Integration of Java Federations in HLA Compliant Simulations Governed by CERTI

Andrej Pancik

Faculty of Informatics, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
apancik@mail.muni.cz

Abstract. High Level Architecture constitutes a modern approach to distributed simulation of complex systems. In this paper, we discuss extending CERTI, an open-source Run-Time Infrastructure, with binding to previously unsupported Java language. In addition, we investigate ways of simplifying the process of adding support for new languages by using automated code generation. We test the extension by modificating the OpenRADAR to use the Flight Gear simulator data while utilizing the Virtual Air middleware.

Keywords: M&S, Modeling & Simulation, Distributed simulation, HLA, High Level Architecture, RTI, Run-Time Infrastructure, CERTI, Java.

1 Introduction

Computer simulation has become a natural part of the system design process. It is also commonly used to verificate and validate theories, to study the behavior of complex systems and to analyze possible outcomes of strategies. [21] defines *simulation* as "the process of designing a model of a real or imagined system and conducting experiments with that model."

This generally entails the representation of key characteristics and behaviors with their subsequent analysis and evaluation. Increasing the interest in simulation brings up questions related to interoperability and portability. Several standards are used to maintain these features in simulation applications.

In this paper we focus on a specific part of computer simulation called distributed simulation and practical aspects of adopting it in production environment. We address the issue of extending the existing software platform CERTI, implementing High Level Architecture standard, with binding to previously unsupported Java language. In addition, we explore ways of simplifying the process of adding new language support by using automated code generation.

Following section describes foundations of the modern simulation techniques, it also covers brief introduction to High Level Architecture and contains information about CERTI architecture. Section 3 in turn examines the reasoning behind the Java binding and its architecture design. Results of the testing are presented in section 4 and final thoughts and remarks are located in section 5.

2 Foundations

2.1 Distributed Simulation and its Technologies

Demands of the modern science and business are shifting from simulation of individual and isolated systems towards simulation of highly complex and/or parallel systems often not running in real-time. This shift is reflected also by architectures of simulation frameworks which focus on loosely coupled systems executing units of simulation.

One of the reasons for such trend is that "every time we wish to build a simulation to represent a complex activity, it makes sense to first build smaller simulations to represent individual entities and then to make these smaller simulations interact with each other to create the desired larger simulation while spreading the computational load. It also makes sense that if we build simulations at a later date, then these simulations can interact with other existing simulations as required." [15] Assuring this kind of compatibility between simulations is another important aspect. Standards related to *interoperation*[1] between them emerged.

To sum up; "*distributed simulation* is concerned with the execution of simulations on loosely coupled systems where interactions take much more time [...] and occur less often." [12] They are usually used to simulate system of systems which is highly elaborated and the disadvantages of this complexity are compensated with issuing standards allowing better interoperation and standardized architecture.

2.2 High Level Architecture

Our work is focused on High Level Architecture (HLA) – one of the architectures commonly used nowadays. "The HLA is a software architecture for creating computer models or simulations out of component models or simulations. The HLA has been adopted by the United States Department of Defense (DoD) for use by all its modeling and simulation activities. The HLA is also increasingly finding civilian application." [13]

In other words, it is a general purpose architecture for distributed computer simulation systems. In addition, it provides a flexible framework for creating simulation and interfaces to live systems. It is also used to facilitate the interoperability of different models and units of simulations. Likewise, HLA has important role in reusability of the code implementing it.

There are several versions of HLA standard. To name the most important ones: older HLA version 1.3 [19] and IEEE 1516 [1] with recently approved revision 1516-2010 nicknamed "HLA Evolved".

[1] M&S Interoperability as defined by US Department of Defense – "The ability of a model or simulation to provide services to and accept services from other models and simulations, and to use the services so exchanged to enable them to operate effectively together." [22]

2.3 Components of HLA

There are three main components that comprise HLA:

- Framework and Rules [1]
- Object Model Template (OMT) Specification [3]
- Federate Interface Specification [2]

The *Framework and Rules* is the collection of rules that must be obeyed by a HLA compliant simulation. Rules must be unchanged across all the simulation units as they address the abstract behavior and define the overall architecture. They include manners of the interaction, the design principles and responsibilities of components.

The *Object Model Template (OMT) Specification* describes the structure of the objects transferred between the units of simulation, all interactions managed by the unit and visible outside the unit. [16]

The *Federate Interface Specification* addresses the interface by which the federate is connected to Run-Time Infrastructure (RTI). RTI is the data distribution mechanism in HLA simulation described in greater detail in section 2.5.

2.4 Terminology

To properly define the structure of HLA powered simulation it is important to mention the terminology commonly used in connection with HLA. This nomenclature is also used throughout this paper.

In HLA a *federate* is a single simulation – a basic unit, a component, of the result system.“ A federate may take the form of an aircraft wing or missile or it may take the form of a complete squadron. The level of aggregation of federates is determined by the developer to meet the required need. A federate is also the unit of software reuse.” [15]

There is no limitation on a purpose of the federates. They may be simulation models, data collectors, simulators, autonomous agents or just passive viewers. [23]

If we connect multiple federates via one RTI and use a common OMT the resulting compound is called the *federation*. A session in which a group of federates participate is then called *federation execution*.

During such execution *objects* and *interactions* are transferred between federates. Every object represents a collection of data fields called *attributes* which are used for communication. Analogically every interaction representing the events sent between simulation has data fields called *parameters*.

2.5 Run-Time Infrastructure

HLA itself is a high level standard. It is focused on describing the system from the point of an architect which is different than that of a software engineer. It does not describe the actual implementation details but more the abstract model

of the simulation. There is no network protocol specification, no message format and no encoding details.

Software part of HLA is known as the *Run-Time Infrastructure*. It is a middleware that supports the HLA simulation with necessary services and it provides essential building ground for the software developers. Currently there are several RTIs available both on commercial and non-commercial basis.

Most of the modern RTIs conform to IEEE 1516 [1] and/or HLA 1.3 [19] interface specifications. However; it is always up to the developer to specify implementation details. This loose definition is the reason why the interoperability between different RTIs from different vendors is not guaranteed.

2.6 Dynamic Link Compatible API

The lack of common ground in the RTI implementations caused any potential migration of an application to another RTI to be difficult. APIs were not fully specified and switching RTI vendors also meant recompiling and relinking the code. To address the problem SISO has developed in year 2004 a complementary HLA API specification called Dynamic Link Compatible (DLC) API (1.3 version is available at [19] whereas IEEE 1516 at [18]).

DLC defines several constraints on both RTI and the federate in order to guarantee the possibility of switching RTI without recompiling the whole application. Today, both major interface specifications exist in DLC flavor and thanks to them the developers can build application independent on specific RTI.

2.7 CERTI

Our paper targets CERTI which is an open-source RTI distributed under GNU General Public License 2 with libraries licensed with GNU Lesser General Public License to allow use in proprietary applications [14]. Development started at ONERA Laboratories in 1996 and the first version was available by the end of year 1997. It is under active development since and it has attracted an active community with the transformation to open-source development model in year 2002 [4].

CERTI is partly compliant with both HLA 1.3 and HLA 1516 and currently supports five language bindings specifically C++, Matlab, Fortran 90, Python and Java. [4] The support of the last one is the main contribution of this paper. Supported platforms contains various flavors of Linux, Microsoft Windows, Solaris, FreeBSD and IRIX.

CERTI is designed in a very modular way. It follows the client-server architecture as seen on figure 1. As a result, simulation using CERTI is scalable and can be easily distributed. All the main components communicate by sending messages through sockets on TCP network.

2.8 CERTI Components

LibRTI is a library linked with each federate using DLC interface (for description of DLC see section 2.6). Purpose of LibRTI is to transform HLA service calls into

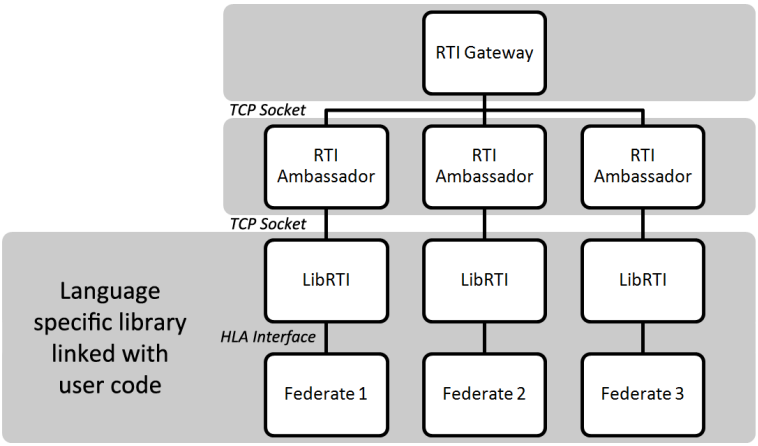

Fig. 1. Architecture of CERTI

messages sent to *RTI Ambassador* (RTIA) and receive responses in form of callbacks. Obviously the LibRTI needs to contain all the necessary implementation of data structures and logic for sending and parsing messages.

Each federate connects to an RTIA. It is a process which exists in one instance for each federate. Its role is to satisfy some requests immediately, while forwarding some requests to *RTI Gateway* (RTIG) [11].

Last main component of CERTI architecture is RTIG. RTIG is responsible for administering the simulation and routing messages between federates. There is only one instance of RTIG in simulation and it is also the central point. This simplifies the implementation of some HLA services such as the creation and destruction of federation execution or maintaining distribution of object classes. Single RTIG may handle several federations but the federation itself must be linked to single RTI.

3 Contribution

In this section we address the necessity of extending CERTI with new language bindings and analyze the process of their creation. Moreover, we propose an architecture of such binding on concrete implementation of Java LibRTI.

To utilize the CERTI framework in the federate one needs to use LibRTI. Implementing it in non-supported language is a non-trivial task. Considering large amount of HLA services or messages sent between LibRTI and RTIA one must take a fair amount of effort.

However, it is very important to support multiple languages as it gives freedom to developers of federates. Every language has advantages and disadvantages and the final choice must consider them with the purpose of the federate

in the scope. What is more, the code on which the federate is based is very often already existent. Developer has to use existing libraries, portions of code or sometimes just architectures and concept that are not present in all languages.

All in all, there is a strong motivation to implement LibRTI in multiple languages. CERTI is programmed mostly in C++ and it has existing bindings to other languages. We decided to explore the possible ways of simplifying the process of adding new language support and to add a new binding to Java.

3.1 Analysis of the Problem

An important part of the new language support development is an analysis of possible routes. There were two main paradigms that had to be analyzed in order to fully evaluate their outcomes.

One obvious option is to implement a simple wrapper around existing C++ version of LibRTI. This can be done with some automation using specialized tools i.e. SWIG [9]. This approach has several advantages. There is variety of supported target languages out of the box after some initial work on specification. On the other hand, wrapping has reported some performance issues in several languages. Other than that, generated code does not comply without some additional work to any specific DLC.

Another way to approach the support of a new language is to recreate a part of the CERTI from the scratch. There are obviously no setbacks related to performance as it runs as fast as possible when properly programmed. In addition, the resulting code is clean and it respects the coding style associated with the language. However, other problems arise from decentralization of the code. There is a unique set of bugs for each new implementation, difficult distribution of changes in network protocols, etc.

During the development process the pros and cons of each of the paradigm were carefully reviewed and we decided to follow the latter one. The challenge was to compensate drawbacks of this approach. We explored the possible ways to cope with this issue and we describe our solution, the message generator, in section 3.4.

3.2 Java SISO DLC 1.3

Another very important task was compliance with standard SISO DLC version 1.3 [19]. It consists of interfaces describing the data structures and RTI ambassador specification. As a result Java application using DLC does not have to be recompiled when switching RTI. During the implementation this behavior was highly valued and breaking it was not an option since it was the main reason of choosing the use of DLC. Therefore, our effort was to keep the binary compatibility with other RTIs.

3.3 Architecture of Java LibRTI

Java LibRTI was designed in a modular way to support maintainability and allow generated code to be easily deployed. Overall architecture is shown in figure 2.

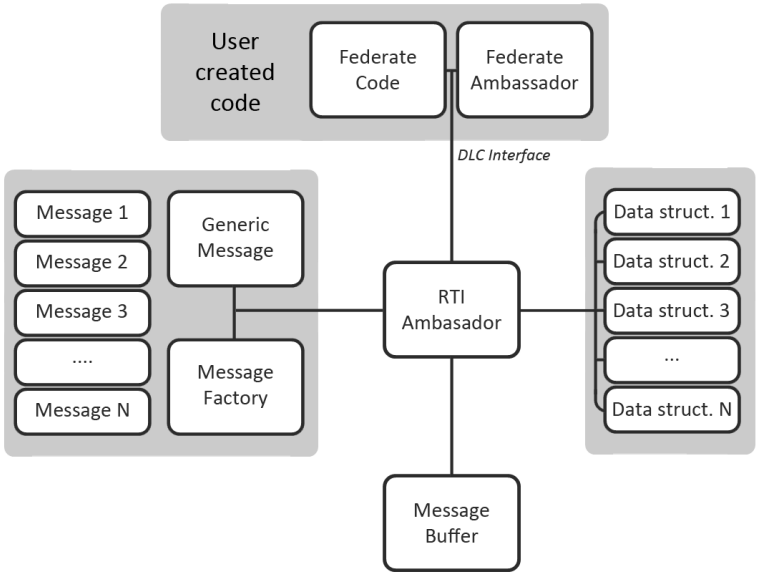

Fig. 2. LibRTI architecture

Federate code communicates with *RTI Ambassador* which exports available functions via HLA DLC 1.3 interface. RTI Ambassador processes the requests and communicates with other parts of CERTI architecture. The process of distribution of these requests and can be observed on figure 3. To put it in the nutshell parameters on DLC methods are converted into messages and then serialized on buffer into the byte stream. After that the stream is sent to RTIA process through the TCP socket. At that point when the response to the service call is received, it is parsed and distributed back to the federate code.

Over time, RTI Ambassador receives callbacks from RTIA process and forwards them to *federate ambassador*. Federate ambassador is a part of federate code and it is mainly responsible for processing callbacks with data.

For more detailed description and implementation aspects of our Java LibRTI see [17].

3.4 Messages Generator

The number of messages transferred between the LibRTI and RTIA process is as high as 144. However, the messages are very similar to each other and they share the common base. They can be described by much simpler rules and a very little amount of data such as the name and the type of a field transferred.

That is the main idea behind the generator: one writes the message specification and let the script do the work and generate the actual code. This approach

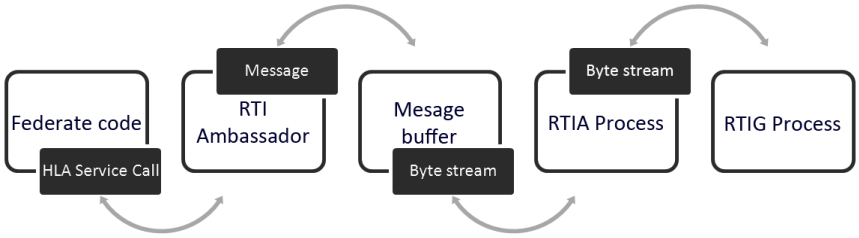

Fig. 3. HLA Service call distribution and transformation diagram

has one major advantage. To support a new language it is sufficient to just write a simple generator and use the common message specification. Moreover, the maintenance of the code is centralized and there is a proper distribution of changes in message specifications so one does not have to change each language separately.

We use the generator to generate the code of all messages in our LibRTI. Overall architecture of the generator is inspired by the Google Protocol Buffers which are "Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data." [6] However, our generator is lightweight and suits our requirements well.

The base and C++ message file generator were written by Eric Noulard in Python. They use the Python PLY module [8] to generate abstract syntax tree (AST)[2] from the specification file. AST is then transformed into valid source code.

4 Testing

To test the functionality of our Java LibRTI we decided to modify the OpenRADAR [7], an open-source application using standard ATC symbolics, to use the Flight Gear simulator data while utilizing the Virtual Air [10] middleware. Simply put the OpenRADAR is virtual radar application used to visualize the situation on Flight Gear Multiplayer Server (FGMS) [5].

Default behavior of OpenRADAR was to connect directly to FGMS. For our purposes we replaced the FGMS data fetcher with our code that retrieved the data from CERTI governed Virtual Air backbone and pushed them to the visualization pipeline. The resulting demo simulation is captured in figure 4.

[2] "The structure of an AST is basically a simplification of the underlying grammar of the programming language, e.g., by generalization or by suppressing chain rules." [20]

Fig. 4. Flight Gear simulator connected to modified OpenRADAR. Multiple black dots next to the white arrow in the left part of image represent the history of plane movements on virtual radar screen with KSFO airport overlay

5 Conclusions and Future Work

In this paper, we present a new approach to extending CERTI architecture with previously unsupported language bindings. The method used supports fast and manageable bug fixing and patch distribution across the whole architecture. Using python based automated code generator makes it possible to introduce new languages more efficiently than before.

The practical outcome of this paper is the publicly available Java LibRTI. Its functionality was demonstrated by the modifications to the virtual radar screen software that made interoperation with flight simulator possible.

While this paper is mainly aimed at High Level Architecture we believe that the outcomes may be used across the modern architectures that are used in distributed simulation nowadays.

We see many possibilities for future work. For example, implementing LibRTI in another language (i. e. .NET platform) would give developers the freedom to choose appropriate language for their applications.

References

1. *IEEE Std 1516-2000: "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules"*. 2000.
2. *IEEE Std 1516.1-2000: "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification"*. 2000.
3. *IEEE Std 1516.2-2000: "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT)"*. 2000.

4. *CERTI Homepage*. 2010. Available online at https://savannah.nongnu.org/projects/certi/.
5. *FlightGear*. 2010. Available online at http://www.flightgear.org/.
6. *Google Protocol Buffers*. 2010. Available online at http://code.google.com/apis/protocolbuffers/.
7. *OpenRADAR*. 2010. Available online at http://mapserver.flightgear.org/git/?p=openradar;a=summary.
8. *PLY (Python Lex-Yacc)*. 2010. Available online at http://www.dabeaz.com/ply.
9. *SWIG*. 2010. Available online at http://www.swig.org/.
10. *Virtual Air*. 2010. Available online at http://virtualair.sourceforge.net/.
11. Benot Brhole and Pierre Siron. *CERTI: Evolutions of the ONERA RTI Prototype*. 2002. Available online at http://breholee.org/files/02F-SIW-018.pdf.
12. Richard M. Fujimoto. *Parallel and distributed simulation systems*. Simulation Interoperability Standards Organization, 2001. Proceedings of the 2001 Winter Simulation Conference.
13. R Dahmann I Kuhl, F Weatherly. *Creating Computer Simulation Systems; An Introduction to the High Level Architecture*. Prentice Hall, 1999.
14. Eric Noulard, Jean-Yves Rousselot, and Pierre Siron. *CERTI, an Open Source RTI, why and how*. 2009. Available online at http://download.savannah.nongnu.org/releases/certi/papers/09S-SIW-015-final.pdf.
15. Department of Defence Canberra. *Distributed Simulation Guide*. Australian Defence Simulation Office, 2004.
16. Department of Defense Defense Modeling and Simulation Office. *RTI 1.3 – Next Generation Programmers Guide Version 3.2*. 2000.
17. Andrej Pancik. *Integration of Java and C++ Federations in M&S HLA Simulations*. 2010. Bachelor's thesis at Masaryk University.
18. Simulation Interoperability Standards Organization Dynamic Link Compatible HLA API Product Development Group (PDG). *Dynamic Link Compatible HLA API Standard for the HLA Interface Specification (IEEE 1516.1 Version)*. Simulation Interoperability Standards Organization, 2004. Available online at http://www.sisostds.org/index.php?tg=fileman&idx=get&id=5&gr=Y&path=SISO+Products%2FSISO+Standards&file=SIS-STD-004.1-2004.zip.
19. Simulation Interoperability Standards Organization Dynamic Link Compatible HLA API Product Development Group (PDG). *Dynamic Link Compatible HLA API Standard for the HLA Interface Specification Version 1.3*. Simulation Interoperability Standards Organization, 2004. Available online at http://www.sisostds.org/index.php?tg=fileman&idx=get&id=5&gr=Y&path=SISO+Products%2FSISO+Standards&file=SISO-STD-004-2004-Final.pdf.
20. Jean-Francois Girard Rainer Koschke. *An Intermediate Representation for Reverse Engineering Analyses, Proc. WCRE, pp. 241-250, IEEE Computer Society*. 1998.
21. Roger D. Smith. *Encyclopedia of Computer Science*. Grove's Dictionaries New York, New York, 2000.
22. David Wilton. *The Interoperability Of Military Simulation Systems In An AUSCANNZUKUS Context*. Australian Defence Science and Technology Organisation, 2001.
23. Ramsey Hage Xiaojun Shen and Nicolas Georganas. *Agent-aided Collaborative Virtual Environments over HLA/RTI*. Multimedia Communication Research Lab (MCRLab) School of Information Technology and Engineering University of Ottawa, 1999.