Abstract: In this paper, we address the problem of improving the design and documentation of distributed simulation environments by using the Unified Modeling Language (UML). We open our presentation with a brief introduction to the UML and lay the foundation for the remainder of our discussion by including a description of its diagrams, design processes, and documentation capabilities. We then discuss the aspects of the UML that complement the documentation developed using the High Level Architecture (HLA) and describe how the UML and HLA Federation Object Model (FOM) design process and documentation can be combined in a complementary and mutually reinforcing manner to achieve a better design and documentation capability for distributed simulations and HLA federations.

1. INTRODUCTION

The design and documentation of simulation environments, even those based upon the High Level Architecture (HLA), remains a challenge for simulation developers, requirements analysts, and users. Even though one of the activities required to achieve HLA compliance for an application is the development of a Federation Object Model (FOM), a FOM is not a complete, comprehensive, or easy to comprehend set of documentation of a simulation environment. The FOM development process also does not adequately support simulation environment design. While the Federation Development Process (FEDEP) does provide some assistance in the design and documentation effort; however, it is not complete and the FEDEP is unfamiliar to the majority of users who are not fluent in the HLA development process. The limitation of the FOM is that it only documents information about an application, such as the types of actors supported, the information that each actor is able to export to the distributed virtual environment (DVE), the information each actor requires from the DVE, and the types of interactions that each actor supports. A FOM also only marginally specifies the required capabilities of each actor and gives the federation designer and user a limited view into the data, parameters, and interactions required by and for each DVE actor. A FOM serves to document the data that must be transmitted within a distributed simulation from a single point of view, the distributed virtual environment actor. The FOM does not document any other aspects of the distributed simulation environment, such as its expected uses, implementation, or interfaces, nor does it allow the designer or developer to assess the distributed simulation environment from different points of view. Potentially useful additional points of view include a structural view, a behavioral view, an implementation view, an environment view, or a user view.

The specification of the FOM for a distributed simulation environment, or federation, requires the definition of an Object Class Structure Table, an Interaction Class Structure Table, an Object Class Attribute Table, and an Interaction Class Parameter Table. In our opinion, these tables do not provide a comprehensive set of documentation of the capabilities required of and provided by a federation. This shortfall is most noticeable when a federation is being designed and evaluated with non-simulation expert users. The limitations of the HLA FOM coupled with the need for increasingly complex distributed simulation environments indicate that the FOM should be augmented with additional information. In this paper, we discuss how the Unified Modeling Language (UML) can be used to satisfy this need.

The Unified Modeling Language is a standard graphical language for developing and composing blueprints (architecture specifications) for software systems. UML is a modeling language for the conceptual and physical representations of a system and it permits models of a system to be developed from a variety of perspectives. UML provides a complete language for capturing the knowledge about a subject and for expressing the knowledge, contains a large and useful set of predefined constructs, and has inherent mechanisms for extensibility. UML also provides a means for visualizing the components of a system and for visualizing the complete system from a variety of perspectives. UML provides constructs for specifying and documenting the building blocks of a system as well as for documenting a complete system. UML inherently provides support for modern or abstract software concepts such as collaborations, frameworks, design patterns, and components and allows a software modeler to model classes, objects, sequencing, constraints, and packages as well as components. The vocabulary of the UML is built from three types of building blocks, things, relationships, and diagrams, which we will discuss in Section Two.

In our view, the use of a standardized set of UML documentation for simulation systems and federations will enable better management of the simulation environment development process and will improve the description of distributed simulation capabilities. A standardized set of
UML documentation can also provide an improved capability to integrate, exploit, and incorporate advances made in simulation technologies. UML-based documentation can aid in migration to increased capability computer systems and help describe system capabilities to non-technical simulation users. Additionally, in our opinion the simulation community needs a vastly improved capability to document a distributed simulation environment’s requirements, functionality, and capabilities and that this documentation should be assembled using a standardized modeling language. Therefore, we advocate development of UML documented use cases for all current and anticipated federations so that their complete set of requirements, functionality, and capabilities are captured.

In this paper we discuss how the UML can be used to improve the FOM development process and the documentation for distributed simulation environments. In the next section we present a background on the HLA and the UML. In Section Three we describe how the UML should be used in conjunction with the existing HLA and FOM development processes. Section Four contains a summary and suggestions for future work.

2. HLA AND UML BACKGROUND

We begin this section we present short introduction to the High Level Architecture, its concepts, and its documentation. The second half of this section presents an introduction to the Unified Modeling Language, its concepts, its diagrams, and its approach to modeling.

2.1 Introduction to the High Level Architecture

The High Level Architecture (HLA), along with the Distributed Interactive Simulation (DIS) protocols, form the foundation for entity-to-entity communication within most modern military distributed simulation environments. The High Level Architecture was formulated to support the goal of constructing federations from modular simulation components by using a general-purpose communication infrastructure. When developing a HLA-based simulation, a central architectural decision was the separation of application functions from communications functions. In HLA, application functionality is the responsibility of the application’s software and is hidden from the communication software. In general, all communication functions are managed by a HLA standardized software system, the Runtime Infrastructure (RTI). The RTI manages communication paths between executing applications and insures that the application acquires the data that it needs (via subscription) and that the application provides the data that other applications request (via publication). The foundational papers for HLA are in the 15th Workshop on Standards for the Interoperability of Distributed Simulations (6, 8, 9, 19).

As part of the activities required to achieve HLA compliance for an application, a Federation Object Model (FOM) must be designed for the complete simulation, called a federation, and a Simulation Object Model (SOM) must be defined for each application, which is called a federate. The FOM documents the entities involved in the federation, the types of interactions required, and the parameters/variables that must be transmitted for each type of interaction. The SOM documents key information about an application, such as the types of actors supported, the information that each actor can export (publish) to the DVE, the information each actor requires from the DVE (subscribes to), and the types of interactions that each actor supports. The SOM also gives a federation designer a view into the data, parameters, and interactions required for each actor application. The specification of a FOM or a SOM for any application requires the definition of an Object Class Structure Table, an Interaction Class Structure Table, an Attribute Table, and a Parameter Table. While the names of the tables used to build a SOM or a FOM are identical, the scope of the tables is vastly different depending upon whether a FOM or a SOM is being assembled. A SOM documents only the information for a single entity in a distributed simulation environment, a FOM documents the information for every entity in the simulation environment, or federation.

The Object Class Structure Table hierarchically defines the classes of DVE actors that a federation or federate can support and the remote DVE actor classes whose instances and attributes contain potentially useful information for the federate. These classes of local and remote actors are defined by specifying the hierarchical relationships between the various classes in the HLA application. All of the actors that participate in a federation are specified in this table.

The second table required in a FOM or SOM is the Interaction Class Structure Table. An interaction is defined as an action that an actor can perform that can affect another actor in the federation. The Interaction Class Structure Table specifies the types of interactions in which each actor in a federation or federate can participate. As in the Object Class Structure Table, the interactions are defined hierarchically; thereby, enabling inheritance to be used to specify interactions that are common to classes of actors. An important specification in the table is the interaction type supported by each interaction class. Each interaction can be one that an actor can initiate, sense, react to, or ignore.

The third table required in a FOM or SOM is the Attribute Table. An attribute is a named portion of a class of actor’s state whose values can change over time. The Attribute Table documents the data produced and consumed by each class of actor in the application. The individual object attributes defined in this table can be subscribed to by a remote actor or published by an actor to the rest of the federation by a federate. By subscribing to a class or a subclass, a remote federate will receive only the information requested, thereby conserving network bandwidth and processing power.
Conversely, by specifying the information that the federate requires to execute, network bandwidth and remote host processing is also conserved because unneeded information is not generated or transmitted. Updates to the class attributes, as defined in the FOM or SOM Attribute Table, are the data that actually flow between federates in a federation (DVE). The attributes for the classes are defined hierarchically and data types from superclasses are inherited by its subclasses. The fourth required table in developing a FOM or a SOM is the Interaction Parameters Table. The Interaction Parameters Table defines each generic interaction and specific interaction that is required to be supported by a federation or federate and the attributes that must be transmitted to perform the interaction.

2.2 The Unified Modeling Language
The Unified Modeling Language (UML) is a standard graphical language for developing and composing software blueprints for software systems (1, 2, 3, 4, 5, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21). UML focuses on the conceptual and physical representation of a system and permits models of a system to be developed from a variety of perspectives. The UML provides a complete language for capturing and expressing knowledge about a subject, contains a large, useful set of predefined constructs, and is extensible. UML provides a means for visualizing the components of a system, for specifying and documenting the building blocks of a system, for visualizing the system, and for documenting a system. UML also provides support for new high-level software concepts such as collaborations, frameworks, design patterns, and components and allows a software modeler to model classes, objects, sequencing, constraints, and packages as well as components.

The vocabulary of the UML consists of three types of building blocks: things, relationships, and diagrams. Things are the major abstractions in the model, relationships tie things together, and diagrams portray aspects of the model. There are four types of things: structural, behavioral, grouping, and annotational. In addition, there are seven types of structural things: classes, interfaces, collaborations, use cases, active classes, components, and nodes. Behavioral things are the dynamic parts of the model and there are two primary types of behavioral things: interactions and state machines. Grouping things are the organizational parts of the UML, and are described using packages. Finally, annotational things are the explanatory parts of the UML and are comments applied to a model to describe or remark about an element.

There are four types of relationships in UML: dependency, association, generalization, and realization. A dependency is a semantic relationship between two things wherein a change to one thing causes a change to the other thing. An association is a structural relationship that describes the connections (links) between things/objects. A generalization is a relationship wherein the objects of the specialized element are substitutable for objects of the generalized element. Finally, a realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to execute. These building blocks are used to build the UML diagrams for a system.

In the UML, there are nine types of diagrams: class, object, use case, sequence, collaboration, statechart, activity, component, and deployment. A class diagram shows a set of classes, interfaces, and collaborations and presents the static design view of the system. An object diagram shows a set of objects and their relationships, and simply presents a snapshot of a class diagram while the objects in it are performing a particular activity. A use case diagram shows a set of use cases and the actors involved in the case(s). In general, a use case shows how the system and a number of users/actors and/or classes interact to accomplish a task. Interaction diagrams, which can be either sequence or collaboration diagrams, show a set of objects, relationships, and the messages passed between them. An interaction diagram presents a view of the system in operation. A statechart diagram presents a view of the system as a state machine. A statechart diagram is used to describe the event-ordered behavior of a system. An activity diagram shows the flow from activity to activity within a system. A component diagram presents the organizations and dependencies among the classes, interfaces, and collaborations in a system as it has been implemented. A deployment diagram shows how the system software is allocated among the computer systems.

By using the diagrams discussed above, UML provides the means for viewing and analyzing an architecture from five points of view. These five points of view are the design view, the use case view, the process view, the implementation view, and the deployment view. The use case view describes the behavior of the system from the point of view of the users/actors; in other words, the view describes the actions and activities that the user can “observe.” The use case view is described using use case diagrams. The design view is comprised of the classes, interfaces, and collaborations in a system. The design view shows how the functional requirements of a system are met. The design view is presented using interaction, statechart, and activity diagrams. The process view describes the threads and processes that comprise the system and primarily focuses on performance and scalability. The process view uses the same diagrams as the design view. The implementation view of a system defines the components and files that are used to assemble the system (or federation in the HLA). The deployment view presents the system's software and hardware topology. The components within a system are typically documented within the component and deployment diagrams.
In the next section we discuss how the components of the UML can be used to augment the current HLA documentation and to improve the design and development of HLA-based simulation environments. Our discussion is only the start of a process. In the end, our discussion should be combined with the thoughts of others to arrive at a joint determination of how the HLA standard should be revised to best exploit UML.

3. COMBINING THE UML WITH THE HLA

The approach that we propose for exploiting the faculties of the UML to develop HLA-based distributed simulations is straightforward in concept and in execution. In our approach, the UML is used to design the simulation environment and to document the behaviors and properties that the environment must possess. This UML-produced information is then used to complete the four FOM and SOM tables. The four tables are then used as the basis for further refinement of the UML-based design of the simulation environment. The design-document-refine cycle is continued until the design stabilizes. Execution of the environment may also uncover additional requirements or gaps in the design. Therefore, we advocate continued design and documentation refinement based upon the results of simulation executions. Clearly, the design-document-refine process never really concludes. Rather, in our view, the process gradually moves from a theoretical, abstract, initial requirements driven process to one that is based upon the analysis of the results of executing the federation simulation environment. The remaining portion of this section is devoted to explaining our approach in more detail and to clarifying the crucial role that the UML can play in the design and documentation of a distributed simulation environment.

The approach we advocate is to define and design the federation simulation environment using UML, the HLA FOM is produced as one of the artifacts of the UML design process. The process is presented in Figure 1. For the sake of clarity in the diagram the iteration arcs have been omitted; in practice the process requires iteration and feedback between each of the phases of the development of UML and between UML and FOM documentation and design. The flow of the process is from the identification of things, chiefly behavioral things, to the description of relationships between the things. The centerpiece of our approach is the use of UML as the foundation for the design and development of the FOM and federation simulation environment because UML permits users and experts to readily evaluate the products of the process. We believe that users can better evaluate the design by examining a visualization of the environment (from one or several viewpoints) and then deriving the FOM from the UML-based description.

The process has as its first step the definition of the requirements for the simulation environment. With the requirements in hand, the next step is the definition of the behaviors for the components (entities) in the simulation environment. These behavior definitions are captured in the use case documentation and the resulting use case diagrams.

We believe that the development of use case documentation and use case diagrams should be the next step in the process because behaviors are more easily derived from the requirements than are the other UML components. Because the definition of the behaviors is so crucial to the design process, we recommend the commitment of significant effort and resources to the definition of the use cases and the development of the use case diagrams. The use cases define the types of behaviors (capabilities) that the distributed simulation environment must exhibit. Once the initial set of use cases have been defined, the use case diagrams are developed. The use case diagrams display the required distributed simulation environment behaviors in a manner that allows an expert, a user or other non-expert to verify their accuracy and to determine if all of the necessary capabilities have been captured.

Based upon the use cases that are developed, a solid conception of the dependencies, objects, and interactions that the environment requires can be developed. This knowledge is then captured in component and class diagrams and in the two types of interaction diagrams. The interaction diagrams should be augmented with annotations for performance specifications for the interactions and the environment. These annotations should include real-time performance limits, latency limits, frequency of information updates, accuracy requirements for the transmitted data, and dead-reckoning formula(s) to be used.
The information in the components and classes diagrams is used to assemble the Object Class Structure Table, as these diagrams identify the chief participants in the distributed simulation environment. By using the information concerning data to be exchanged captured in the Object Class Structure Table and the collaboration diagrams, the Object Class Attribute Table can then be built. The two types of interaction diagrams along with the collaboration diagrams are the main sources of information concerning the types of interactions that must be reported in the Interaction Table. The publish and subscription needs for each object can be determined directly from these same diagrams as well. Finally, by using the information contained in the interaction diagrams, the Interaction Table, and the Object Class Attribute Table, the Interaction Class Parameters Table is assembled. The interaction diagrams and the Interaction Table determine the interactions to be reported, while the Object Class Attribute Table defines the data available to be exchanged during an interaction.

We believe that the diagrams and the tables should be developed in an iterative, top-down manner. We believe that a top-down approach is best in light of the priority for developing the simulation environment to meet the demands of the requirements and the specifications of the use cases. The alternative is to develop the simulation environment to maximize the amount of existing capabilities that are re-used to achieve the desired simulation environment.

Figure 1: Combining the Unified Modeling Language and the High Level Architecture Federation Object Model Processes

This approach effectively gives the system requirements a priority that is too low. Of course, re-use should be a consideration in the development of the simulation environment, but in our opinion re-use should follow upon the determination of requirements and not lead it. Finally, the other UML diagrams that we have not mentioned in conjunction with the FOM and the development process for the federation simulation environment, such as the deployment diagrams, should be used at the very end of the process since their information is tightly bound to a particular implementation of a distributed simulation environment. These diagrams effectively document the particulars for a given implementation, but have little value to add in the design and development phases for a FOM and to its associated simulation environment.

4. SUMMARY AND FUTURE WORK

In this paper we discussed how the UML can be used to improve the FOM and federation simulation environment development and documentation process. We presented a brief background and introduction to the HLA and the UML and described how the UML can be used in conjunction with the existing HLA and FOM development processes to design and document a federation. In brief, we advocate that the main effort of the design and development process be accomplished using the tools provided by the UML and that the FOM documentation should be derived directly from the UML products. As regards future work, we believe that the most important matter is the development of extensions to the UML that will support a more formalized and standardized description of the timing-related, performance, accuracy, and dead-reckoning requirements for a federation simulation environment.
REFERENCES


