A Component Architecture for Simulator Development[1]

Nico Kuijpers
Paul van Gool
Hans Jense
TNO Physics and Electronics Laboratory
P.O. Box 96864
2509 JG, The Hague
The Netherlands
kuijpers@fel.tno.nl, vangool@fel.tno.nl, jense@fel.tno.nl

Keywords: simulator architecture, simulator components, HLA-RTI, RD&E applications

ABSTRACT: *Research, Development and Engineering applications require the rapid development of simulators, preferably through the reuse of simulator components. In order to facilitate the reuse and exchange of simulator components, research institutes and industry in The Netherlands are collaborating in the SIMULTAAN project. The main result of this project will be a simulator component architecture that facilitates 'plug & play' with components to build a simulator. Another result will be a component repository, which facilitates the exchange of simulator components.*
The realization of the simulator architecture is the Run-time Communication Infrastructure (RCI). The application programmer is shielded from the complexity of simulator interoperability standards. In addition to inter-federate communication, the RCI will also provide inter-component communication. If performance requirements are met, inter-component communication will be based on an available HLA-RTI, otherwise a dedicated RTI will be developed.

1. Introduction

When developing a simulator, several basic components are combined to provide the simulator's operator with a virtual representation of real-world dynamics and the real-world environment. Recurring stages of simulator development are: requirements analysis, design, implementation, integration and testing of the system. The simulator's functionality is usually divided amongst several components (e.g. visual, motion and dynamic model). The interfaces used to integrate the components are usually based on per-project requirements. This limits the reusability of the components. An architecture that supports the reuse and exchange of components is proposed in this paper.

The simulator component architecture is intended to maximize the reuse potential of components by defining a standard interface for simulator components. In this way simulator development time will be reduced. By making sure that components comply to the standard interface, and comply to a number of rules, they can be reused in another simulator built on the same architecture. The common architecture will be used in an RD&E environment where the rapid reconfigurability of simulators is essential. However, it can also be used in an industrial environment.

In order to meet the requirements with respect to flexibility and rapid prototyping, an object-oriented, layered architecture is developed. Applications will be built on top of the so-called Run-time Communication Infrastructure (RCI). The RCI provides the SIMULTAAN developer with an abstraction layer (or middle-ware) which shields the developer from the underlying interoperability standards.

To further encourage the reuse and exchange of simulators and components, the HLA Object Model Templates [1] will be used to describe the intrinsic capabilities of SIMULTAAN components and federates.

The paper is organized as follows. Section 2 describes the current simulator development in The Netherlands. The SIMULTAAN project and its goals are presented in Section 3. Section 4 describes the development of simulators based on the results of the SIMULTAAN project. Section 5 introduces the SIMULTAAN Object Models. The objectives and the architecture of the RCI are presented in Section 6. Finally, conclusions are drawn in Section 7.

[1] This work has been carried out in the framework of the SIMULTAAN project which is partly funded by the Dutch initiative for High Performance Computing and Networking (HPCN).

2. Current Simulator Development

Within The Netherlands the SIMULTAAN consortium has the knowledge and capabilities to develop full-mission simulators for both military and civil use. However, knowledge and disciplines are spread amongst the consortium members which causes delay and repetition during design of the system and the integration of subsystems.

Negotiations are necessary per project to achieve agreements between participating companies and institutes. Work is in general defined at subsystem level. The simulator architecture and the interfaces between the different components and subsystems have to be defined case by case. Partly because of ad-hoc interface definitions, reuse of available hardware and software is limited. Usually modifications are necessary to meet the newly defined interface definitions. Thus a considerable amount of work is needed for requirements analysis, design, implementation, integration and testing of the system.

Clustering of knowledge and experience in The Netherlands is necessary to compete with the established international simulator industry. The SIMULTAAN project was defined to provide the knowledge clustering and intelligence infrastructure for simulator development.

3. The SIMULTAAN Project

SIMULTAAN is a 2.5 year project, started in January 1997, which brings together knowledge and experience in the area of simulators and distributed simulation from universities, research institutes and industry in The Netherlands. The project is partly funded by the Dutch initiative for High Performance Computing and Networking (HPCN). The six consortium members are

- TNO Physics and Electronics Laboratory (project leader);
- National Aerospace Laboratory NLR;
- Delft University of Technology, Faculty of Aerospace Engineering;
- Siemens Netherlands NV;
- Fokker Space BV;
- Hydraudyne Systems & Engineering BV.

SIMULTAAN aims at the development of a generic simulator architecture. This architecture will be the basis for federations consisting of several simulators and appropriate scenario management facilities. The SIMULTAAN results will be demonstrated at the end of the project by realizing one representative federation. The project results will be applicable to future simulator projects.

A SIMULTAAN federate consists of a number of components; one of them is the federate manager, which controls the operation of the federate. Examples of components commonly found in simulators are a visual system, a motion system, a dynamic model, and a mock-up server.

SIMULTAAN will facilitate the interoperability of federates and components by providing a standard interface to the application programmer. Two types of communication can be distinguished: communication between components (local communication) and communication between federates (global communication). Both types of communication will be supported by the architecture. For the communication between federates international standards for simulator interoperability will be used. This means the SIMULTAAN architecture will support both DIS and HLA by providing one common interface.

Four results of the project can be distinguished:

1. The SIMULTAAN Simulator Architecture (SSA) which is the common high-level architecture for simulators and tools developed by the SIMULTAAN partners.
2. The Run-time Communication Infrastructure (RCI) which is an implementation of the SSA. The RCI will be discussed in Section 6.
3. A set of components and federates compliant with the SSA standards and developed for the purpose of the SIMULTAAN Demonstration.
4. The SIMULTAAN Object Repository (SOR) which is part of the infrastructure to reuse and exchange components and federates developed by the SIMULTAAN partners.

The SIMULTAAN Simulator Architecture (SSA) will facilitate interoperability between the components of a federate and between federates of a federation. The SSA includes the SSA Rules, the SSA Interface Specification and the SSA Object Model Templates.

The SSA Rules are rules with which a SIMULTAAN federate or component has to comply. They define the responsibilities and relationships in a SIMULTAAN federation. The SSA Interface Specification (SSA-IF) is a formal, functional description of the interface between the application and the Run-time Communication Infrastructure (RCI). The SSA Object Model Templates (SSA-OMT) are standardized formats to define the functionality of federates and components and their respective interactions. The SSA-

OMT will be equivalent to the HLA-OMT [1].

The Run-time Communication Infrastructure provides the run-time interface services for communication between components in a federate and between federates in a federation, according to the SSA Interface Specification [2]. The RCI can be regarded as middle-ware, which will hide the complexities of the underlying interoperability standards from the SIMULTAAN component or federate developer. This way component development time can be reduced.

Another result of the project is the SIMULTAAN Object Repository (SOR). The SOR will contain SSA compliant simulators, components and tools. It may also contain configuration, initialization, and validation data for components, simulators, and tools. In this way SIMULTAAN will support the process of simulator development (e.g., requirements analysis, design, composition and validation). The SOR allows controlled access by the SIMULTAAN partners.

4. SIMULTAAN Federate Development

SIMULTAAN Federate Development describes the way SIMULTAAN partners will produce federates using the results of the project. New components may have to be developed or existing components may need to be adapted. During the design process, such needs will be identified and translated to component requirements. The Federate Development process will result in a validated simulator or tool. A federation can be created by producing its federates and defining the interactions between them in a Federation Object Model (FOM).

User requirements for the federate are specified in cooperation with the end-user and can be regarded as a starting point for the development. From the user requirements, the system requirements are identified. The system requirements initiate the design process of the federate.

The federate will be designed with optimal use of existing components. Therefore access is needed to the descriptions of components that are available in the SIMULTAAN Object Repository (SOR). When all components are available, the developers can build the federate. For validation, the federate is tested against the requirements and demonstrated to the end user.

5. SSA Object Model Templates

In HLA, object models are used to describe an object's intrinsic capabilities [1]. In order to facilitate the reuse and exchange of components and federates, the following object models have been identified for SIMULTAAN.

- Each component will have a Component Object Model (COM). This object model formally specifies the attributes and interactions a component publishes to other components. It also specifies the attributes and interactions a component will subscribe to during run-time.
- A Simulator Component Object Model (SCOM) formally specifies all interactions and attributes between the components of one federate. Using this model it can be determined whether all subscriptions are actually published.
- The Simulator Object Model (SOM) formally specifies the attributes and interactions a federate publishes to the federation. It also specifies the attributes and interactions a federate will subscribe to during run-time. The SIMULTAAN SOM is equivalent to the HLA SOM.
- The Federation Object Model (FOM) formally specifies all interactions and attributes within the federation. Using this model it can be determined whether all subscriptions are actually published. The SIMULTAAN FOM is equivalent to the HLA FOM.

Distinction between COM and SCOM on one hand, and SOM and FOM on the other, enables different treatment of local and global communication. Local communication is the exchange of information between components in a federate, whereas global communication is the exchange of information between federates.

The SIMULTAAN object models enable clear specifications for the capabilities of federates and components. Federate and federation development in SIMULTAAN can be compared to the HLA Federation Development and Execution Process (FEDEP) [3].

6. Run-time Communication Infrastructure

The SIMULTAAN Simulator Architecture (SSA) is the common high-level architecture for simulators and tools. The SSA will provide services to both the components and the federate. Components are 'glued' together to form an aggregate federate. The Run-time Communication Infrastructure (RCI) is the implementation of the SSA services.

Components and federates are different in nature and have their specific requirements on the SIMULTAAN Simulator Architecture (SSA). For example,

communication between components is often local within one site, point-to-point and optimized with some dedicated protocol (e.g. reflective shared memory like SCRAMNET). Communication between federates is between sites, one-to-many and less optimized. Furthermore, communication between federates requires compliance with standards for simulator interoperability. The SSA provides an architecture where the different requirements are united in one solution.

As described in the previous sections, the SIMULTAAN Simulator Architecture (SSA) will shield the interoperability standards from the component or federate developer by presenting an abstraction layer (or middle-ware). The RCI will provide the component developer with the necessary functionality to incorporate the component into a SIMULTAAN federate. The RCI is a protocol-independent interface to the simulated environment. The design of the abstraction layer and the Application Programmer's Interface (API) are discussed in this section.

The design of the RCI is mainly inspired by HLA and one of its objectives is to enable the migration from DIS to HLA with minimal changes. The object-oriented design of the RCI promotes both the reuse of existing software components and facilitates the extension of the RCI itself.

For a SIMULTAAN component, two simulation environments can be identified:

1. The environment inside a federate: This environment consists of a set of collaborating components. It presents an overview of the other components within the federate.
2. The environment outside a federate: This environment represents the federation. It presents an overview of all simulated entities that are part of the federation.

The SIMULTAAN environment will combine the two environments. It will give a component an overview of the other components in its federate and an overview of the simulated entities in the federation. The top-level object model [4] of the RCI is shown in Figure 1.

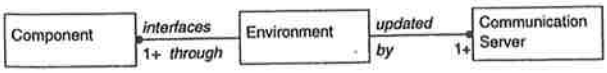

Figure 1: RCI Top Level Object Model

Environment:
The SIMULTAAN Environment will provide components an overview of both the federate and the federation. The Environment will reflect the current state of the federate, i.e., the state of all its components. It will allow the addition and deletion of components. Furthermore, it will allow components to subscribe to relevant information. Each component can publish data to which other components of the federate may subscribe. Components can send and receive events. The translation of the events to a specific interoperability standard (such as DIS or HLA) is left to the Communication Server.

Component:
A Component is the basic building block for a SIMULTAAN federate. The interface between the Component and the Environment is the only interface the component developer will have to deal with.

Communication Server:
The Communication Server represents the object that takes care of the actual communication. Its function can be compared to that of the HLA-RTI. In a way it represents a distributed operating system. The interface between the Environment and the Communication Server will be based, to some extent, on the HLA Interface Specification [2]. A Communication Server will communicate with other Communication Servers. Three specialisations of the Communication Server are shown in Figure 2. Notice that it is possible to have multiple Communication Servers within the RCI, which enables the use of a different Communication Server for local and global communication. To this end the distinction between COM/SCOM and SOM/FOM is made, as mentioned in Section 5.

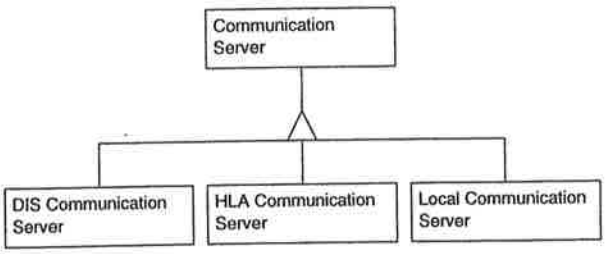

Figure 2: Communication Server Inheritance Diagram

DIS Communication Server:
A specialisation of the Communication Server that will translate all events to DIS PDUs, and vice versa. This type of server will only be used for inter-federate communication.

HLA Communication Server:
A specialisation of the Communication Server that will translate all events to HLA interactions and HLA attributes, and vice versa. This type of server will be used for inter-federate communication and perhaps for inter-component communication, if the HLA-RTI provides sufficient performance.

Local Communication Server:
A specialisation of the Communication Server that will be used for inter-component communication. The Local Communication Server can be used for inter-component communication over a dedicated communication medium such as reflective shared memory (SCRAMNET).

6.1 Environment

As shown in Figure 1 components communicate exclusively with the environment. The environment class provides a view of the 'world', as seen by a component.

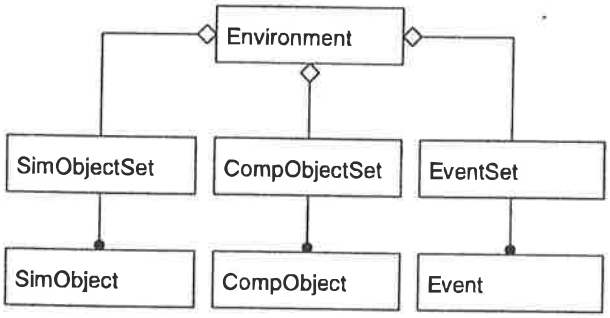

Figure 3: Environment Object Model

SimObjectSet:
Set of simulated entities relevant to the federation.

SimObject:
The state of a simulated entity.

CompObjectSet:
Set of component objects relevant to the federate.

CompObject:
The state of a SIMULTAAN component.

EventSet:
Set of simulation events, such as simulation management events and other unique occurrences in the simulation session.

Event:
A unique occurrence in the session, e.g., an object state update, simulator management requests, component creation.

The Environment will also supply components with functionality to create and destroy a federate. Creation and destruction of a federate will be done by a federate manager. Once a federate has been created, other components can join the federate.

The Environment provides a publish and subscribe mechanism. Components may publish object- and event-classes. This means that during the federation execution, these types of objects and events will be created by the component. Components may also unpublish, which means that objects and events will no longer be provided by the component.

Subscription possibilities will be offered by the Environment. Components can indicate an interest in other objects or events. The creation of attribute sets, as needed by the HLA-RTI, will not have to be done by the developer. The RCI will perform these tasks. A component can specify its response to events by defining callbacks. It is possible to specify multiple callbacks to an event.

The environment class will supply methods to create and delete objects. Further methods include the possibility to send events, and retrieve the time. Events can be sent to the entire federation, a specific federate or a specific component within the federate. The Environment class is completely protocol-independent and therefore entirely reusable.

6.2 Communication Server

The Communication Server represents the underlying interoperability standard. Its main goal is to exchange object and event information with other Communication Servers and keep the Environment up-to-date. The Communication Server will be partially protocol-dependent and therefore each interoperability standard will require its own Communication Server.

The interface between Environment and Communication Server will be based on the HLA Interface Specification to establish compatibility with HLA [2]. Therefore, the design of the Communication Server subsystem will include classes that correspond with RTI Management Services. Note that the Communication Server is nothing more than a facade for these management services [5].

The communication server functionality is divided into various subsystems to reduce complexity. The

Communication Server object model [4] is shown in Figure 4.

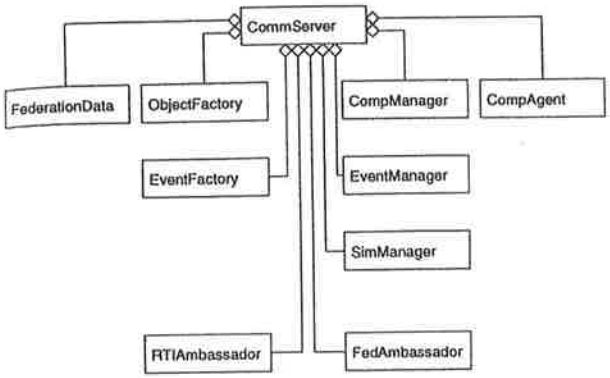

Figure 4: Communication Server Object Model

The FedAmbassador and RTIAmbassador classes provide the connection to the HLA-RTI. The other classes identified in Figure 4 will be discussed below.

FederationData:
The FederationData object will parse the SCOM and FOM in order to determine all possible components that may be instantiated and all possible events that may be sent during the execution of the federate.

The SCOM and FOM will be a symbolic representation of the components and events (and their attributes and parameters, respectively). The FederationData class will provide a mapping of these symbolic names and the RCI handles for them.

EventFactory:
Events are the only means for a component to communicate. Events may contain data. The EventFactory class will be used to convert the protocol-independent events to HLA interactions, DIS PDUs, or some other communication protocol, and vice versa. This class has a close relation with the FederationData class. The EventFactory class will have three methods.

The first method can be used to create an instance of an Event of a specific type. That instance can then be modified and sent through the RCI. The other two methods are used to convert an event to the protocol-dependent representation and to convert a protocol-dependent representation to an event.

ObjectFactory:
The ObjectFactory class [5] will be used to convert the protocol-independent representation of CompObjects and SimObjects to an HLA object or an EntityState PDU, or some other communication protocol, and vice versa. This class has a close relation with the FederationData class. The ObjectFactory class resembles the EventFactory and will also have three methods.

The first method can be used to create an instance of a CompObject or SimObject. That instance can then be modified and sent through the RCI. The other two methods are used to convert an object to the protocol-dependent representation and to convert a protocol-dependent representation of an object to a CompObject or SimObject.

EventManager:
The EventManager will handle all publication and subscription issues for Events that possibly occur in the federate. It will be possible to change the transport and order type of events.

The EventManager is the class that actually receives the events from the FedAmbassador instance and sends the events to the RTIAmbassador instance. It will have to transform the events to either DIS PDUs, HLA interactions, or some other representation. It will use the EventFactory class for that.

The EventManager will use the FederationData instance to determine if an incoming event is a subclass of an event the component is subscribed to. If that is the case, the EventManager will pass the event on to the CompAgent.

Similar classes are defined to manage the components and federates. These classes are called CompManager and SimManager, respectively.

CompAgent:
A component can register a callback for any type of event. The callback specifies the action to be taken when such an event occurs. The CompAgent handles incoming events by invoking the callbacks registered for that event. An invocation of a callback can either be the execution of a global function or a method.

6.3 Concept of Execution

Each application that is part of the federate needs to create an instance of the Environment. Objects of a certain Component class will be instantiated and will inform the Environment that they wish to join the federate. When doing so they will specify a frequency at which they wish to be scheduled.

Figure 5 presents the operational concept of RCI. The application communicates with the Environment. The

Environment is kept up-to-date by one or more Communication Servers. For global communication, a choice has to be made between the DIS and HLA Communication Server, wheras for local communication a dedicated Communication Server may be used. The Communication Servers do not necessarily communicate over the same communication medium.

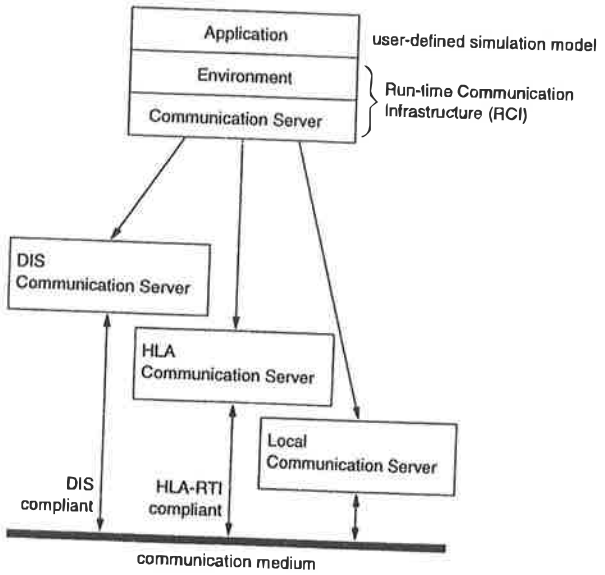

Figure 5: Operational Concept of the RCI

Once a component has successfully joined a federate, it will be able to publish Event classes and Object classes. This will inform the Environment what type of Objects and Events will be created by the component during this session. The component can also subscribe to Events and Objects. For each subscription it may register a callback. Multiple callbacks can be registered. Once this is done, the component hands over control to the Environment.

The Environment will receive incoming Events from the Communication Server and will activate the appropriate callbacks. A component can subscribe to the Sync Event which is a clock tick that will be generated at the frequency the component requested. A Component can also send its own Events. In a nutshell, this is the concept of execution of the RCI.

7. Conclusions

In this paper the SIMULTAAN architecture for simulator development has been discussed. The SIMULTAAN Simulator Architecture is intended to maximize the reuse potential of components by defining a standard interface. By making sure that components comply to the standard interface, and comply to a number of rules, they can be reused in another simulator built on the same architecture.

The global design strategy for the SIMULTAAN Simulator Architecture was presented, followed by a more detailed discussion of the RCI. Although the concepts are based on HLA, some important differences can be identified. The main differences between HLA and the SIMULTAAN approach can be summarized as follows:

- An abstraction layer (or middle-ware) will be realized to hide the complexities of the underlying interoperability standards.
- The various interoperability standards will be shielded from the developer to enable migration from DIS to HLA with minimal changes in the application code.
- Mechanisms for communication between the components within a federate will be provided.
- The exchange of simulator components between SIMULTAAN partners will be encouraged by the SOR.
- Multiple Communication Servers can be used to allow dedicated high-speed inter-component communication with minimal changes in the application code.

The first implementation of the RCI will be built on top of the HLA-RTI. The results of the SIMULTAAN project will be applied in future collaborations between the SIMULTAAN partners.

8. References

[1] Defense Modeling and Simulation Office (DMSO), "High Level Architecture Object Model Template".

[2] Defense Modeling and Simulation Office (DMSO), "High Level Architecture for Simulations Interface Specification", Version 0.3.

[3] Defense Modeling and Simulation Office (DMSO), "Federation Development and Execution Process (FEDEP) Model".

[4] J. Rumbaugh et al, "Object-Oriented Modeling and Design", Prentice-Hall, 1991.

[5] E. Gamma et al, "Design Patterns --- Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

Author Biographies

NICO KUIJPERS is a member of the scientific staff in the Command & Control and Simulation Division at TNO-FEL. He is project leader for several

collaborative projects in the area of distributed simulation, both nationally with partners in Dutch industry and academia, and internationally within European research programs. He holds a M.Sc. in Computing Science from Eindhoven University of Technology and a Master of Technological Design in Software Technology, also from Eindhoven University of Technology.

PAUL VAN GOOL is a member of the scientific staff in the same division. He has experience in the mathematical modeling of fixed- and rotary-wing aircraft for real-time simulation and the design and implementation of real-time distributed interactive simulation systems. He holds a Ph.D. in Aeronautical Engineering from the Delft University of Technology.

HANS JENSE is a senior scientist in the same division, where he coordinates the Virtual Environment R&D program, and is technical lead in various VE and simulation related projects. He holds a Ph.D. in Computer Science from the University of Leiden.

Summary Report
The 1998 Spring Simulation Interoperability Workshop

Position Papers

Volume III

March 9-13, 1998

SISO, Inc.
PO Box 781238
Orlando, Florida 32878-1238