Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

INTERNET-BASED SIMULATION USING OFF-THE-SHELF SIMULATION TOOLS AND HLA

Steffen Straßburger, Thomas Schulze, Ulrich Klein

Department of Computer Science
Otto-von-Guericke University Magdeburg
Universitätsplatz 2
D-39106 Magdeburg, GERMANY

James O. Henriksen

Wolverine Software Corporation
7617 Little River Turnpike, Suite 900
Annandale, VA 22003-2603, U.S.A.

ABSTRACT

The United States Department of Defense's High Level Architecture for Modeling and Simulation (HLA) provides a standardized interface for distributed simulations. The recent advent of HLA has greatly increased interest in the use of distributed, interoperable simulation model components. To date, most models using HLA have been developed in conventional high-level languages (primarily C++). This paper presents approaches by which HLA can be used to interconnect distributed model components which are developed using commercially available, off-the-shelf *simulation* software. The requirements imposed on such simulation software by HLA are discussed, and four approaches for adapting such software for use with HLA are presented. A generalized, model-independent approach which was developed for SLX is presented.

1 INTRODUCTION / MOTIVATION

In the past several years, the Internet and its multimedia front-end, the WWW, have undergone rapid expansion and achieved world-wide acceptance. Exploiting the Internet and WWW for use in modeling and simulation holds great promise, but provides significant technical challenges (Dorwarth et al. 1997).

The development of distributed simulations requires two types of functionality, (1) a simulation language or package in which model components are constructed, and (2) tools which implement a protocol by which model components can be interconnected. In an ideal world, both types of functionality would be integrated into a single simulation package which one could purchase "off the shelf." In the absence of an integrated package, one can consider two approaches. One could start with an existing language, such as Java, which fulfills many of the requirements for communication over a network, and add simulation capabilities. Alternatively, one could start with a simulation language and add the tools necessary to accomplish interoperability over a network. Silk (Healy 1997) is an example of the former approach. The work described in this paper is an example of the latter approach.

We chose SLX (Henriksen 1996, 1997) as an off-the-shelf simulation tool, because it provides both excellent simulation capabilities and mechanisms for communicating with packages written in languages other than SLX.

There are a number of protocols which have been developed for constructing distributed, interoperable simulations (e.g. *Aggregate Level Simulation Protocol ALSP*, *Distributed Interactive Simulation DIS*). We chose the High Level Architecture (HLA) as the protocol for adding distributed interoperability to SLX. Still under construction, DoD's HLA protocol surpasses its predecessors while maintaining a degree of upward compatibility.

Most of the sample applications using HLA released by DoD to date have been written in conventional high-level programming languages such as C++, ADA95, Java or even FORTRAN.

Constructing simulations in conventional languages has a number of disadvantages that have been well-known since the 1960s. Many factors influence the choice of a language. Often, language choice is almost a matter of religion. Notwithstanding the burgeoning use of C++ as a tool for developing simulations, we feel that it is particularly ill-suited for such use, for the following reasons:

1. It is difficult to learn.
2. It is too easy to make mistakes which have disastrous consequences, but are difficult to find, e.g., faulty use of pointers.
3. It lacks an inherent mechanism for describing parallelism.
4. Its debugging tools are simulation unaware, i.e., they operate at a level far below that which would be convenient for most simulationists.

HLA was designed to be usable with a wide variety of tools. This generality prompted us to consider its use in conjunction with classical, discrete event simulation tools which have no built-in distributed simulation capabilities.

Section 2 of this paper presents a short introduction into HLA. Section 3 discusses the requirements that HLA imposes on commercially available, off-the-shelf simulations tools. Section 4 presents several general solutions for adapting a simulation tool for use with HLA. Section 5 presents a detailed description of a solution we developed for SLX.

2 HIGH LEVEL ARCHITECTURE

HLA is a simulation interoperability standard currently being developed by the US Department of Defense (DMSO 1997). The architecture is defined by:

1. rules which govern the behavior of a distributed simulation, called a *federation*, and the individual distributed components, called *federates*, which comprise the federation. (DoD 1996).

2. an interface specification which defines the interface between each federate and the Runtime Infrastructure (*RTI*). The RTI is responsible for providing communication services and coordination among federates. All communication among federates is conducted using RTI services. Federates cannot communicate with one another directly. Communication is based on the use of *ambassadors*: A federate calls methods of an RTI ambassador object to communicate with the RTI. Conversely, the RTI calls methods of a federate ambassador object to communicate with it. The RTI's ambassadors are built into the RTI. Federate ambassadors must be provided for each federate.

3. an *Object Model Template* (*OMT*) which provides the framework for defining federations and federates. The OMT is the foundation of the object-oriented world view embraced by HLA. A Simulation Object Model (SOM) must be supplied for each federate in accordance with the OMT. A federation can be viewed as a contract between federates on how a common federation execution is intended to be run (DoD 1997a).

The RTI provides several services which support interoperability. One of the most important services for event-driven simulation tools is the coordination of their simulation clocks. The time management services provided by HLA allow the transparent running of federates under different time regimes including real-time, time-stepped, and event-driven simulations (DMSO 1997a).

3 REQUIREMENTS

The requirements imposed on simulation and animation tools by the HLA are considerable. They can be separated into two categories: (1) requirements derived from the HLA Interface Specification and the resulting programming paradigm, and (2) requirements derived from being part of a distributed simulation in general. Both categories will be discussed in the following sections, and some general solutions will be discussed.

3.1 Requirements derived from the HLA programming paradigm

One of the major building blocks of HLA is the Interface Specification. HLA defines a two-part interface which federates are required to use for communicating with the Runtime Infrastructure (RTI). This interface is based on an ambassador paradigm. A federate communicates with the RTI using its RTI ambassador. Conversely, the RTI communicates with a federate via the federate's ambassador. From the federate programmer's point of view these ambassadors are objects and the communication between the participants is performed by calling methods of these objects. (Figure 1)

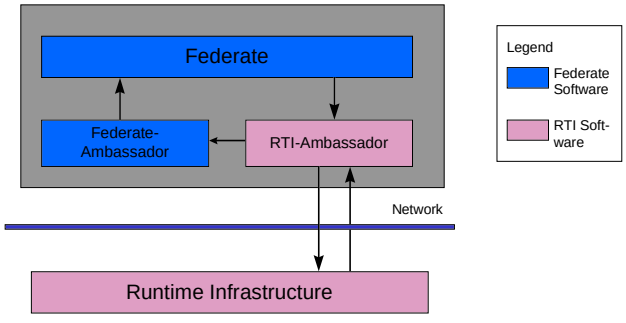

Figure 1: The Ambassador Paradigm

In versions of the RTI provided by DMSO, the RTI ambassador object is provided in a software library which is used by a federate. This library can be dynamically connected to the federate during model execution; i.e., federates do not have to be bound to the RTI in advance of execution. The Windows NT version of the RTI uses Microsoft dynamic link libraries (DLLs). RTI implementations for other operating systems use similar facilities.

A federate is responsible for constructing proper calls of methods of the RTI ambassador object. The definitions a federate needs to be able to construct proper calls are provided as part of HLA. For simulations developed in C++, these definitions are provided in the form of header (".hh") files. For federates developed in languages other

than C++, alternative means must be used to construct calls to the RTI ambassador.

A federate must also provide an ambassador object which contains "callback" functions (methods) which can be called by the RTI. These methods must conform to the HLA standard. The skeletal definitions of the calling sequences of the required federate ambassador methods are provided in the form of C++ header files. The header files contain an abstract object class for the federate ambassador. To construct a federate, one fills in the skeleton with actual code by deriving a concrete object class from the abstract one. While this is straightforward for federates written in C++, alternative means must be developed to construct ambassadors for simulations developed in other languages.

3.2 Requirements derived from being part of a distributed simulation

Being member of a distributed simulation imposes some general problems that stand-alone simulations do not have to deal with.

Synchronization

Participants in a distributed simulation have to coordinate their local advances in logical simulation time. To do so they have to take into account their mutual dependencies. In previous approaches to distributed simulation, developers had to devise and implement their own time management algorithms (Fujimoto 1990).

HLA provides a mechanism for coordinating logical simulation time of event driven simulators. Actually HLA offers much more than simple "coordination": it provides a transparent time management that makes it almost unnecessary for federates to know the time regime used by other federates.

In order for this to work, the RTI requires federates to request their time advances by calling the appropriate methods of the RTI ambassador object. For discrete event simulation tools, which are the focus of this paper, the interface function "nextEventRequest" is of major interest. A federate must issue a call to this function before advancing in its logical time. A federate is then expected to honor certain callback functions that the RTI might call. In particular, it must wait for a "timeAdvanceGrant" callback.

The latter is the typical scenario for a conservative synchronization. In such a scenario the need to constantly request the time advances could be automated by adding a new synchronization thread to the simulation model (Figure 2). This thread should have the lowest priority to ensure that at a certain logical simulation time all actual "simulation" events have been processed before the time stamp of the next event is determined. This way no event can be scheduled with a time stamp lower than the one the request is issued for. As a prerequisite for this kind of synchronization to work the simulation tool has to offer a function for the determination of the next scheduled event time (e.g. GPSS/H: NAC1, SLX: next_imminent_time( )).

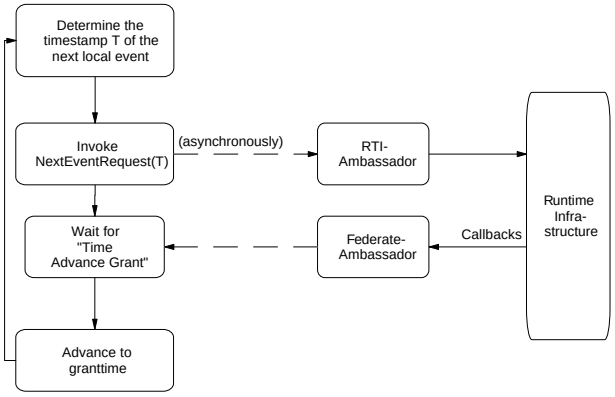

Figure 2: Principle of the Synchronization Thread

HLA also supports optimistically synchronized federates. Although we are currently experimenting with this approach, our experiments are not discussed in this paper.

In a distributed simulation a model also has to react to *external* events. The occurrence of such an event is communicated to the federate through certain callback functions in the federate (e.g. attribute updates, interaction messages etc.) which are called by the RTI.

The two fundamental actions a simulation must perform in order to achieve synchronization are to:

1. determine the next logical event time of the simulator and request an advancement to this time at the RTI; and

2. wait for a time advance grant from the RTI and react to any external events that have been sent to the federate.

Data exchange and data representation

A simulator must also have the capability of receiving and sending data about the objects comprising a model. This is done by using HLA interface functions and is a relatively conventional programming task. Additionally, there is the need to store data about remote objects in the local simulation tool. This is necessary because under HLA, such semantic information cannot be stored within the RTI; i.e., all object representation is maintained solely within the federates. Therefore a simulator has to somehow be able to receive updates and store them locally in order to make future use of them.

4 THE FOUR GENERAL SOLUTIONS

Given the requirements discussed in section 3 we have identified 4 general possibilities for integrating existing simulation tools into the High Level Architecture.

1. **Re-Implementation of the tool with HLA-extensions**
   This solution is, of course, the most obvious one. If the source code of a tool is available and well documented this is the most straightforward and probably the least complicated solution. Skopeo (Lorenz and Ritter 1997), a web-based animation tool developed at the University of Magdeburg, is an example of an animation tool for which HLA-compliance has been accomplished in this manner.

2. **Extension of intermediate code**
   Some simulation tools translate model descriptions written in a tool-dependent modeling language into another programming language (e.g. C++). This intermediate code is then compiled to an executable file. It is possible to modify this code to realize the HLA extensions. Since this code is compiler *generated*, an automated solution is desirable.

   Examples for tools that could theoretically be extended in this way are ACSL, MODSIM, and JavaGPSS (Klein et al. 1998).

3. **Usage of an external programming interface**
   This solution is well suited for tools that offer an open and extensible architecture. The tool should offer a library interface (in Windows: a DLL interface) with the ability to call arbitrary functions or methods in these libraries. Additionally the tool should make it possible to to implement callback functions or methods.

   Section 5 of this paper describes a solution for the simulation tool SLX which is based on this approach.

4. **Coupling via a gateway program**
   The last solution for tools which can not be connected to the RTI by any of the prior methods is the development of a gateway program. The gateway program could communicate with the simulation tool via appropriate means (e.g. files, pipes, network) depending on the capabilities of the simulation tool (Figure 3) .

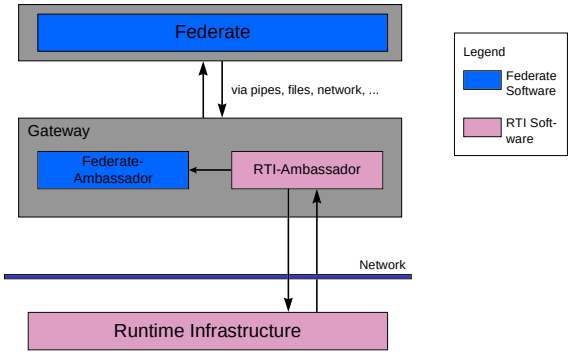

Figure 3: Principle of the Gateway Solution

The gateway program would translate between the simulation tool and the RTI and therefore be the actual member (federate) in the HLA federation. Gateway programs are a common means for integrating existing applications (e.g. legacy applications such as DIS simulations) into HLA (Cox et al. 1996). A gateway could also be used for connecting older simulation tools for the DOS operating system such as GPSS/H to the RTI.

5 THE SOLUTION FOR SLX

SLX is a new discrete event simulation tool for the Windows 95/98/NT operating systems (Henriksen 1996). SLX has an open software architecture. It has a library interface which allows SLX programs to call functions in any standard Windows DLL. This is one of the basic prerequisites for connecting a commercial tool like SLX to the RTI without modifying the actual source code of the tool. In addition to this interface, SLX also offers the extraordinary feature of automatically generating C/C++ header files which describe the contents of SLX objects in C/C++ syntax. A C/C++ function which is placed in a DLL is easily called from SLX, and it can retrieve and modify the attributes of SLX objects.

Although SLX is a very open tool, not all of the requirements stated earlier can be fulfilled directly. For example, it is not possible to directly create instances of C++-objects *inside* SLX. Therefore it is not possible to call the methods of the RTI ambassador object directly from SLX. Furthermore it is not possible to directly implement the federate ambassador object with its callback methods inside SLX. In addition, there are some differences between SLX and C/C++ data types. Although SLX has a C-like syntax, there are differences between SLX and C in lowest-level data manipulation. For example, SLX protects users from many of the pitfalls of C (e.g. freeing memory although there still is a pointer pointing to it, exceeding the boundaries of arrays, mismatched types, etc.). SLX strings are different from those of C. SLX internally keeps track of

the current and maximum length of strings and does not use the zero-termination of C.

These examples illustrate that some data type conversion is unavoidable when data are manipulated by both SLX and C/C++. Since the RTI is C++ based, some type conversions are necessary to use RTI functions from an SLX program. It is therefore not possible to directly access the ambassador objects. Given these shortcomings, we must ask whether they outweigh the gains of using SLX as a modeling tool.

Fortunately, the solution to the problems stated earlier is very straightforward: a simple "wrapper" library callable from SLX can be developed to wrap around the RTI (Figure 4).

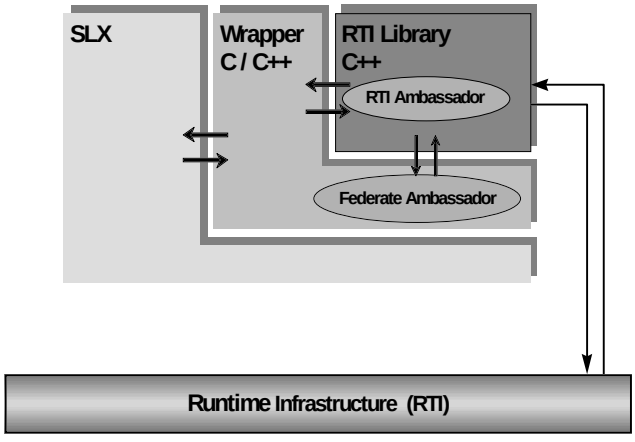

Figure 4: Connecting SLX to the RTI Using a Wrapper Library

Communication between SLX and RTI

The methods of the RTI ambassador object (C++ functions) that have to be called from SLX are wrapped by "normal" C-functions which can be called by SLX. This is done on a 1:1 basis for each HLA function required. The wrapper functions perform the necessary conversions between SLX data types and RTI data types. They also simplify RTI programming efforts for the SLX user. The process of dealing with the somewhat confusing RTI types (attribute handle set, attribute value pair set, ...) is hidden from the SLX user.

Communication between RTI and SLX

Another task that the wrapper library has to perform is the implementation of the federate ambassador object. The federate ambassador is responsible for receiving all kind of data from the RTI. This reception process is handled internally by the wrapper library. Since the wrapper library cannot *call* SLX to tell that something has happened, a mailbox-principle is used: The federate ambassador stores the data that were received in an SLX object in a fixed structure. The SLX model can then access this object to query the things that happened outside. This is both straightforward and suitable for performing the required tasks.

In addition to this static object, our solution introduces dynamic structures for the actual objects that are being modeled. When a simulation models certain objects and interactions according to its HLA simulation object model, these HLA logical structures are mapped into real SLX objects which can be accessed from SLX and the wrapper DLL. This simplifies the process of sending and receiving attribute updates or interactions. The problem with this approach is that the wrapper library does not know the structure of these SLX objects at *compile time*. Therefore the DLL has to calculate the address of each object attribute at runtime. However, the SLX run-time environment provides C/C++ callable library routines which the DLL wrapper uses to retrieve attribute address information using attribute *names* or pointers. This guarantees error-free mapping into SLX objects.

Synchronization Issues

The HLA programming paradigm expects a federate to *tick* the RTI to trigger the reception of any currently pending callback invocations by calling the *tick*-method of the RTI ambassador. This is usually done during the interval between requesting a time advance and waiting for the according time advance grant. As a simplification for the SLX user, this is handled internally by the wrapper library. The SLX user simply requests to advance to the next logical event time and then (after a while) receives a time advance grant.

```
forever
{
    NextEventTime= next_imminent_time();
    grantTime = RTI_NextEventRequest( NextEventTime);
    wait until (time == grantTime);
    ... // query any external events
    yield; // hand over control to other simulation threads
}
```

Figure 5: Code Fragment of the Synchronization Thread Used in SLX Corresponding to the General Shown in Structure Figure 2

If an external event has to be processed, the grant time may be smaller than the actual time advance that had been requested. The user then is expected to check the static and dynamic SLX objects for any data or events that need to be processed. After that the normal simulation execution can proceed; possibly by issuing the same time advance request again. Figure 5 shows the suggested usage of the time advance functions provided by the wrapper library for SLX.

To ensure the behavior outlined above the functions of the wrapper library for requesting the time advances work synchronously, i.e. the wrapper library keeps control until a time advance grant is received. This is different from the original RTI ambassador functions which work asynchronously. Figure 6 shows a simplified code fragment illustrating the basic algorithm of the time management functions provided by the wrapper library.

```
//global variables

boolean          timeAdvGrant;
double           grantTime;
...

double RTI_NextEventRequest (double NextEventTime)
{
        try
        {
                timeAdvGrant = RTI::RTI_FALSE;
                ms_rtiAmb->nextEventRequest(NextEventTime);
        }
        catch ( RTI::Exception& e )
        {
                return (-1);
        }
        while (timeAdvGrant == RTI::RTI_FALSE)
        {
                int eventsToProcess = 1;
                while ( eventsToProcess )
                {
                        eventsToProcess = ms_rtiAmb->tick();
                        //tick the RTI until timeAdvanceGrant callback
                        //is invoked (which will set timeAdvGrant and
                          grantTime)
                }
        }
        return (grantTime);
}
```

Figure 6: Code Fragment of the RTI_NextEventRequest Function of the Wrapper Library for SLX

Performance

The wrapper library offers acceptable performance. We have successfully tested several sample federations with up to 4 federates (3 of them SLX) which were all strictly synchronized via the RTI. A part of a reference federation modeling a traffic light scenario is shown in figure 7.

The execution speed of the same model implemented in a monolithic fashion without HLA (as one single SLX model) would be much faster (between 10 to 50 times). However, the primary goal of HLA is not to gain speedups through parallel processing of models. Rather, the main focus of HLA is to facilitate interoperability among a wide variety of simulation tools. To improve general RTI performance is one of the major goals of the current development efforts for an RTI version 2.0.

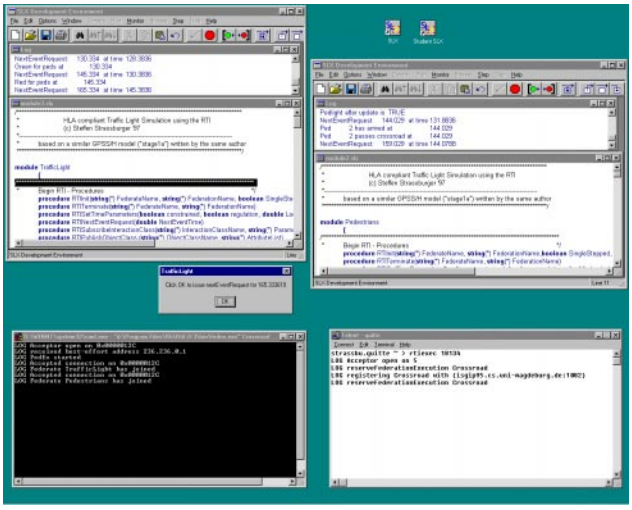

Figure 7: HLA Federation With Two SLX Federates Running on the Same Computer

It was possible to run all three SLX federates of this federation on one high performance PC (Dual-Pentium II System with 64 Mbytes memory).

5.1 SAMPLE APPLICATIONS

The solution for SLX has been successfully tested in the following sample applications:

1. Federation “Traffic Light”: Three SLX federates (car traffic, pedestrian traffic, and traffic light control) were developed to simulate a classical traffic light scenario (Klein and Straßburger 1997).

2. Federation “Manufacturing”: Two SLX federates (a manufacturing federate, a command and control federate), and a visualization federate (the animation system Skopeo mentioned earlier) were implemented (Schulze et al. 1998).

3. Federation “Shipping Agencies”: A variable number of SLX federates, each modeling a shipping agency can interact with each other in this federation (Schulze et al. 1998a).

4. Federation “Free Driving”: A single SLX federate modeling psycho/physical vehicle-following behavior was coupled with a real-time training simulator. The SLX model had to appropriately react to the vehicle modeled by the training simulator (Klein et al. 1998a).

6 CONCLUSIONS

We have identified a basic set of requirements that simulation tools must meet in order to make them HLA compliant. We have suggested four general solutions for adopting existing discrete event simulation tools to HLA. The basic ideas behind these solutions can also be applied for other tools which have to be made HLA-compliant.

We introduced a model-independent HLA connection for the simulation tool SLX. This solution provides access to a basic set of HLA functionality from SLX. It comes with an easy to use interface for "doing" HLA without having to care about the "dirty" programming work with the RTI.

Certain higher techniques of HLA remain to be implemented. Most of this work is in the area of HLA data distribution management, which had not been implemented in the DMSO-provided versions of the RTI we used. RTI support for this service group is announced for later this year and will then be incorporated in future versions of the wrapper library for SLX.

The solution for SLX could, in principle, be used for connecting other simulation tools with a standard DLL interface to the RTI. Modifications are only necessary for the data transfer back to the simulation tool, since this is dependent from the internal data structure representation in the respective tools.

REFERENCES

Cox, A., D. Wood, M. Petty, K. Juge. 1996. Integrating DIS and SIMNET into HLA with a gateway. In *Proceedings of the 15th DIS Workshop on Standards for the Interoperability of Defense Simulations*, Orlando FL, September 16-20 1996, pp. 517-525.

Defense Modeling and Simulation Office (DMSO). 1997. The High Level Architecture Homepage. URL http://hla.dmso.mil/.

Defense Modeling and Simulation Office (DMSO). 1997a. HLA Time Management Design Document, *Version 1.0, dated 15 August 1996*. Available online at the HLA Homepage (DMSO 1997).

Department of Defense (DoD). 1996. High Level Architecture Rules, Version 1.0, dated 15 August 1996. Available online at the HLA Homepage (DMSO 1997).

Department of Defense (DoD). 1997. High Level Architecture Interface Specification, Version 1.2. Available online at the HLA Homepage (DMSO 1997).

Department of Defense (US). 1997a. High Level Architecture Object Model Template, Version 1.1, dated 12 March 1997. Available online at the HLA Homepage (DMSO 1997).

Dorwarth, H.; P. Lorenz; K. C. Ritter; and T. J. Schriber. 1997. Towards a Simulation and Animation Environment for the Web. In *Proceedings of the 1997 Winter Simulation Conference*, eds. S. Andradóttir, K. Healy, D. Withers, B. Nelson, pp. 1338-1344.

Fujimoto, R. 1990. Parallel Discrete Event Simulation. In Communications of the ACM, 1990, no. 10, pp. 30-53.

Healy, J. H. and R. A. Kilgore. 1997. SILK™: A Java Based Process Simulation Language. In *Proceedings of the 1997 Winter Simulation Conference*, eds. S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, pp.475-482, SCS, Atlanta

Henriksen, J.O. 1997. An Introduction to SLX™. In *Proceedings of the 1997 Winter Simulation Conference*, eds. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, pp.559-566, SCS, Atlanta

Klein, U. and S. Straßburger. 1997. Die High Level Architecture (HLA): Anforderungen an interoperable und wiederverwendbare Simulationen am Beispiel von Verkehrs- und Infrastruktursimulationen. In Kuhn, A. and S. Wentzel (Ed.), *Proceedings of the 11th Simulation Symposium ASIM 97*. Nov. 11-14, 1997. Vieweg Verlag, pp. 529-534.

Klein, U., S. Straßburger, J. Beikirch. 1998. Distributed Simulation with JavaGPSS based on the High Level Architecture. In *Proceedings of the 1998 International Conference on Web-based Modeling and Simulation* Jan. 11-14, 1998, San Diego.

Klein, U., Th. Schulze, S. Straßburger and H.-P. Menzler. 1998a. Traffic Simulation Based on the High Level Architecture. In *Proceedings of the 1998 Winter Simulation Conference*, eds. Medeiros, D.J. and Ed Watson, SCS, Washington.

Lorenz, P. and K. C. Ritter. 1997. Skopeo: Platform-Independent System Animation for the W3. In Deussen, O. and P. Lorenz (Ed.), *Proceedings of the Simulation und Animation Conference Magdeburg*, March 6-7, 1997. SCS European Publishing House San Diego / Erlangen / Ghent / Budapest 1997, pp. 12-23.

Schulze, Th., U. Klein, S. Straßburger, K.-C. Ritter, E. Blümel, and M. Schumann. 1998. HLA basierte verteilte Simulationsmodelle für die Fertigung. In Preim, B. and P. Lorenz (Ed.), *Proceedings of the Simulation und Animation Conference Magdeburg*, March 6-7, 1998. SCS European Publishing House San Diego / Erlangen / Ghent / Budapest 1998. pp.19-31.

Schulze, Th., G. Lantzsch, U. Klein, and S. Straßburger. 1998a. Interoperabilität zwischen Simulationsmodellen auf Basis der High Level Architecture. In *Erfahrungen aus der Zukunft*, Tagungsband 8. ASIM-Fachtagung Simulation und Logistik, eds. Mertins und Rabe, IPK Berlin 1998, 369-379.

AUTHOR BIOGRAPHIES

STEFFEN STRASSBURGER holds a Master's degree in Computer Science from the Otto-von-Guericke University, Magdeburg. He is currently working towards his PhD degree at the Institute for Simulation and Graphics at the same university. His experience with inter-networking and simulation includes a one-year-stay at the University of Wisconsin, Stevens Point. His main research interests lies in distributed simulation and the High Level Architecture.

THOMAS SCHULZE is an Associate Professor at the Otto-von-Guericke-University in Magdeburg in the Department of Computer Science. His research interests include modeling methodology, public systems modeling, traffic simulation, and distributed simulation with HLA. He is an active member in the ASIM, a community for simulation in Germany.

ULRICH KLEIN is a PhD candidate at the University of Magdeburg, Germany. He holds a Master's degree in Industrial Engineering from the University of Karlsruhe and has been involved in Emergency Management since 1992. He has two years of experience as Project Manager for Command, Control, and Communication Systems for Public Safety and Security in Europe. His research topics include Emergency Management, Urban Infrastructure Management and Logistics, Geographic Information Systems, and the High Level Architecture.

JAMES O. HENRIKSEN is the president of Wolverine Software Corporation. He was the chief developer of the first version of GPSS/H, of Proof Animation, and of SLX. He is a frequent contributor to the literature on simulation and has presented many papers at the Winter Simulation Conference. Mr. Henriksen has served as the Business Chair and General Chair of past Winter Simulation Conferences. He has also served on the Board of Directors of the conference as the ACM/SIGSIM representative.