HLA-based Simulation within an Interactive Engineering Environment

Sven Pawletta, Wolfgang Drewelow
University of Rostock
Department of Electrical Engineering
D-18051 Rostock, Germany
sven.pawletta@etechnik.uni-rostock.de

Thorsten Pawletta
University of Wismar
Department of Mechanical Engineering
D-23952 Wismar, Germany
pawel@mb.hs-wismar.de

Abstract

This article discusses prerequisites and possible methods of a Matlab/HLA integration and presents a novel concept of an HLA interface that is suitable for interactive usage. The result of a performance comparision study of a demonstration program on different Matlab/HLA interface implementations is given.

1. Introduction

The *High Level Architecture* (HLA) is a relatively new technology in the field of simulation. The primary objectives of HLA are re-usability and interoperability of simulations. A basic approach of HLA is the standardization of interfaces and data models on a highly abstract level. HLA is founded on a very general *component-based simulation* perspective. A component is a single simulation. It can also be other kind of programs such as visualization and real-time process coupling. With this approach HLA leads inevitably into the area of distributed simulations. In contrast to other distributed processing concepts, run-time minimization or resource maximization do not belong to the primary objectives of the HLA approach.

In recent years, many people have been wondering whether this HLA approach can be applied profitably to non-military areas ([13, 12, 4, 1]). At present, there exist two major approaches for realizing HLA-based simulation studies:

1. Implementation (programming) of components with the help of conventional higher programming languages, such as C++, ADA 95 and Java, using a HLA application programming interface ([2]).

2. Implementation (modelling) of components with the help of an HLA-compliant simulation system or other HLA-compliant special-purpose systems ([14, 5]).

The first approach provides full flexibility in HLA functionality. For complex applications that may contain various components for discrete, continuous and hybrid simulation, visualization, data recording, process coupling etc., this approach is rather expensive in terms of development and testing. In addition, special programming knowledge in parallel and distributed systems is necessary.

In fact, the application of the second approach will not have the problems faced by the first approach. Here, the development of components is possible without special programming knowledge. However, the transparency of the HLA functionality will be reduced depending on the type of system used. Currently, there are only a few HLA-compliant systems available, and the development of extensions of existing systems is still a research challenges.

For the authors, the following question was the starting point for their research in HLA-based simulation:

> *Is the HLA approach applicable in control engineering analysis and design procedures?*

In order to solve analysis and design problems in the field of automatic control successfully, it is extremely important to be able to produce prototype solutions easily and fast. This is usually not possible by conventional programming using high-level languages. Hence, the first HLA approach is not suitable for such application areas.

On the other hand, scientific and technical development environments such as Matlab, Matrix-X, Scilab and Octave have been proven to be suitable and can be used as universal platforms[1] for engineering analysis and design problems. These systems, SCEs (*scientific and technical computing environments*), represent highly effective environments for performing complex mathematical and technical tasks.

In contrast to higher programming languages, SCEs are characterized by a compact notation of algorithms (matrix as base-type), an interactive (interpretive) way of working and a huge collection of numerical algorithms, symbolical

[1] An outline to common systems is given in [7].

and graphical methods. The open architecture of SCEs permits the existence of specialist extensions (toolboxes) on the same platform. These toolboxes are numerously available and highly integrable.

Hence, if HLA-compliant special-purpose systems can be integrated into the above-mentioned SCEs and if problems do not occur due to system-dependent restrictions of the HLA functionality, the second approach will be acceptable as it is simple and efficient to handle.

In order to have unrestricted access to the full HLA functionality, a direct HLA integration into a SCE must be implemented. Therefore, it is necessary to solve the problem of the SCE/HLA integration before the question raised earlier can be examined.

Until recently, the main difficulty is the lack of both solution and concept for such a direct SCE/HLA integration. This difficulty generally occurs if parallel and distributed processing have to be embedded into an interactive environment.

This article suggests for the first time (as far as the authors know) a SCE/HLA integration concept and describes a prototype implementation in the form of a HLA toolbox for Matlab. The work is based on experiences that the authors gained during the integration of different message passing systems into interactive engineering environments ([7, 10]).

The first two sections concentrate on introducing basic terms and the main concepts on HLA and Matlab necessary for an integration. Possible integration methods and fundamental aspects of an interactive HLA interface are discussed in the next section. Finally, the result of a performance comparision study of a demonstration program on different Matlab/HLA interface implementations is given.

2. The High Level Architecture (HLA)

The HLA standard defines an architecture for the component-based simulation. 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 *federate*. In nontrivial applications, at least two components are used to form a distributed system called *federation*.

The current version 1.3 of the HLA standard ([2]) 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. It defines in the first section a series of services in a language-independent manner that have to be provided by a run-time system (called Runtime Infrastructure; RTI) and by the federates. The second section defines the calling conventions of the services for the programming languages IDL, C++, Ada 95 and Java.

Since the HLA standard defines only the interface between a federate and a RTI, different implementations of such a run-time system are possible (s. fig. 1).

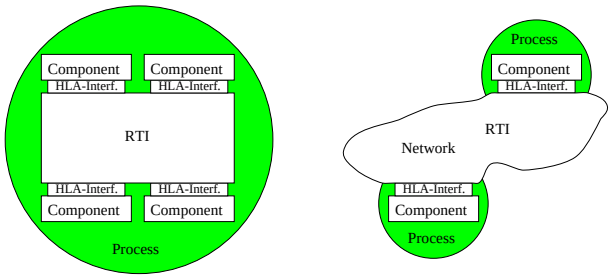

Figure 1. Possible RTI implementations.

A run-time system for IP-based networks has been developed and is freely available from the Defense Modelling and Simulation Office (DMSO). The authors based their research on the DMSO run-time system RTI 1.3.5 ([2]).

3. The Scientific and Technical Computing Environment Matlab

Matlab plays a prominent role among the SCEs. Together with its specialist extensions (toolboxes and subsystems), it provides probably the most extensive collection of sophisticated methods. For example, routines for numerical integration and multidimensional data visualization and animation, contained in the base-system, and toolboxes or subsystems for continuous, discrete and hybrid simulation (ODE Tbx., PDE Tbx., Simulink, Stateflow, GPSS Tbx.; [6, 11, 3]). Even toolboxes that enable parallel and distributed computing are available (DP Tbx., PT Tbx., MultiMatlab Tbx; [7]).

The primary objective of Matlab is *rapid prototyping*. This is very different from conventional programming with high-level languages such as Fortran, C and C++ which are used for the production of run-time and memory-efficient code.

The two key concepts for rapid prototyping in Matlab are:

- Matlab operates *interpretively*. This enables the interactive execution of routines. Also, Matlab routines can be executed directly after coding without compilation. Furthermore, the interactive nature of Matlab allows easy and efficient testing of user-defined routines.

- Matlab provides an integrated *array-oriented programming language* such that it is not necessary to declare data objects explicitly or provide dimensions

for the data objects.[2] Because of this array-oriented nature, scientific and technical problems, which are based on vector or matrix calculations, can be coded and executed very efficiently.

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

User-defined routines can be implemented in Matlab as M-scripts or M-functions using the integrated language or as MEX-functions using C or Fortran.[3]

MEX-functions are the primary Matlab interface for more sophisticated extensions. In particular, MEX-functions are used for the integration of large scale external software into Matlab ([6]). Although this interface was originally designed for linking individual external routines (fig. 2), it provides sufficient functionality to integrate an entire function library into Matlab.

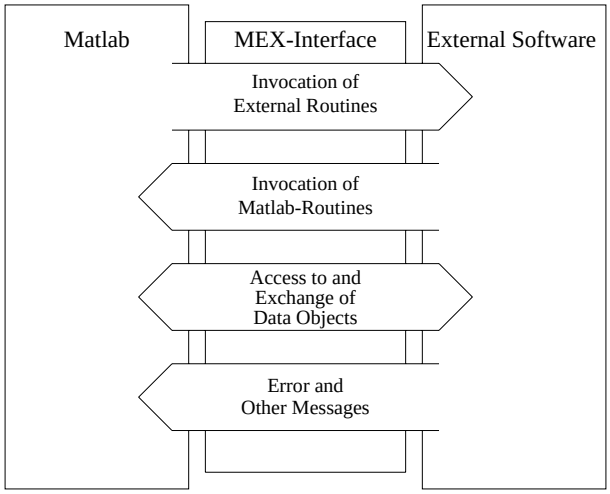

Figure 2. The Matlab MEX-interface.

When binding external libraries into Matlab via the MEX interface, the object code of a MEX-function is loaded dynamically by the Matlab kernel at run-time. It is unloaded once it is not needed.

The principle structure of such a binding for C language is represented in figure 3. Since the library functions operate independently, each routine is linked via a separate MEX-function, which usually carries the same name as the library function. The major task of the MEX-functions is to convert the Matlab calling parameters (arrays) into C function arguments (`m2c(`*iarys*`,&`*iargs*`)`) before a library routine is called and to re-convert the result (`c2m(`*oargs*`,&`*oarys*`)`) after the routine finishes.

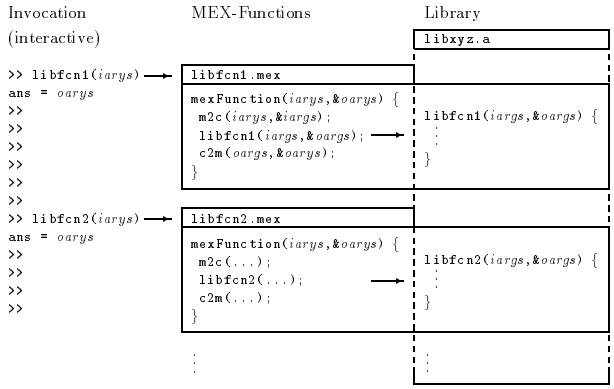

Figure 3. Binding of an external library.

If global and persistent data objects are not used in a library that is linked, no problem related to the dynamic loading and unloading of object code will occur. However, if a function library uses global or persistent data objects, the binding mechanism has to ensure that the entire object code is always loaded as a single module. Furthermore, the dynamic memory management must not unload such a module automatically under any circumstances (*module locking*). Otherwise, status information would be lost. Therefore, the binding of such a library is realized with a single MEX-function as shown in Figure 4. In order to invoke the correct library function, a numerical code has to be selected for the required library function (*fcn*). Since this method leads to an unacceptable calling syntax for an interactive environment, an user interface, consisting of M-functions which provide an appropriate syntax, has to be set up.

4. Matlab/HLA integration

There are two fundamental methods to add HLA functionality to Matlab.

The first method is the implementation of a RTI designed specifically for the integration into Matlab. This seems to be an unfavorable approach as large amount of implementation effort is required. However, from an engineering application perspective, this method has important advantages: A

[2]The memory management handles data objects dynamically and associatively. The necessary identification, type and dimension information are kept implicitly within the data objects.

[3]M for *Matlab* and MEX for *Matlab external*.

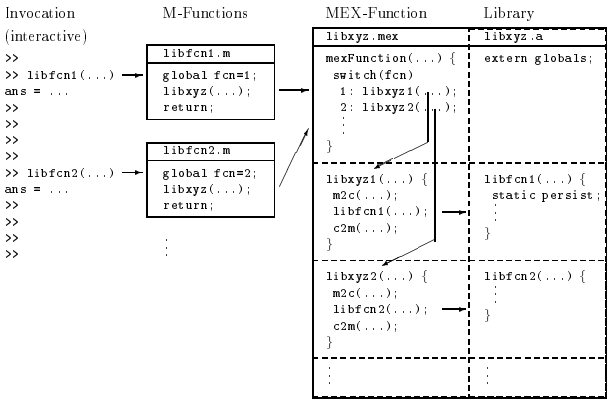

Figure 4. Binding of an external library containing global and persistent data objects.

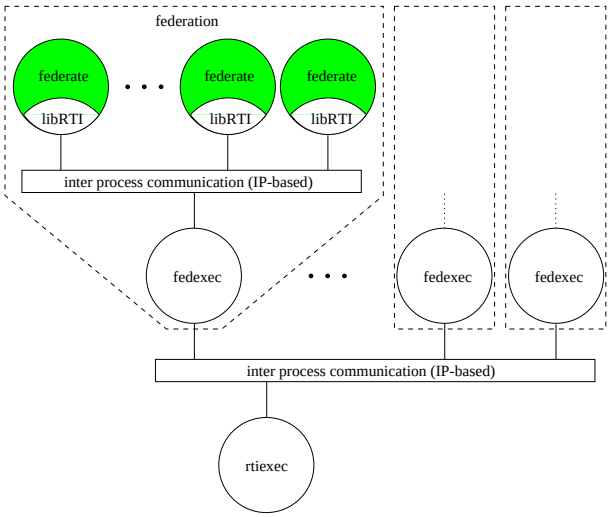

Figure 5. Structure of the DMSO RTI 1.3.5

specially implemented RTI can be scaled according to the actual requirements of an application[4] and it can be ported to any prescribed platform (e.g. non-IP networks, real time operating systems).

This method was not explored by the authors because of the high implementation effort.

The second method is the integration of an existing RTI with Matlab. At the beginning of the authors' investigation, the DMSO RTI was the only run-time system available. At present, this situation has changed due to availability of commercial implementations. Basically, the advantage of this approach is a reduced implementation effort. However, there are more restrictions on the possible platforms that can be used and also the integration into Matlab is more difficult.

4.1. Matlab binding of the DMSO RTI 1.3.5

The components of DMSO RTI 1.3.5 and their relationships are represented in Figure 5.

The run-time system management (*run-time infrastructure executive*) is implemented as independent 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 independent program (fedexec) which controls joining as well as leaving of federates to or from a running federation.

The bidirectional interface between the federates and the RTI is implemented as class library libRTI (s. fig. 6). It provides methods for federates to request services from the RTI (federate-initiated services). These methods are summarized in the concrete class RTIambassador. Furthermore, the libRTI contains the abstract class FederateAmbassador defining prototypes for all methods that a federate has to provide for the RTI (RTI-initiated services).

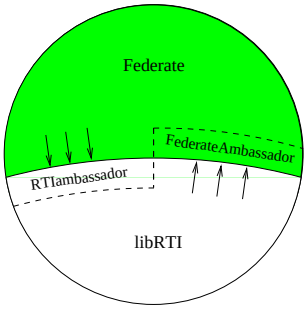

Figure 6. The bidirectional RTI interface.

In order to connect the RTI to Matlab, the class library libRTI has to be linked via the MEX-interface. Since the RTI library holds status information at run-time, the coupling has to be realized with a single MEX-function as shown in figure 4. Actually, the binding of the RTI library is much more complex than the one shown in figure 4, because there is no common language base (MEX-interface: C or FORTRAN; libRTI: IDL, C++, Ada95 or Java) and the interface of the RTI library is bidirectional. In fact, multiple bidirectional mappings (M $\leftrightarrow$ C $\leftrightarrow$ C++) have to be implemented within the MEX-functions of a HLA Toolbox for Matlab.

4.2. An interactive Matlab/HLA interface

With the binding of a RTI library into Matlab, a simple HLA interface, on which Matlab-based HLA federates can

[4] A full HLA-compliant implementation has to provide about 130 services, by which only very few are used depending on the application.

be developed, is possible. However, such an interface is not suitable for interactive usage.

In order to support an interactive way of working as it is usual in Matlab, an HLA interface has to meet the following criterias.

1. Interactively usable designators:

   For the routines of the RTI library, like any other object-oriented programming, fully descriptive designators are used. 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 Matlab/HLA interface.

   Furthermore, the instance designators are unnecessary[5] and should be refrained in a Matlab/HLA interface[6]. In order to create a separate designator space, a short prefix is more suitable.

2. Vectorization and data objects with implicit types:

   The routines of the RTI library perform, like any conventional programs, only scalar operations with elementary data objects as parameters. The repeated execution of an operation with different parameters, often needed in many applications, has to be programmed by additional control structures. Since inputing control structures interactively is time-consuming and fault-prone, the routines of a Matlab/HLA interface should be vectorized as much as possible.

   Moreover, the usage of the Matlab 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 RTI library with more than 80 routines are unnecessary in a Matlab/HLA interface).

3. Interactive callbacks:

   In conventional programming, RTI-initiated HLA services are supplied in form of callback-routines, which can be invoked by the RTI library over the FederateAmbassador. However, in an interactive environment, 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 Matlab/HLA interface.

   If such a default callback is invoked by the RTI, an interpreter within the local variable 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 an user implements specific callback routines, the default routines are overlayed.

4. Selectable exception handling:

   The routines of the RTI library do not handle exceptions by their own, but return the exceptions back to the caller. In an interactive environment, 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 deactivated.

The above criterias were identified during several prototype implementations of a HLA toolbox for Matlab. For more information see [9].

5. Application and efficiency

5.1. Traffic simulation

For testing and demonstration purposes, several Matlab-based federates which simulate and visualize the car and pedestrian traffic at a traffic light crossing, were developed using the HLA Toolbox.

In detail it concerns different time-stepped, event-driven and real-time simulations of the car and pedestrian traffic as well as for the traffic light control and a visualization component.

The commented Matlab sources of all federates can be seen in the WWW ([8]).

5.2. Efficiency comparison

For the investigation of the run-time overhead of Matlab-based federates and C++ federates, the demonstration HelloWorld, which is available as source code in the DMSO RTI distribution, was implemented using different versions of the HLA Toolbox.

The result of the investigation is shown in figure 7. The curves indicate the run-times of 10 successive executions of a federation, consisting of two federates of the same implementation. The following implementations were examined:

C++ (OO): object-oriented HelloWorld implementation with C++

[5] Within a single federation, a federate needs only one RTIambassador and one FederateAmbassador instance.

[6] For test purposes, both function-oriented and object-oriented invocation interfaces are implemented and their performance examined (s. sect. 5.2).

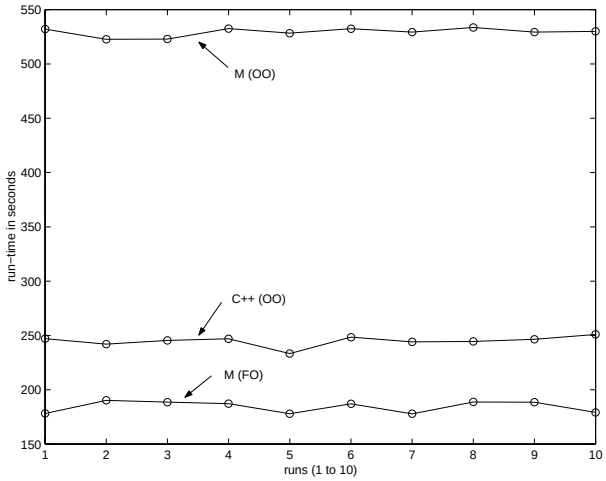

Figure 7. Efficiency comparison.

M (OO): object-oriented HelloWorld implementation using an object-oriented Matlab/HLA interface

M (FO): function-oriented HelloWorld implementation using a function-oriented Matlab/HLA interface

Surprisingly, only the object-oriented Matlab implementation shows an obvious run-time overhead compared to the C++ federate. On the other hand, the function-oriented Matlab implementation takes the shortest time.

The cause of this result has not been examined in detail yet. However, the result suggests that the object-oriented nature of the C++ and M(OO) implementations is responsible for the poor run-time behavior.

6. Conclusions

The HLA Toolbox for Matlab, introduced in this article, represents, as far as the authors are aware, the first realization of a method, with which HLA based distributed simulations can be *interactively* developed, tested and executed within a SCE. Furthermore, the HLA Toolbox for Matlab represents an example of the HLA integration for the entire class of scientific and technical computing environments and could increase thereby – because of the preferential use of SCEs instead of conventional programming languages for many engineering problems – the acceptance of HLA.

However, this will also depend crucially on the suitability of the HLA approach in engineering analysis and design procedures. This is still been examined by the authors.

The next investigations will deal with the inclusion of the Matlab-based special-purpose simulation tools, Simulink (continuous simulation, [6]) and MatlabGPSS (discrete process-oriented simulation, [11, 3]), into HLA-based distributed simulations.

Finally, our thanks go to Ho Shen Shyang who helped us to improve English spelling and expression of the manuscript.

References

[1] A. Bouguezouli and S. Strassburger. Application of HLA based solutions for modeling and control of chemical plants. In *Proc. 12. Symp. Simulationstechnik ASIM98, Zurich, Switzerland*, 1998.

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

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

[4] U. Kein. Distributed simulation for emergency management based on the High Level Architecture. In *Proc. Int. Emergency Management Conf., Washington, D.C.*, 1998.

[5] 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.

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

[7] S. Pawletta. Extension of a scientific and technical computing and visualization system to an environment for developing parallel applications. Phd thesis, University of Rostock, 1998.

[8] S. Pawletta, W. Drewelow, and T. Pawletta. HLA-TB Homepage. http://www-at.e-technik.uni-rostock.de/hla/.

[9] 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. (in preparation).

[10] S. Pawletta, T. Pawletta, A. Westphal, and W. Drewelow. The DP-Toolbox: Distributed and Parallel Processing with Matlab. *at - Automatisierungstechnik*, 47(9):441–443, 1999.

[11] T. Pawletta, S. Pawletta, and W. Drewelow. Process-oriented simulation in interactive SCEs. In *Proc. Simulation and Visualization SimVis98, Magdeburg, Germany*, pages 181–194, SCS Int., Ghent, Belgium, 1998.

[12] T. Schulze, U. Klein, S. Starssburger, K.-C. Ritter, E. Bluemel, and M. Schumann. HLA based distributed simulation models for manufacturing. In *Proc. Simulation and Visualization SimVis98, Magdeburg, Germany*, pages 19–31, SCS European Publishing House, Delft, 1998.

[13] T. Schulze, S. Strassburger, and U. Klein. Migration of HLA into civil domains: Solutions and prototypes for transportation applications. *SIMULATION*, 73(5):296–303, 1999.

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