A Component Architecture for Federate Development[1]

Marco Brassé
Wim Huiskamp
TNO Physics and Electronics Laboratory
Command & Control and Simulation
P.O. Box 96864
2509 JG The Hague, The Netherlands
Brasse@fel.tno.nl, Huiskamp@fel.tno.nl

Olaf Stroosma
Delft University of Technology
Faculty of Aerospace Engineering
P.O. Box 5058
2600 GB Delft, The Netherlands
O.Stroosma@lr.tudelft.nl

Keywords:
Simulator Architecture, Simulator Components, HLA-RTI, RD&E Applications

ABSTRACT: *The SIMULTAAN Simulator Architecture (SSA) is the product of a joint project of Dutch Simulation Industry and Research Institutes. The SSA is based on the High Level Architecture (HLA) to promote interoperability and reusability on several levels. On the level of Federations and Federates, the SSA is fully compatible with HLA. As an extension to HLA, the SSA defines a new level, that of the Federate Component. SIMULTAAN Federates are composed of Components (e.g. sensors, dynamic model, visual) in order to increase the potential for re-use. Component re-use is encouraged by the SIMULTAAN Object repository (SOR), where the SIMULTAAN partners can store and retrieve Components and/or Federates.*

SSA Federates communicate via a data-exchange middle-ware layer, called the Run-time Communication Infrastructure (RCI). The RCI is currently based on the HLA RTI, but allows other standards such as DIS. The innovative approach of SSA is that the RCI extends the Federate interoperability concepts of HLA by providing data-exchange between SSA Components in a similar way. With this approach the RCI abstracts Components from the intra-SSA Federate protocol and network hardware.

SIMULTAAN Federation execution is coordinated by a two-part system: the Federate Manager and the Scenario Manager. The Federate Manager is an SSA Component which controls the other Components within the Federate and represents the Federate to the Federation. The Scenario Manager is an SSA Federate which controls the behaviour of the Federates within the Federation by issuing commands to the Federate Managers.

A SIMULTAAN Federation is defined by its Federation Object Model (FOM), which is equal to the HLA FOM. The HLA Simulator Object Model (SOM) describes each SSA Federate in the Federation. Components are described by their SSA Component Object Models (COM's). A particular SSA Federate is defined by an aggregate of COM's, the SSA Simulator Component Object Model (SCOM).

.

1. Introduction

A simulator is usually divided into several components, which may be manufactured by different parties. Simulator component technology can be used for the rapid and cost-effective development of simulators through the re-use and exchange of existing simulator components.

Components are considered the basic building blocks of a simulator, which can potentially be used for more than one type of simulator. Examples of simulator

[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).

components are a dynamics component, a component that visualises the virtual environment, or a component that handles the I/O inside a simulator mock-up. Consequently, a simulator can be thought of as a set of interacting components. The total functionality of the simulator may be expressed as the 'sum' of the functionality of its constructing components.

In order to apply component technology successfully, the component/simulator architecture has to fulfil certain conditions: the functionality of each component must be well defined and the interfaces between components have to be defined in a formal manner. Formally described interfaces reduce incompatibility problems that might otherwise not be noticed until the integration phase. A clear distinction between the interface(-description) of a component and its functionality improves the re-use of that component since the interface agreements of a new simulator usually imply modifications of the interfaces of available components. The architecture must also support a mechanism to co-ordinate the overall behavior of the components, which should for example ensure proper initialization of components before they are allowed to exchange simulation data.

This paper focuses on the simulator/component architecture that was developed in the framework of the SIMULTAAN project, called the SIMULTAAN Simulator Architecture or SSA.

The paper is organized as follows. Section 2 describes the SSA which is the main product of the Dutch SIMULTAAN project. Section 3 describes the development of Federates based on the results of SIMULTAAN. The RCI middle-ware layer is discussed in Section 4. The SIMULTAAN demonstration set-up, which was used as a functional proof of the SSA concept is presented in Section 5. Finally, conclusions are drawn in Section 6.

2. SIMULTAAN Simulator Architecture

SIMULTAAN was a 2.5 years project which brought together knowledge and experience in the area of simulators and distributed simulation from universities, research institutes and industry in The Netherlands. The six members of the consortium 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.

Two main results of the project can be distinguished:

1. *SIMULTAAN Simulator Architecture*.
   A generic framework applicable for a wide range of simulators, including manned mock-ups of vehicles, high-fidelity flight simulators and unmanned simulators.
2. *Permanent Intellectual Infrastructure*.
   The SIMULTAAN consortium strengthened working relationships between its partners.

The SIMULTAAN Simulator Architecture (SSA) defines a simulator component architecture that addresses the identified needs for a successful federate development process and makes effective use of simulator component technology. The SSA is intended to maximize the re-use 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 re-used in other simulators built on the same architecture. The SSA is often used in an RD&E environment that requires rapid re-configurability of simulators, but it can also be used in an industrial environment.

The SSA facilitates interoperability between Federates in a Federation. On the level of Federates and Federations, the SSA is fully compatible with HLA. As an extension to HLA, the SSA introduces a new level, that of the Federate Component. A SIMULTAAN Federate is composed of SIMULTAAN Components. The SSA facilitates interoperability between Components inside a Federate, in a similar manner as HLA-RTI does between Federates.

The SSA identifies the following key architectural elements: *Component, Run-time Communication Infrastructure (RCI), Federate Manager (FM),* and *Scenario Manager (SM).*

A *Component* is the basic building block for a SIMULTAAN federate. All SIMULTAAN Components interact with the simulation environment through a standard interface, which is provided by the Run-time Communication Infrastructure.

The *Run-time Communication Infrastructure* (RCI) is an object-oriented middle-ware layer for exchanging data between Components as well as between Federates. The RCI provides the Component developer an abstraction layer to shield the developer as much as

possible from the underlying interoperability standards.

Each SIMULTAAN Federate is built from a set of Components with one obligatory Component, called the *Federate Manager*. The Federate Manager acts as an intermediary between the Components in the Federate and the rest of the Federation; it represents the Federate to the Federation. The Federate Manager keeps track of the state of its Federate and its Components.

The SIMULTAAN *Scenario Manager* is an SSA Federate that controls the behavior of the Federates within the Federation by issuing commands to the Federate Managers (like start scenario execution, stop scenario execution, hold scenario execution). The Scenario Manager is the only SIMULTAAN Federate that does not have a Federate Manager.

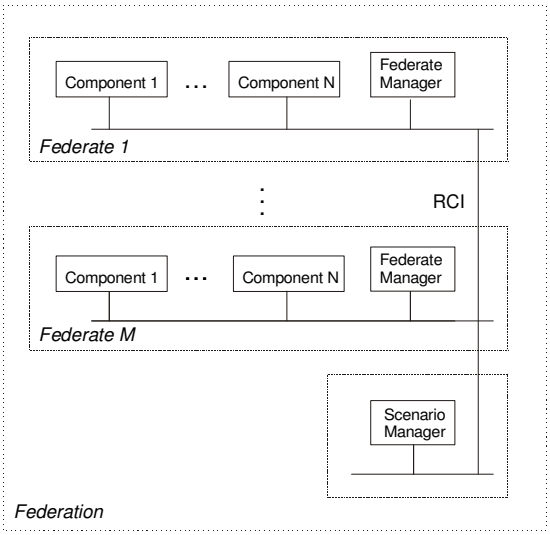

Figure 1

Besides the identified elements in the architecture, the SSA consists of 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 most prominent SSA Rule is that all Components must adhere to the SSA State Transition Diagram (SSA-STD), which is depicted in Figure 2. The SSA-STD is used by the Federate Manager to co-ordinate the state transitions of the Federate during the scenario execution. The Federate Manager prepares its Federate for joining the Federation and when the Federate has joined the Federation, the Federate Manager initiates and checks the state transition of the Components in its Federate, as requested by the Scenario Manager.

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 is equivalent to the HLA-OMT [2]. The different object models used in the SSA are presented in Section 3.

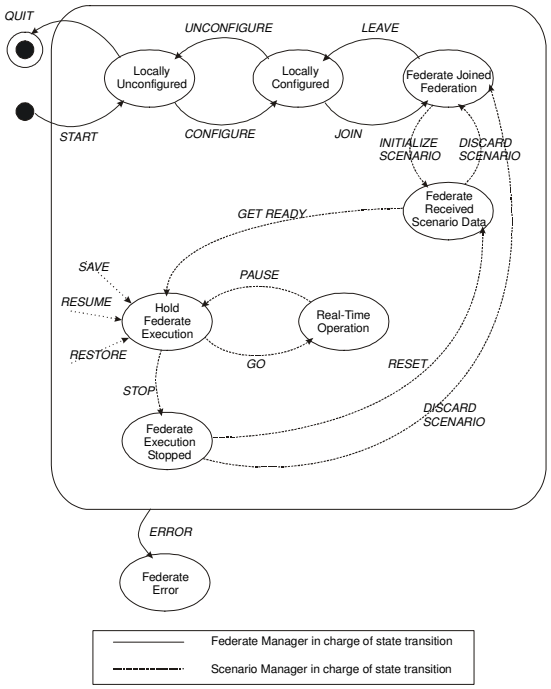

Figure 2

3. SIMULTAAN Federate Development

SIMULTAAN Federate Development describes the way SIMULTAAN partners produce federates. 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 SIMULTAAN Federation Object Model (FOM), which is equivalent to the HLA-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 SIMULTAAN Federate.

On the Federation level, the SSA identifies the SSA-SOM and the SSA-FOM, both equivalent to the HLA-SOM and HLA-FOM. On the Component level, the SSA identifies the SSA-COM and the SSA-SCOM, which are explained next.

Each SIMULTAAN component has a Component Object Model (SSA-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.

Each SSA Federate is build from a set of interoperable Components. The interactions and attributes that are exchanged between all Components of one Federate, including the data that is exchanged with other SSA Federates, is formally described in the SIMULTAAN Simulator Component Object Model (SSA-SCOM). The difference between the SCOM and the SOM is that the latter merely describes the interface between SSA Federates and not the intra-Federate communication between the Components.

The SSA-COM and the SSA-SCOM object models have similar roles on the component level compared with the SOM and the FOM on the federation level. Both the SSA-COM and SSA-SCOM descriptions are expressed in the HLA-OMT format.

Distinction between COM and SCOM on one hand, and SOM and FOM on the other hand, 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. Distinguishing between local and global communication allows for a mapping of the local communication onto a dedicated network that is able to handle intra-Federate high-speed data exchange while having the possibility to communicate with the outside world at another data exchange rate and possibly via other physical network media.

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) [4].

A SIMULTAAN Federate will be designed with optimal use of existing components. Therefore access is needed to the descriptions of object models and components that are available in the SIMULTAAN Object Repository (SOR). The SOR will contain SSA compliant simulators, components and tools. It may also contain configuration, initialization, and validation data. The SOR allows controlled access by the SIMULTAAN partners.

4. Run-time Communication Infrastructure and Code Generation

The SIMULTAAN Simulator Architecture (SSA) is the common high-level architecture for simulators and tools. The SSA provides services to both the Components and the Federate, which is the distributed composition of the Components. All Components interact with the simulation environment through a standard interface, which is provided by the Run-time Communication Infrastructure. The RCI is the implementation of the SSA Interface Specification.

The RCI provides the component developer with the necessary functionality to incorporate the Component into a SIMULTAAN Federate. The RCI provides a protocol-independent interface to the simulated environment. The design of the abstraction layer and the Application Programmer's Interface (API) have been discussed in detail in a previous paper [1], and is briefly summarized below.

The RCI consists of two separate software layers, one is called the *Environment* and the other is called the *Communication Server* (see Figure 3).

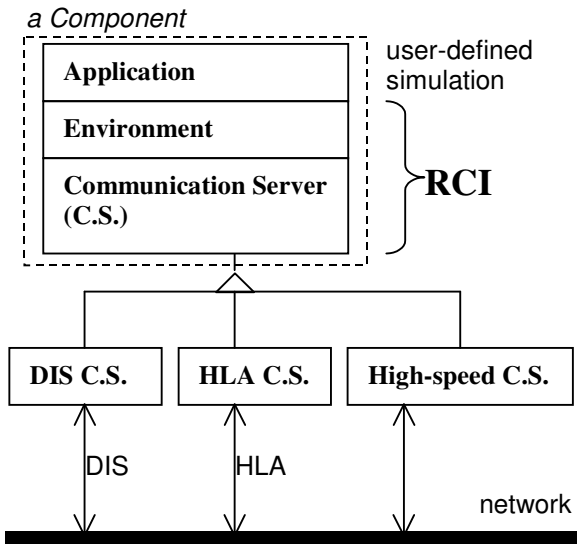

Figure 3

The Environment provides components with an overview of both the federate and the federation. The Environment reflects the current state of the federate, i.e., the state of all its components. It allows the addition and deletion of components. Furthermore, the RCI allows 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.

The Communication Server represents the layer 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 is based on the HLA Interface Specification [3]. A Communication Server communicates with other Communication Servers to exchange object and event information to keep the Environment up-to-date. Currently, the Communication Server is based on the HLA RTI, but allows other standards such as DIS. Dedicated versions of the Communication Server can be implemented for the support of specific simulation protocols or network layers. This requires only minimal changes in the application-specific source code as it merely interacts with the Environment.

The innovative approach of SSA is that the RCI extends the Federate interoperability concepts of HLA by providing data-exchange between SSA Components in a similar way. Components use equivalents of the HLA federation management services, declaration management services and object management services in a similar way. In this way, the RCI abstracts Components from the intra-SSA Federate protocol and network hardware, and establishes a clear separation between communication aspects and application-specific aspects. This enables a Component developer to focus on the required functionality of the specific Component rather than the technical details of the communication aspects.

To further facilitate the developer with an abstraction of the communication it is noted that the simulation objects and the simulation events are *formally* described in an HLA-OMT format through its Component Object Model (SSA-COM). This enables the use of automatic code generators to construct object-oriented classes (for instance C++ or Java) for each simulation object and simulation event in the COM. The automatic code generation approach has proven to be highly successful in SIMULTAAN. The generated code shields the application programmer from doing elaborate bookkeeping concerning attribute updates, while making use of the encoding and decoding facilities offered by the RCI to communicate attribute and parameter values along the physical network.

5. Implementation and Demonstration

In SIMULTAAN, the SSA Federation is mapped onto an HLA Federation. This approach is straightforward as the SSA-SOM and SSA-FOM are identical to the HLA-SOM and HLA-FOM.

An SSA Federate consists of a set of Components, which always includes at least the Federate Manager Component (the exception on this rule is the Scenario Manager). The set of Components –and hence the SSA Federate- is implemented as an HLA federation on itself, each Component being mapped onto an HLA federate. This ensures that communication on the Component level is similar to communication on the Federate level. Furthermore, the SSA-COM and SSA-SCOM play a role analogous to the HLA-SOM and HLA-FOM within the constructed HLA federation.

In the SSA implementation we now have one HLA federation representing the SSA Federation and additional HLA federations for each SSA Federate. It is noted that each SSA Federate is represented in the SSA Federation by its Federate Manager Component. The SSA Federation now consists of all Federate Manager Components (implemented as HLA federates) of the

participating SSA Federates plus the Scenario Manager.

This concept is feasible only if the Federate Manager Component can be part of two HLA federations, i.e., the HLA federation that constitutes the SSA Federate and the HLA federation that constitutes the SSA Federation. To this end, the Federate Manager requires a special version of the RCI that is equipped with two Communication Servers, one for local communication inside the SSA Federate, and one for global communication between the SSA Federates. This implies that the RCI, as part of a single application, must fully participate in two HLA federations simultaneously, which is not feasible with the RTI version used.

For this purpose, a special version of the Communication Server of the RCI was developed that is able to communicate with another Communication Server along a separate socket connection. The FM is subsequently split up in two processes, one process participates in the HLA federation that forms the SSA Federate and the other process participates in the HLA federation that forms the SSA Federation. In this way, a bridge is implemented between two HLA federations, exchanging data through a separate connection.

This approach requires that each simulation event and simulation object update must be encoded/decoded along the socket connection. A shared memory solution was considered, but the socket solution has the advantage that the processes may reside on different (low-end) computers and it proved to be a very portable solution across different operating systems.

A functional proof of concept of the SSA architecture has been presented in a large demo on the 24th of June, 1999 at TNO-FEL in which all SIMULTAAN partners participated. The demo used the DMSO RTI 1.3v5 as the underlying distributed simulation layer for the RCI Communication Server.

A rescue/evacuation scenario was conceived that comprised two manned fire-truck Federates at TNO-FEL (the Hague) and a manned rescue helicopter Federate located at the Delft University of Technology.

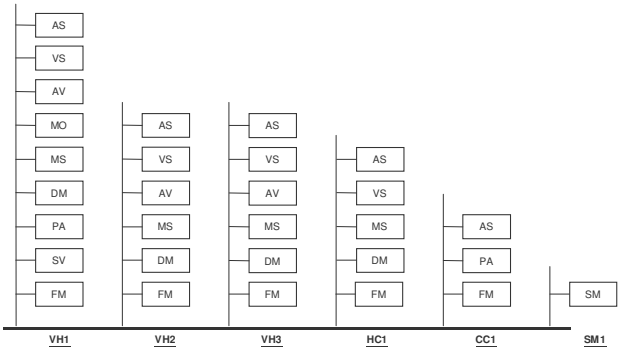

Figure 4

In Figure 4, a schematic view is presented of the Federates that participated in the SIMULTAAN demonstration. In this figure, VH1, VH2 and VH3 represent fire truck Federates, HC1 is the helicopter Federate, CC1 is a control center Federate to assess the performance of the players, and SM1 depicts the Scenario Manager. The components of each SSA Federate are shown in the boxes.

Some examples of the components used are the Federate Manager (FM), a Dynamics Model (DM), a Visual System (VS), a Mock-up Server (MS), a Motion Platform (MO), an Audio System (AS), and a performance assessment Component (PA). Each component was executed on a separate computer. The computer hardware consisted of a mixture of Unix based machines (SGI) and NT machines. Both low fidelity and medium fidelity mock-ups were provided.

6. Conclusions

In this paper the SIMULTAAN architecture for simulator development has been discussed. The SIMULTAAN Simulator Architecture is intended to maximize the re-use potential of components by defining a standard interface. Components that comply to the standard interface, and comply to a number of rules, can be re-used in another simulator built on the same architecture.

The global design strategy for the SIMULTAAN Simulator Architecture was presented, followed by the Run-time Communication Infrastructure. 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) and a code generator is applied to hide the complexities of the

underlying interoperability standards.

- The various interoperability standards are shielded from the developer to enable simulation protocol migration with minimal changes in the application code.
- Mechanisms for communication between components within a federate are provided.
- 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 has been built on top of the HLA-RTI. In the near future, the RCI will have additional support for the real-time scheduling of Component tasks. Further activities concern the development of a dedicated version of the RCI that is based on a high-speed Communication Server.

The results of the SIMULTAAN project will be applied in future collaborations between the SIMULTAAN partners.

7. References

[1] Nico Kuijpers, Paul van Gool, Hans Jense, "A Component Architecture for Simulator Development", Simulation Interoperability Workshop, Spring 1998

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

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

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

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

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

Author Biographies

MARCO BRASSE is a member of the scientific staff in the Command & Control and Simulation Division at TNO-FEL. He is a project member for several projects in the area of distributed simulation. He holds an M.Sc. in Computing Science and a Master of Technological Design (MTD) in Software Technology, both from Eindhoven University of Technology, the Netherlands. His research interests include parallel and distributed computing, software architectures, and design methodologies.

WIM HUISKAMP is a research scientist in the same division. He holds an M.Sc. degree in Electrical Engineering at Twente University of Technology, The Netherlands. He works in the field of High Performance Computing and Networking and specialises in design and implementation of distributed computer architectures. His research interests include system architectures, real-time visual simulation and multi-media technology. Wim was the project lead for SIMULTAAN.

OLAF STROOSMA is a software architecture researcher at the Control and Simulation Division of Delft Aerospace. He holds an M.Sc. degree in aerospace engineering from Delft University of Technology (DUT). He has participated in the SIMULTAAN project of the joint Dutch simulation industries and institutes, as well as in the SIMONA project aiming the development of an advanced flight simulator at DUT. His research interests include software architectures for real-time distributed simulation, and the application of intelligent agent technology to improve human-computer interaction.