On the Integration of HLA into SCEs

Sven Pawletta[1], Wolfgang Drewelow[1] and Thorsten Pawletta[2]

[1]University of Rostock, Department of Electrical Engineering, D-18051 Rostock, Germany, E-mail: sven.pawletta@etechnik.uni-rostock.de; [2]University of Wismar, Department of Mechanical Engineering, D-23952 Wismar, Germany, E-mail: pawel@mb.hs-wismar.de

Presently, two ways exist to build HLA-based simulations: (1) federates are programmed with high-level languages under direct use of RTI libraries; and (2) federates are created with HLA-compliant special-purpose systems (mostly simulation systems). Both methods are not particularly accepted in application domains, where interpreter-based scientific computing environments (SCEs) are preferred. To solve this problem, the possibilities of an integration of HLA into SCEs are examined. The discussion is mainly focused on the following aspects: impact of RTI implementations, RTI/SCE-linkage and design of an HLA interface for interactive/interpretive use. An example implementation for the SCE MATLAB (HLA Toolbox), the results of a benchmark test and a demonstration application are presented.

Keywords: HLA, SCEs, distributed simulation, RTI, MATLAB, CACSD-systems

1. Introduction

The High Level Architecture for modeling and simulation (HLA) has its origin in the military domain. Nevertheless, specifications and supporting software are openly available and can be used for civil projects. Consequently, there is a growing community of HLA users in the civil domain.

Schulze et al. compare in [8] the military with the civil simulation worlds. They determine on the one hand, that the approaches and general methods which are used in both communities are very similar. On the other hand, there are major differences in how simulations are developed. This is also true for HLA-based distributed simulations.

In the military community, federates for a distributed simulation are usually programmed with high-level languages under direct use of a RTI library. This is the original method to create HLA federates.

For civil projects, simulation systems are preferred. Consequently, a number of investigations have been made, in order to enable existing simulation systems to cooperate with HLA [3, 8, 9].

HLA-compliant simulation systems are a great improvement over manual programming for many application fields. Nevertheless, there are domains where even this method is not particularly accepted. One of these domains is control engineering. Here, scientific and technical computing environments (SCEs) have been established as powerful tools for solving analysis and design problems. For these domains, we need a third method in order to build federates for HLA-based simulations within SCEs.

Therefore, this article discusses requirements and solutions for integrating HLA into SCEs. At first, basic terms around HLA and SCEs are introduced. Then, we examine the integration problem. Finally, an example implementation for the widely-used SCE MATLAB, the results of a first benchmark test and a simple demonstration application are presented.

TRANSACTIONS of The Society for Modeling and Simulation International
ISSN 0740-6797/00
Copyright © 2001 The Society for Modeling and Simulation International
Volume 18, Number2, pp.

2. The High Level Architecture (HLA)

The HLA standard defines an architecture for component-based distributed simulations. A component is mainly a single simulation. However, components can also have other functions such as data collection, visualization and process coupling. A HLA component is called a federate. In nontrivial applications, at least two components form a distributed system, which is called federation.

The current version 1.3 of the HLA standard [1] consists of three parts—Rules, Object Model Template and Interface Specification.

The interface specification is the most important part of the standard for the actual work with HLA. In the first section, it defines services in a language-independent manner that have to be provided by a run-time system, which is called Run-Time Infrastructure (RTI), and by the federates. The second section defines calling conventions for the programming languages IDL, C++, Ada 95 and Java.

Since the HLA standard defines only the interface between federates and the RTI, completely different RTI implementations are possible (Figure 1). A run-time system for IP-based networks is available from the Defense Modeling and Simulation Office (DMSO).

3. Scientific Computing Environments (SCEs)

Today, SCEs are the dominating tools for computation and visualization in many engineering domains. Originally, they were created to support the computer-aided control system design (CACSD) process. Up to the mid-1990s, there was a relatively

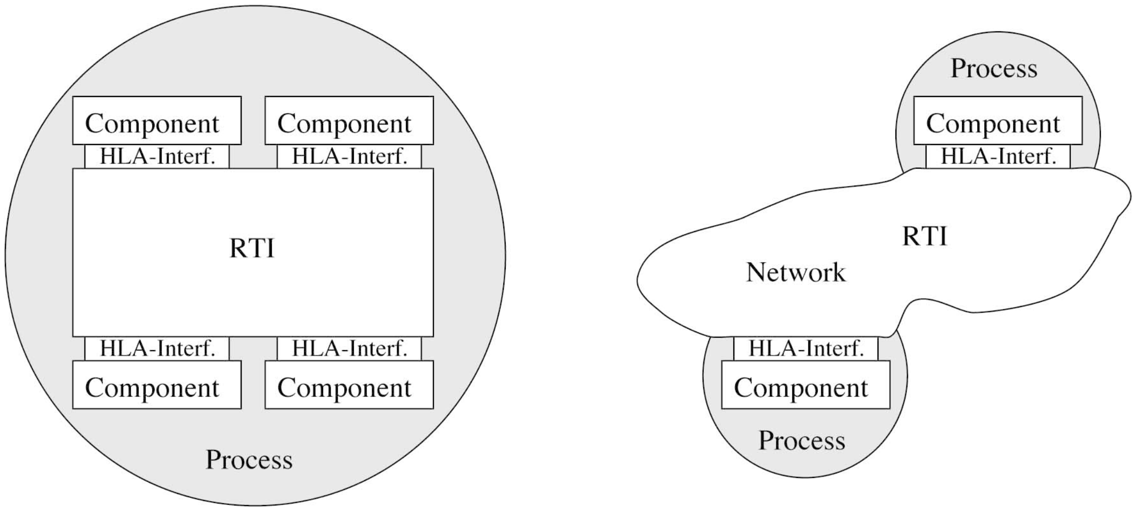

Figure 1. Possible RTI implementations

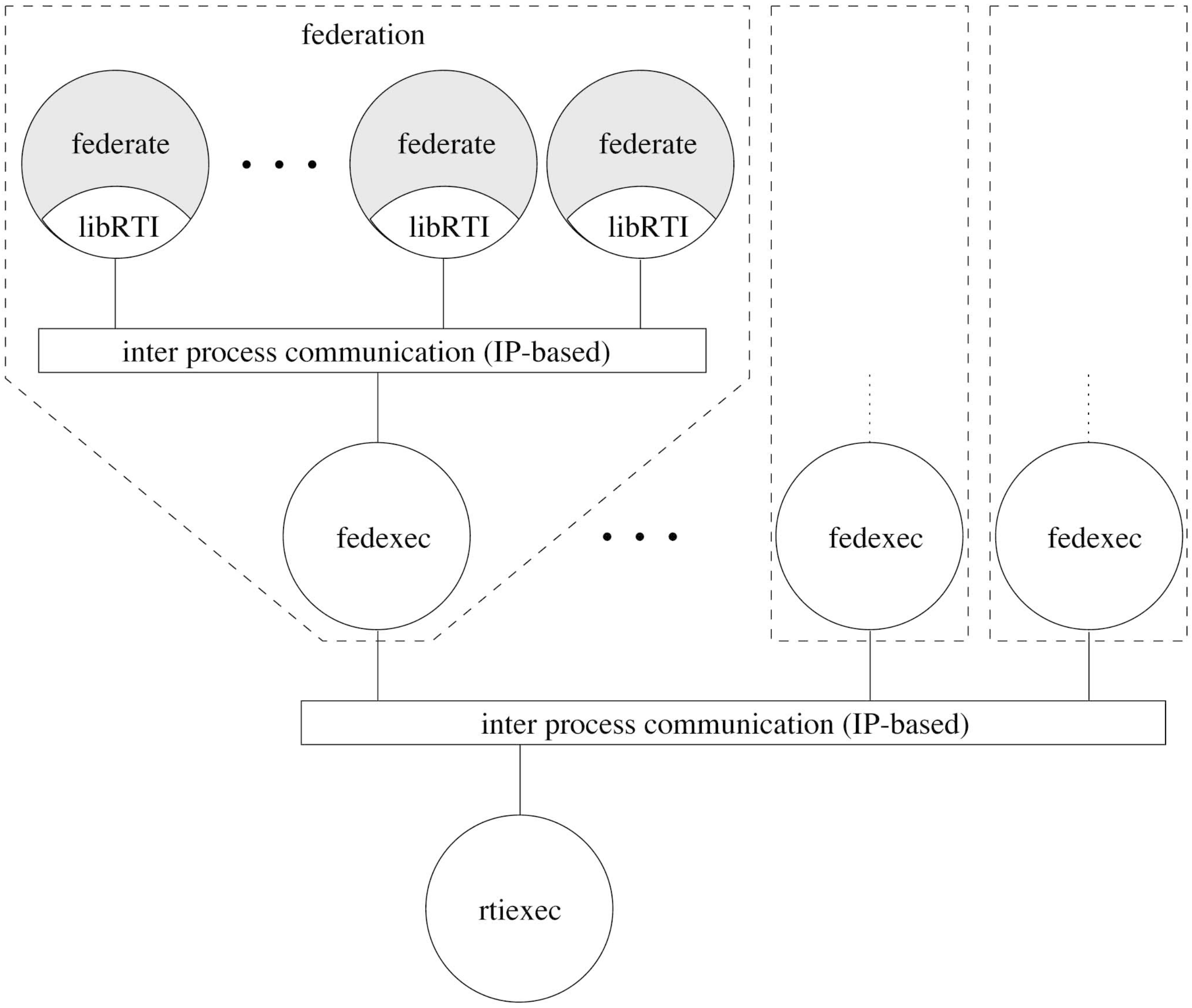

Figure 2. Structure of DMSO RTI 1.3

clear distinction between computer numeric systems (CNS), e.g., MATLAB, Matrix$_X$) and computer algebra systems (CAS), e.g., Maple, MuPAD). Recently, there are also hybrid systems available (e.g., MATLAB/Maple-linkage).

Primarily, SCEs are used to develop prototypes. This is very different from conventional programming with high-level languages such as Fortran, C and C++, which are used to produce run-time and memory-efficient code.

The two key concepts for SCE-based *rapid prototyping* are:

- SCEs operate *interpretively*. Hence, routines can be executed directly from a command prompt. Moreover, user-defined routines can be executed without compilation. This kind of interactivity guarantees a very short error correction cycle.
- SCEs provide integrated *array-oriented programming languages,* which support implicitly typed data objects, whereby, scientific and technical problems, which are mainly based on vector or matrix calculations, can be coded very efficiently.

Accordingly, the way of solving problems using SCEs is completely different from conventional programming. One-time analysis and design problems can be solved easily by entering the data set and invoking one or more existing routines. If a problem has to be solved several times with different data sets, the necessary routine sequence is written in a script, which can be executed in the same way as a predefined routine. Unlike conventional programming which involves the implementation of a self-contained application program, the principle of solving problems with SCEs is the gradual extension of already available functionality. The result of such a solution process is usually the construction of a new library or toolbox containing a number of problem-specific routines.

In the following section, we use MATLAB as example system to discuss the HLA/SCE integration. MATLAB is a model for many other SCEs and provides probably the most comprehensive collection of sophisticated methods relevant to simulation tasks. This includes routines for numerical integration, multidimensional data visualization and animation, containment in the base-system, and tool-boxes or subsystems for continuous, discrete and hybrid simulation (ODE Tbx., PDE Tbx., Simulink, Stateflow, GPSS Tbx.; [2, 4, 7]). Even toolboxes for parallel and distributed computing are available (DP Tbx., PT Tbx., MultiMatlab Tbx; [5]).

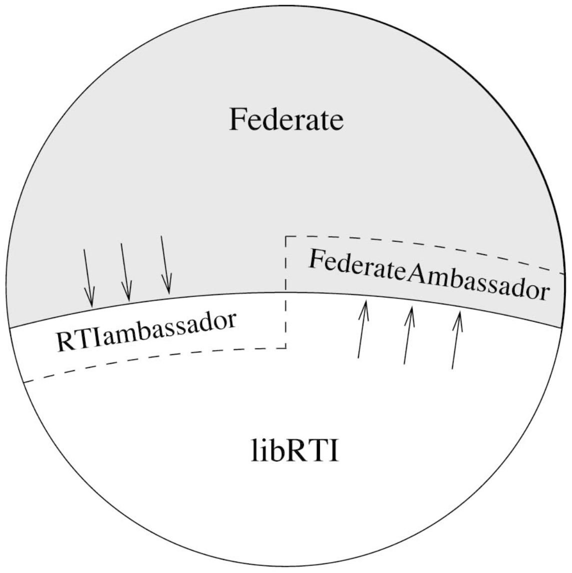

Figure 3. The RTI/Federate interface

4. HLA/SCE Integration

There are two general approaches to add HLA functionality to SCEs. The first approach is the implementation of a RTI, which is specifically designed to operate in conjunction with a SCE. Because of the necessary implementation effort, this seems to be an unfavorable approach. Nevertheless, it has also notable advantages—A full HLA-compliant RTI implementation has to provide about 130 services, but many applications use only a small subset of them. A specifically implemented RTI could be scaled according to the requirements of an application domain. Moreover, it could be ported to any desired platform (e.g., non-IP networks, real time operating systems).

The second approach is the integration of an existing RTI into a SCE. This approach reduces significantly the implementation effort and shall be examined in the following paragraphs.

4.1. RTI/SCE Linkage

As mentioned in Section 2, the implementation of a run-time infrastructure is not standardized. Nevertheless, the problem of creating a suitable linkage between a RTI and a SCE is mostly independent of the concrete RTI implementation. The following discussion is based on RTI implementations from the Defense Modeling and Simulation Office (DMSO), which are openly available.

The components of the DMSO RTI 1.3 and their relations are shown in Figure 2. The run-time system management (*run-time infrastructure executive*) is implemented as a separate program (rtiexec) and controls the execution (starting and terminating) of federations. It is the global part of the RTI.

The federation management (federation executive) is likewise implemented as a separate program (fedexec), which controls joining as well as leaving of federates to or from a running federation.

The crucial component for the RTI/SCE linkage is the RTI library (Figure 3). It provides a bidirectional interface consisting of the concrete class *RTIambassador* and the abstract class *FederateAmbassador*. The *RTIambassador* is the interface to the RTI services, which can be requested by federates (federate-initiated services). The *FederateAmbassador* defines prototypes for all methods that a federate has to provide for the RTI (RTI-initiated services).

Almost all SCEs provide an application programming interface (API) for extensions written in Fortran or C. In the case with MATLAB, it is called MEX-interface (MATLAB external interface). Although these interfaces were originally designed for linking individual external routines, they provide sufficient functionality to integrate an entire function library.

Figure 4 shows the principle of such a binding for MATLAB. Since the library functions operate independently, each routine

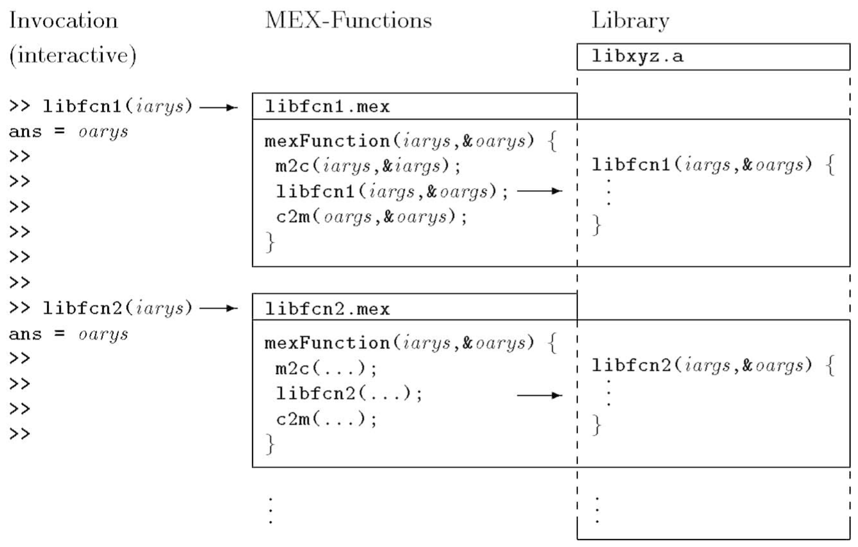

Figure 4. Binding of an external library via the MEX-interface

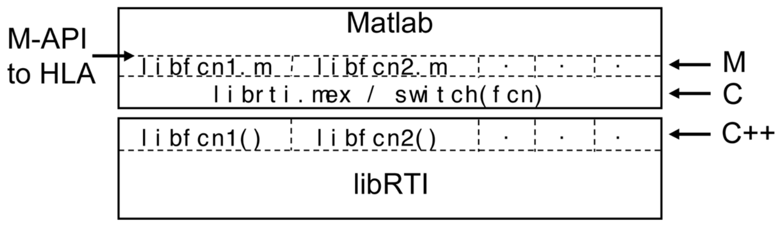

Figure 5. Binding of a RTI library

is linked via a separate gateway function (MEX-function), which usually carries the same name as the library function. The major task of the gateway functions is the conversion of the SCE calling parameters (arrays) into C function arguments (m2c(*iarys*, *&iargs*)), before a library routine is called, and the re-conversion of the result (c2m(*oargs*, *&oarys*)), after the routine returns.

Unfortunately, a RTI library cannot be linked in such a simple manner, because it contains global and persistent data objects. In this case, the binding mechanism has to ensure, that the entire library code is always loaded as a single module. Furthermore, dynamic memory management must not automatically unload this module under any circumstances (*module locking*). Otherwise, status information would be lost.

Therefore, the binding of a RTI library has to be realized with a single gateway function as shown in Figure 5. The selection of the desired RTI function has to be done by a numeric code (*fcn*). In order to provide a simple calling syntax, this calling mechanism has to be hidden by a suitable user interface inside the SCE.

4.2. An Interactive HLA Interface

Actually, the binding of a RTI library to a SCE, as discussed in the last paragraph, is only the first step of an HLA/SCE integration. In order to allow the SCE typical interactive way of working, an HLA interface within a SCE has to meet the following criteria.

(1) *Interactively usable designators.* RTI routines have fully descriptive designators. This leads to a very long winded calling syntax such as *rtiAmb.unconditionalAttributeOwnershipDivestiture*(...), which cannot be managed interactively. Therefore, abbreviated routine designators have to be used in a SCE suitable HLA interface.

Furthermore, instance designators are not necessary because within a single federation, a federate needs only one *RTIambassador* and one *FederateAmbassador* instance. In order to create a separate designator space, a short prefix is more suitable.

(2) *Vectorization and data objects with implicit types.* RTI routines perform only scalar operations with elementary data objects as parameters. The repeated execution of an operation with different parameters, which is often needed in applications, has to be programmed by additional control structures. Since inputting control structures interactively is time-consuming and fault-prone, the routines of a SCE suitable HLA interface should be vectorized as much as possible.

Moreover, the use of the SCE typical data objects with implicit type and dimension information leads to a simplification of the invocation syntax and reduces the number of interface routines (e.g., all the auxiliary classes contained in the DMSO RTI library with more than 80 routines are unnecessary in a SCE/HLA interface).

(3) *Interactive callbacks.* In conventional programming, RTI-initiated services are supplied in form of callback-routines, which can be invoked by an RTI library over the *FederateAmbassador*. However within a SCE, it is uncommon to code routines "in advance." In order to solve this problem, default callbacks for all RTI-initiated services have to be already provided within a SCE/HLA interface.

If such a default callback is invoked by the RTI, an interpreter within the local scope of the callback routine is started. Parameters transferred to the callback routine can be examined and further action can be executed interactively. After leaving the callback routine (entering return), the control flow returns to the RTI. If the user implements specific callback routines, the default routines are overlaid.

(4) *Selectable exception handling.* RTI routines do not handle exceptions on their own, but return the exceptions back to the caller. In SCEs, this method is not always suitable. Often, the exception handling should take place within the routine and the exception types *error* and *warning* are sufficient. If a more complex exception handling is necessary, an additional return parameter is requested during the invocation of the routine. Using this parameter, an exception is returned to the caller and the default exception handling is disabled.

5. The HLA Toolbox

The described approach of integrating HLA into an SCE using an existing RTI library, as well as an SCE suitable HLA user interface, have been implemented with the DMSO RTI 1.3 for the SCE MATLAB. The HLA Toolbox [6] provides, for comparison purposes, an object-oriented and a conventional function-oriented HLA interface.

In order to examine the run-time overhead of MATLAB-based federates in comparison with C++ federates, the demonstration program HelloWorld, which is included as source code in the DMSO RTI distribution, was implemented with both interface versions of the HLA Toolbox. The results of the investigation are shown in Figure 6. The curves indicate the run-times of 10 successive executions of a federation, consisting of two federates of the same implementation: C++(OO) – C++ object-oriented, M(OO) – MATLAB object-oriented, M(FO) – MATLAB function-oriented.

Surprisingly, only the object-oriented MATLAB implementation shows an obvious run-time overhead. The function-oriented MATLAB implementation needs even less run-time than the C++ reference implementation.

In order to demonstrate the usage of the HLA Toolbox in a more realistic application, a simple traffic simulation has been developed (Figure 7). In this application, several MATLAB-based federates simulate and visualize the car and pedestrian traffic at a traffic light crossing. In detail, the following federates are included: time-stepped, event-driven and real-time simulations of the car and pedestrian traffic, event-driven and real-time control programs for the traffic lights, and a visualization federate without time management. The commented MATLAB sources of all federates are given in [6].

6. Conclusions

With the integration of HLA into SCEs, a third method for building HLA-based simulations becomes available. This new method

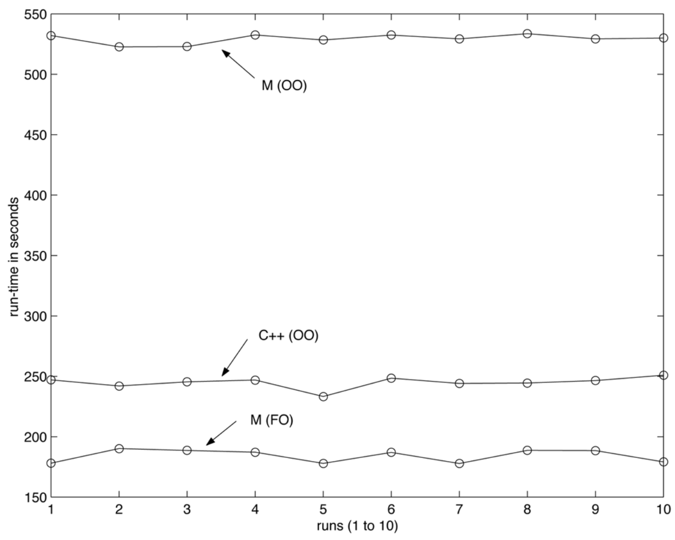

Figure 6. Run-time comparison

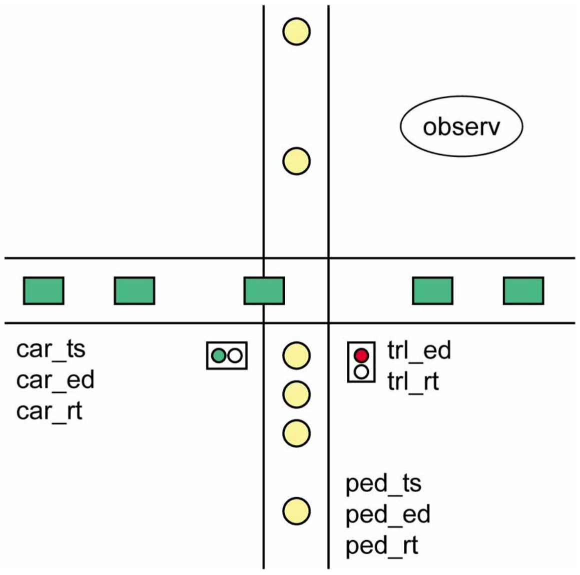

Figure 7. A simple traffic simulation

is especially important for application domains where SCEs are the preferred tools.

The usage of an existing RTI reduces significantly the implementation effort for the HLA extension of a SCE, and guarantees interoperability with other HLA-compliant federates and systems.

With the HLA Toolbox for MATLAB, the authors have shown that the integration approach is feasible in two steps. The first step is the linkage of an existing RTI library with the SCE. This step is a nontrivial technical problem. The second step is the construction of an interactively usable HLA interface within the SCE. This step is a technical as well as a conceptual problem. Our next investigations will deal with the inclusion of the MATLAB-based special-purpose tools Simulink (continuous block-oriented simulation, [4]) and MatlabGPSS (discrete process-oriented simulation, [2]) into HLA-based distributed simulations.

7. References

[1] DMSO. HLA Homepage. http://hla.dmso.mil/.

[2] W. Drewelow, S. Pawletta, and T. Pawletta. Hybrid and process-coupled transaction-oriented simulation in SCEs. In *Proc. Simulation and Visualization SimVis99,* Magdeburg, Germany, pp 181–192, SCS Int., Ghent, Belgium, 1999.

[3] U. Klein, S. Strassburger, and J. Beikirch. Distributed Simulation with JavaGPSS, based on the High Level Architecture. In *Proc. Int. Conf. on Web-based Modeling and Simulation,* San Diego, 1998.

[4] MathWorks. Documentation set.
http://www-europe.mathworks.com/access/ helpdesk/help/ fulldocset.shtml.

[5] S. Pawletta. *Extension of a Scientific and Technical Computing and Visualization System (SCE) to an Environment for Developing Parallel Applications.* ARGESIM/ASIM-Verlag, Wien, 2000. 146 pp (in German).

[6] S. Pawletta, T. Pawletta, and W. Drewelow. *High-Level Architecture Toolbox (HLA Toolbox) for Use with Matlab: User's Guide and Reference Manual Version 0.* Inst. of Automatic Control, Univ. of Rostock, 2000.

[7] T. Pawletta, S. Pawletta, and W. Drewelow. Process-oriented simulation in interactive SCEs. In *Proc. Simulation and Visualization SimVis98,* Magdeburg, Germany, pp 181–194, The Society for Computer Simulation International, Ghent, Belgium, 1998.

[8] T. Schulze, S. Strassburger, and U. Klein. Migration of HLA into civil domains: Solutions and prototypes for transportation applications. *SIMULATION*, Vol. 73, No. 5, pp 296–303, 1999.

[9] S. Strassburger, T. Schulze, and G. Lantzsch. *Simplex 3 and SLX - coupled with HLA. In Proc. 13. Symp. Simulationstechnik ASIM99,* Weimar, Germany, pp 331– 336, SCS International, 1999.