A Component Architecture for Simulator Development

Marco Brassé

The SIMULTAAN Simulator Architecture (SSA) is a distributed simulator component architecture, developed in a joint project of Dutch Simulation Industry and Research Institutes. SSA-compliant simulators are composed of a set of distributed Components (e.g., sensors, dynamics model, visual system) in order to increase the potential for reuse. The interfaces and capabilities of simulator components are described by formal object model templates, based on the HLA (High Level Architecture) interoperability standard. Simulator components that comply to the formal interface descriptions, and a number of rules, can be reused in another simulator built on the same architecture. A component repository further facilitates the exchange of well described simulator components that are developed by different partners. The SSA also promotes the use of code-generator tools and middleware for component based simulator development. The architecture has been successfully demonstrated last year and will be further developed by TNO-FEL.

Introduction

Distributed simulations have become an indispensable tool in many research and development areas, both in the military and the civil domain. A relative new application area is the acquisition of products through the evaluation of computer generated prototypes in a virtual reality environment. This application area is often termed Simulation Based Acquisition (SBA). Multiple simulators are networked to provide for a specific simulation environment in which the requirements and the design can be evaluated before an actual prototype or product is manufactured. In this way, functional and operational constraints can be identified in an early stage of the development process. A well-known example in the military domain is the development of a reconnaissance vehicle that must be equipped with a to-be-determined set of sensors suited to perform certain tasks in a complex warfare environment.

Creating a research environment for Simulation Based Acquisition and design is merely cost-effective if the simulators are highly reconfigurable and can be used throughout many projects. This suggests that a simulator should be composed of simulator *components* that can be integrated in a uniform manner. In general, a simulator is composed of functional models in which each model can be mapped onto a separate component, for example a flight simulator may be based on a visual component, a dynamics component, a radar component, and a human-simulator interaction component. It should be possible to simply replace a simulator component by another functional equivalent one while maintaining the same simulator behavior.

Simulator component technology inherently allows the participation of different specialized companies in the development process of a simulator. Each participating company focuses on the realization of a specific component that fits best with the techno-

logical background of that company. For example, a partner that is specialized in mechanical engineering could develop a motion-based platform wheras a partner that is specialized in aeronautical engineering could develop a flight dynamics model. This approach has also been recognized by the dutch simulation industry, represented in SIMNED. This has led to the SIMULTAAN project in which the concept of a component based simulator architecture has been developed, called the SIMULTAAN Simulator Architecture [1, 2].

The SIMULTAAN Simulator Architecture (SSA) is the main subject of this article. After providing some background on the SIMULTAAN project, the SSA component based simulator architecture is unfolded.

SIMULTAAN

SIMULTAAN was a 2.5 years project which started early 1997 and ended with a successful demonstration of the component based simulator architecture concept in the summer of 1999. The SIMULTAAN project brought together knowledge and experience in the area of simulators and distributed simulation from universities, research institutes and the simulation industry in The Netherlands. It was partly funded by the Dutch initiative for High Performance Computing and Networking (HPCN). The six members of the consortium are:

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

Two main results of the project can be distinguished:

1. *SIMULTAAN Simulator Architecture (SSA)*. 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 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 the development of 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 that are built on the same architecture. The SSA is used in a research and development environment that requires rapid re-configurability of simulators, but it can also be used in an industrial environment.

The successful results of SIMULTAAN have led to a further development of the SSA framework by TNO Physics and Electronics Laboratory, the project leader and primary developer of the SSA. The SSA is also in the process of being integrated in TNO-FEL's Electronic Battlespace Facility (EBF), which is a research facility for distributed simulations and virtual reality technology in the military domain [3]. More information on the EBF can be found on the EBF web-site [4].

SIMULTAAN Simulator Architecture

Simulator component technology is based on distributed simulation technology. A distributed simulation traditionally consists of multiple networked applications that are executed on possibly various computer platforms [5]. This approach requires an interoperability standard that allows applications to communicate with one another no matter where they are located or who has designed them. Several interoperability standards have been developed independently by several organizations in industry and government.

A well-known interoperability standard is the CORBA standard, developed by several industrial partners organized in the Object Management Group [6]. Another interoperability standard is initiated by the United States Department of Defense

in 1995, which is named the High Level Architecture (HLA) [7].

The HLA standard is aimed at distributed simulations and promotes the re-use of simulation models. HLA attempts to specify the general structure of the interfaces between simulation applications without making specific demands on the implementation of each simulation. HLA is developed in a co-operative, consensus-based forum of developers, organized in the Simulation Interoperability Standards Organization (SISO) [8], and is currently in the process of becoming an IEEE standard (IEEE 1516). For a change, the steering committees of CORBA and HLA actually do work together, and the two standards co-exist in the world of distributed programming. In fact, the reference implementation of the HLA software actually relies on a real-time CORBA implementation for providing application interoperability.

Since the SIMULTAAN Simulator Architecture is based on the HLA standard, some HLA terminology is inevitably used in the next sections. Therefore, we give a quick HLA mini-survival guide here. See the Xootic Magazine on Simulation for a more complete introduction [9]. A *federation* consists of simulation applications, called *federates*. Federates may be simulation models, data collectors, simulators, or other tools that interact with other federates. A simulation session, in which a number of federates participate, is called a *federation execution.* Simulated entities are called *HLA objects.* Simulation events are called *HLA interactions.* All possible exchange of data-types between the federates of a federation is defined by the *Federation Object Model* (FOM). The capabilities of a federate, in terms of the objects and interactions it can exchange with other federates, is defined by the *Simulation Object Model* (SOM). The FOM and SOMs may be regarded as contracts that function as interface specifications for the federate developers.

The HLA is formally defined by three parts [7].

- The *Interface Specification* is a formal, functional description of the interface between the HLA application and the underlying Run-Time Infrastructure, see below.
- A set of *HLA Rules* is defined to which HLA applications have to comply.
- The *Object Model Templates* define the structure of the FOM and the SOM descriptions.

The *Run-Time Infrastructure* (HLA-RTI) is the implementation of the HLA Interface Specification and forms the basic software communication layer for all HLA federates. The HLA software can be compared to a distributed operating system for all communications between the federates in a federation. Although the HLA standard is an open standard, RTI implementations are not considered Open Source software. More information about HLA can be found on the HLA web-site [7].

Now the SIMULTAAN Simulator Architecture (SSA) also 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 actually composed of SIMULTAAN Components. This means that a single simulator is actually a distributed simulation, composed of multiple applications, called *Components*, each responsible for a specific functional model of the simulator. The SSA facilitates interoperability between Components inside a federate, in a similar manner as the HLA-RTI does between federates.

As the SSA is an extension of the HLA, its definition is also based on three parts, namely the SSA Interface Specification, the SSA Rules, and the SSA Object Model Templates, which are all extended versions of their HLA equivalents [7]. The SSA identifies the following key architectural elements: *Component, Run-time Communication Infrastructure (RCI)*, *Federate Manager (FM)*, and *Scenario Manager (SM)*, (Figure 1).

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

The *Run-time Communication Infrastructure* (RCI) is an object-oriented middleware layer for exchanging data between Components as well as between federates. The RCI provides the Component developer a high-level abstraction layer to shield the developer as much as possible from the underly-

ing communication framework, which is, probably not surprising, the HLA-RTI. For the communication between the distributed Components, preferably a dedicated high-performance HLA-RTI could be used. See section “Run-time Communication Infrastructure and Code Generation” below for a more elaborate discussion of the RCI middleware layer.

Each federate is composed of a set of distributed 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 and it presents information from the federation to its Components. The Federate Manager also keeps track of the state of its federate and its constituent Components.

The *Scenario Manager* is a special federate that controls the behavior of the federates within the federation by issuing commands to the Federate Managers (such as ‘start scenario execution’, ‘stop scenario execution’, ‘hold scenario execution’).

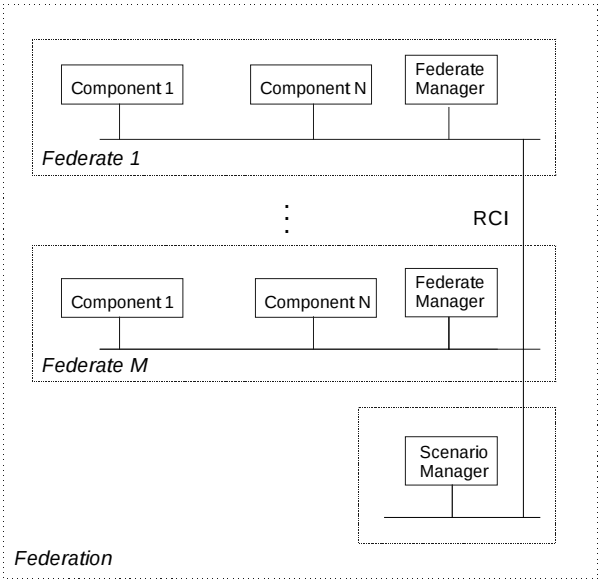

Figure 1:

Although this article should not be too technical, one SSA Rule is especially worth mentioning, namely that all Components must adhere to the 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 Components, for *joining*, i.e., connecting to, the federation. When the federate has joined the federation, state transition may be requested by the Scenario Manager. The Federate Manager receives these state transitions, checks them with the federate’s state, and forwards them to the Components. It collects the responses from the Components, and sends a reply to the Scenario Manager in return.

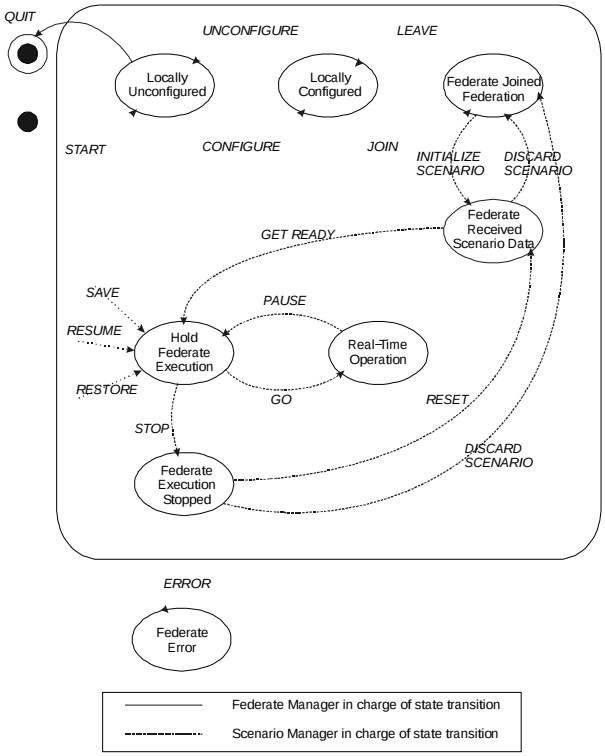

Figure 2:

The SSA Interface Specification (SSA-IF) is a formal, functional description of the interface between the application and the Run-time Communication Infrastructure (RCI). It describes the interface calls that a Component uses to exchange data to and from other Components and federates.

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 equal to the HLA-OMT [7]. The different object models used in the SSA are presented in Section 3.

Run-time Communication Infrastructure and Code Generation

The Simulator Architecture (SSA) is the SIMULTAAN architecture for networked simulators and tools. The SSA provides services to both the Components and federates. All Components interact with the simulation environment through a standard interface, which is provided by the Run-time Communication Infrastructure (RCI).

The RCI provides the component developer with the necessary functionality to develop a SSA compliant simulator component that can be integrated in a simulator. The RCI provides a protocol-independent interface to the simulated environment. The RCI implements all functionality described by the SSA Interface Specification. Components do not interact directly with each other, but only via calls to the RCI software library that is linked with the component's application.

The design of the abstraction layer and the Application Programmer's Interface (API) are discussed in full detail in another paper [1], and is briefly summarized below.

The RCI middleware layer performs the following tasks:

- exchange of data between Components;
- handle object and interaction bookkeeping;
- process user-defined event callback functionality;
- present object attribute updates and interactions to the Component;
- maintain synchronization between Components.

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

The Environment layer 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 run-time creation and deletion of components. It also maintains data exchange *interests* of Components.

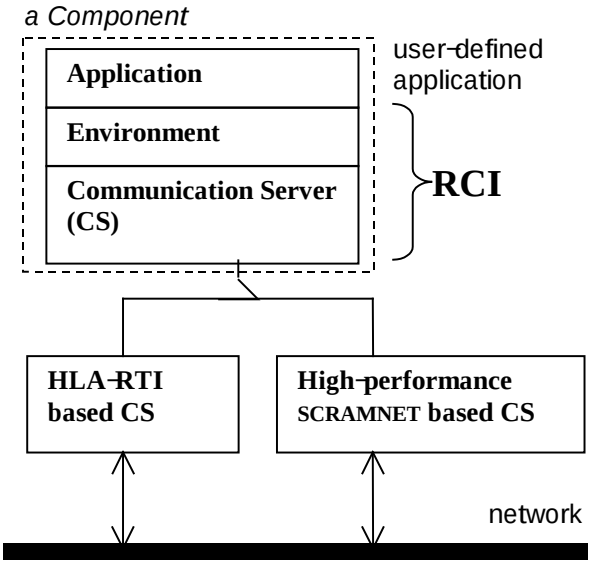

Figure 3:

For example, the Visual Component of a simulator needs the position of the simulated vehicle in the simulated world, which is calculated by the Dynamics Component. Now the Visual Component must *subscribe* to the object that represents the state of the simulator; this is done through an invocation of the method *Environment::subscribeToObject()* without knowledge of which other Component actually updates the simulator's state. In case the Dynamics Component calculates the new position, this component must *publish* the simulator's state object of which the position is an attribute through invocation of the method *Environment::publishObject()*. Conversely, the Dynamics Component application does not need to know to which other federates the new simulator state must be sent. This is actually done by the *Communication Server* sub-layer that matches the publish and subscribe interest using a distributed computation. A similar approach is used to distribute events between Components.

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 [7]. A Communication Server communicates with other Communication Servers to exchange basic object and event information to provide the Environment with the most recent data

updates. Currently, the Communication Server is based on the HLA RTI (see Figure 3). Dedicated versions of the Communication Server can be used for the support of specific simulation standards or network layer protocols. The middleware layer approach now requires only minimal changes in the application-specific source code as the application source code merely interacts with the Environment layer.

The innovative approach of SSA is that the RCI middleware layer extends the federate interoperability concepts of HLA by providing data-exchange between Components in a similar way. Components use equivalents of specific HLA capabilities, such as federation management services, declaration management services and object management services in a similar way [7]. In this way, the RCI abstracts components from the interoperability protocols and network hardware, and establishes a clear separation between communication aspects and application-specific or domain-dependent aspects. This enables a Component developer to focus on the required application functionality 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 through its Component Object Model (SSA-COM), which is an extension to the HLA Object Model Template, and is similar to the SOM (see Section 3). This enables the use of automatic code generators to construct object-oriented classes (for instance C++ or Java) for each user-defined simulation object and simulation event in the SSA-COM. The automatic code generation approach has proven to be highly successful, not only in SIMULTAAN but also in other projects (for example the 'Laguna Beach: HLA on baywatch' project, in which the automatically generated code is linked with 'legacy' simulation code [10]).

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.

For example, the SSA-COM may describe a subclass called 'AggregateEntity' with 'EntityID' as one of its integer attributes, stated in the following SSA-COM description:

```
Class (ID 10) (
      (Name "AggregateEntity")
      (Attribute (Name "EntityID")
            (DataType "integer")
            (Cardinality "1")
      )
      (Attribute ...
      )
      (SuperClass 2)
)
```

The RCI Code Generator will produce the following piece of object-oriented source code (in terms of a piece of a C++ header file). The component developer is now able to provide EntityID with a value, whose new value is subsequently distributed to other interested Components by the RCI middleware layer, based on the publish and subscribe interests of the Components.

```
class AggregateEntity
   : public BaseEntity{
public:
   AggregateEntity();
   virtual  ~AggregateEntity();

   int getEntityID();
   void setEntityID(int);
private:
   int m_
};
```

The use of code generators have already proven to be useful in many other computing science disciplines where formal languages are introduced. As more and more (parts of) specifications and designs are written in a formal (meta-)language, code generators will be used more frequently to minimize programming and debugging time.

SIMULTAAN Simulator Development

SIMULTAAN Federate Development describes the way partners should manufacture 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 manufacturing 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.

Each Component has a Component Object Model (SSA-COM). This object model formally specifies the object attributes and interactions a component publishes to other components. It also specifies the object attributes and interactions a component will subscribe to during run-time. Each federate is built up from a set of interoperable distributed Components. The interactions and object attributes that are exchanged between all Components of one federate, and including the data that is exchanged with other 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 standardized SSA Object Model Templates. The SIMULTAAN object models enable clear specifications for the capabilities of federates and components. Federate and federation development in SIMULTAAN are comparable to the HLA Federation Development and Execution Process (FEDEP) [11].

A 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, models and tools. It may also contain configuration, initialization, and validation data. In traditional software engineering, a software product is validated against its well-defined functional requirements which should reflect the sponsor's needs. In the simulation field, however, the issue of validation is much more complex, especially if some real-world phenomena are to be simulated. It is often unclear what aspects of the real world have to be modeled to obtain a simulation that is useful *for its intended purpose.* Verification and validation procedures are commonly used to identify and solve problems in a particular product, but they can also be used to merely assess, and document, the quality of a product. This documentation is actually the key to the success of the 're-use' of simulation components: it is often the primary source of information on which the decision is based whether the component can be re-used as part of some other product. A well-defined verification and validation policy, in conjunction with an up-to-date Repository, is the most prominent precondition for the success of simulation component re-usability [12,13].

SIMULTAAN Simulator Demonstration

A functional proof of concept of the SSA component based simulator architecture has been presented at a large demonstration on the 24th of June, 1999 at TNO-FEL in which all SIMULTAAN partners participated. The demo used the reference RTI (version 1.3.5) as the underlying communication layer for the RCI Communication Server.

A rescue and evacuation scenario was conceived that comprised two human controlled fire-truck simulators (federates) located at TNO-FEL, in the Hague, and a rescue helicopter simulator located and controlled at Delft University of Technology. Together with instructors support, a training simulation exercise was presented to the audience. The audience was presented with real-time visual images of the virtual terrain from within the helicopter and the main fire truck.

??? presents a schematic view of the federates that

were developed for the demonstration. In this figure, VH1, VH2 and VH3 represent manned fire truck simulators, HC1 is the helicopter federate at TUD, CC1 is a control center with training instruction and monitoring tools, and SM1 depicts the Scenario Manager. The boxes (vertical) represent the various Components that were present in each simulator (federate).

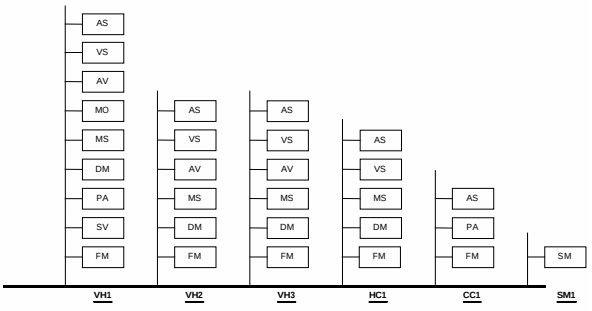

Figure 4:

Examples of the several types of components 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 high-performance graphics workstations (SGI) and Windows NT machines. Both low fidelity and medium fidelity mock-ups for human-simulator interaction were provided.

The most important result of the demonstration was the reduction in test and integration time. The integration phase took less than two weeks, which is a remarkable achievement for the development of so many new components with many different types of hardware and software, and the involvement of six partners in a research related project. To a large extent, this may be contributed to the use of formal interface descriptions for simulator components, and the use of a middleware layer approach for abstracting the applications from the complexities of the interoperability standard. Component developers really focused on the component's functionality, as well as all non-functional requirements, instead of dealing with all intricate communication and network issues.

Concluding remarks

In this paper the SIMULTAAN architecture for simulator development (SSA) has been discussed. The SIMULTAAN Simulator Architecture is intended to maximize the re-use potential of simulator 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 of the SIMULTAAN Simulator Architecture was presented, followed by the Runtime Communication Infrastructure and the middleware layer approach. 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:

- The SSA identifies networked Components and necessary communication mechanisms between components inside a distributed federate.
- The SSA provides the Component developer with an abstraction layer (or middle-ware) and a code generator to hide the complexities of the underlying interoperability standard.
- The SSA shields the interoperability standard from the developer to enable migration to a future interoperability standard while keeping changes in the application code to a minimum.

The first implementation of the RCI has been built on top of the HLA-RTI. Currently, the lessons learnt are being implemented and the SSA will be used by the SIMULTAAN partners this year in a follow-up project.

References

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

[2] Marco Brassé, Wim Huiskamp, Olaf Stroosma, "A Component Architecture for Federate Development", Proc. 1999 Fall Simulation Interoperability Workshop, Orlando, Florida, September 1999.

[3] N.H.L. Kuijpers, R.J.D. Elias, R.G.W. Gouweleeuw, *Electronic Battlefield Facility* in Battlefield Systems International 96 'Integrated Battlefield Management', Volume 2, 4–6 June 1996, Chertsey, UK.

[4] TNO-FEL's Electronic Battlespace Facility, http://www.tno.nl/instit/fel/ebf/en/.

[5] DIS Steering Committee, "The DIS Vision, Map to the Future of Distributed Simulation", Version 1, Institute for Simulation & Training, May 1994, Orlando, Florida, IST-SP-94-01.

[6] CORBA standard, http://www.omg.org/corba.

[7] HLA Technical Reports, http://hla.dmso.mil

[8] Simulation Interoperability Standards Organization, http://www.sisostds.org.

[9] Marco Brassé and Nico Kuijpers, "Standardising Distributed Simulations: The High Level Architecture", Xootic Magazine, Volume 7 Number 1, July 1999.

[10] Leo Breebaart, Marco Brassé, Wim Huiskamp, Hans Jense, "Laguna Beach: HLA on Baywatch?", Proc. 1999 Fall Simulation Interoperability Workshop, Orlando, Florida, September 1999.

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

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

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

Author Biography

Marco Brassé is a member of the scientific staff in the Command & Control and Simulation Division at TNO Physics and Electronics Laboratory (TNO-FEL). He is a software architect for several projects in the area of distributed simulation, both nationally and internationally with partners in the armed forces and simulation industry. His present activities are focused on HLA-compliant component based simulator development strategies for real-time simulators. He holds a M.Sc. in Computing Science and a Master of Technological Design in Software Technology, both from Eindhoven University of Technology. He can be reached by E-mail at 'brasse@fel.tno.nl'.