or consume (subscribe to) data. The RTI uses these declarations for routing data, pruning data, and interest management.

Regarding routing, the RTI uses subscriptions to decide what federates should be informed of the creation or update of entities. Other publish-and-subscribe mechanisms, like the OMG Event Service, include an administrative interface that allows producers and consumers to indicate their intent. In the RTI, declaration management services fulfill this role.

Declarations are used for pruning inapplicable data when updates are delivered to a subscriber. This is part of the mechanism for protecting federates from extensions to the FOM. The RTI will deliver to a subscriber only those attributes that are defined for the class to which the federate subscribed, and will prune away attributes defined only in a subclass of the class subscribed to.

Finally, the RTI uses declarations to indicate interest to publishing federates. The RTI can tell a federate whether any other federate is subscribed to data it intends to produce, so that it can cease producing when no other federate is receiving.

How does a federate create and update simulation entities?

The object management services are those used for the actual exchange of data. A federate uses services from this group to register new instances of an object class and to update its attributes. Other federates will have services from this group invoked on them to discover new instances and to receive updates of instance attributes. These services are used also to send and receive interactions.

Other services of this group are used in connection with routing spaces. If routing spaces are in use (data distribution management, discussed later), a federate might cease to receive updates regarding an object either because the producer has stopped sending them or because a change in the routing spaces prevents their delivery. There are services to allow the receiving federate to interpret the silence.

How do federates share responsibility for simulating an entity?

In HLA terms, "simulating an entity" means furnishing values for instance attributes. The ownership management services in the RTI implement the HLA's notion of responsibility for simulating an entity. They allow that responsibility to be shared or transferred among federates.

As we said above, HLA Rules 5 and 8 require a federate to own an instance attribute before it can update its value. The RTI ensures that at most one federate at a time owns a given instance attribute. The owning federate is responsible for updating it. Responsibility for simulating an entity can be shared between federates, in two ways.

First, the complete modeling of an entity may be shared among federates. If the entity is represented by an instance with several attributes, different federates may own various attributes of that instance, and thus be responsible for updating the attributes of that instance that they own. Ownership management services support the assumption of ownership to allow this.

Secondly, the modeling of entities may pass from one federate to another in the course of a federation execution. Ownership of an instance attribute may be transferred from one federate to another. Transfers of ownership may be initiated by the present owner or the prospective owner.

Ownership management can be ignored if it is irrelevant to a federation (but see Rule 8). This group is an example of the effort to make the service groups independent: the default behavior regarding ownership of the object management services is reasonable.

How are events ordered in a federation?

In a component-based simulation, where the components may be executing in their own threads of control, the proper ordering of events between federates is a significant problem to be solved. In the HLA, ordering of events is expressed in "logical time." Logical time is an abstract notion: it is not necessarily tied to any representation or unit of time.

The RTI's time management services do two things: (1) they allow each federate to advance its logical time in coordination with other federates; and (2) they control the delivery of time-stamped events such that the federate will never receive events from other federates in its "past," i.e., events with logical times less than its logical time.

The RTI allows a federate to choose the degree to which it participates in time management. A federate may be time-constrained, in which case its advance of logical time is constrained by other federates; or time-regulating, in which case its advance of logical time regulates other federates. A federate may be time-con-

strained and time-regulating, or neither. Federates will make different choices depending on their purposes and the requirements of the federation. The RTI allows federation executions that mix the choices.

How can a federation control the distribution of data?

Simulations with a large number of simulated entities become constrained, as the number of entities grows, by the computation needed to process state updates and the network bandwidth needed to transmit them. The services in this group, data distribution management, form a general scheme for characterizing the production and consumption of data. Their purpose is to allow federates to assign their production of and subscription to data to abstract regions of routing spaces. These assignments allow the Runtime Infrastructure to route data only to interested federates, where interest is defined both in terms of classes of objects and values of attributes. This saves both network bandwidth and receiver computation, and permits the construction of federations with large numbers of entities.

The groups of services are as independent as possible

It is a goal of the design of the HLA services that a federate developer who does not need the functions of a group of services can ignore them safely. The services in one group can be used without reference to another, and the default behavior is reasonable and useful. This goal is realized in the case of ownership, time, and data distribution management. The federation, declaration, and object management groups, however, are always required for the exchange of data.

7. Summary

This paper has described the HLA. Particular attention has focused on the features of the HLA definition that ensure that the HLA incorporates only the essential standards for simulation reuse and interoperability and allows flexibility and extensibility wherever possible. To summarize:

- The HLA standard contains three parts: the Rules, the Object Model Template; and the Interface Specification for the Runtime Infrastructure.
- The goal of HLA is to allow simulation applications to be built from components which themselves are simulations.
- HLA is a software architecture in the common sense. The HLA exhibits several architectural styles: layers, abstract data types, implicit invocation, and support for distribution.
- HLA is architecture, not implementation.
- HLA addresses the persistent problems of component-based simulation; it avoids standardizing transitory technology.
- The object model and standardized interfaces to the RTI insulate federates from change.

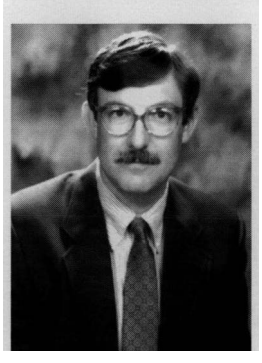
8. References

[1] Shaw, Mary, and Garlan, David. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

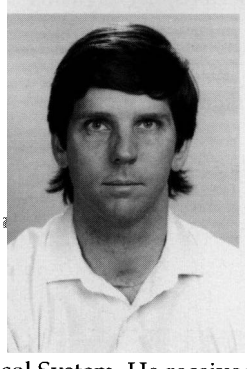
[2] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, July 1996.

[3] Object Management Group. "UML Summary." OMG document number ad/97-08-03.

Judith Dahmann has been the Chief Scientist for the Defense Modeling and Simulation Office of the U.S. Department of Defense since 1995. The DMSO sets policy and provides tools for modeling and simulation throughout the DoD. As Chief Scientist, Dr. Dahmann oversees the evolution, acceptance and application of the HLA. Before joining the DoD, she spent more than 20 years at The MITRE Corporation, where she led, among other projects, the development of the Aggregate Level Simulation Protocol, a precursor of the HLA. She holds an MS from the University of Chicago and a PhD from Johns Hopkins University.



Frederick Kuhl is a Senior Principal Engineer with The MITRE Corporation. He currently leads the international standardization effort for the HLA, and has been a leader in the field since the beginning of the activities to prototype implementations of the HLA infrastructure. Dr. Kuhl has applied object-oriented programming to prototype air traffic control systems, and distributed-object computer technology to distributed simulation. He earned his PhD in Computer Science from Texas A&M University.



Richard Weatherly is the Chief Engineer of The MITRE Corporation's Information Systems and Technology Division, where he leads the company's HLA infrastructure development and verification team. He is a core member of the HLA interface specification team and has chaired working groups that designed most of the specification. Prior to that, Dr. Weatherly was the Project Leader and Designer of the Aggregate Level Simulation Protocol System. He received his PhD in Electrical Engineering from Clemson University.