Applying RESTful Web Services Architecture to HLA Based Simulation Applications

Rukiye CELIK

HAVELSAN A.S. Eskisehir Yolu 7.km 06520, Ankara, Turkey

+905058034036

rsutbas@havelsan.com.tr

Assoc. Prof. Veysi ISLER

Department of Computer Engineering,

Middle East Technical University 06800, Ankara, Turkey

+903122105591

isler@ceng.metu.edu.tr

Keywords:

Distributed Simulation, Service Oriented Architecture, Performance, HLA Evolved, WSDL, SOAP, REST

High Level Architecture (HLA) is a well known, widely accepted distributed simulation standard. HLA focuses on interoperability and reusability of simulation applications but interoperability between RTI (Runtime Infrastructure) implementations of different vendors could not be achieved properly. Also providing reusability of simulation applications is hard because of tight coupling between application and communication layer of simulation code. On the other hand, Service Oriented Architecture (SOA) and Web Services concept is very popular today in context of providing loosely coupled Systems of Systems. One of the major improvements in the most recent HLA standard, IEEE 1516.2010-HLA Evolved, is definition of the web services API based on SOAP (Simple Object Access Protocol). Web services support for HLA-based simulation applications provides more reusability and interoperability. Also by using HLA Web Services API, simulation applications can interoperate with each other over Wide Area Network (WAN) without restricted by firewall issues. Although SOAP based Web Services has the advantage of providing a formal definition language WSDL (Web Service Definition Language), they are known as "Heavy Weight" services. Another way of SOA integration is using RESTful (compliant to Representational State Transfer) Web Services which is widely used for providing cloud services. In this paper we provide a prototype for RESTful Web Services API for HLA.

1. Introduction

High Level Architecture (HLA) is a well known, widely accepted distributed simulation standard [1][2][3]. Developers can describe their simulation systems and interoperate with other simulation systems within a general framework provided by HLA [3]. Three main components of HLA are: HLA Framework and Rules [3], the HLA Interface Specification [4], and the Object Model Template [5]. The HLA standard promises to provide interoperability and reusability of simulation systems via these three components. The implementation of HLA Interface Specification is called as "Runtime Infrastructure (RTI)".

On the other hand, Service Oriented Architecture (SOA) and Web Services concepts are very popular today in context of providing loosely coupled Systems of Systems. Services available in a network such as the web are used by software applications built over an architectural style provided by SOA. SOA promotes loose coupling between software components so to enable reuse. Web services are the preferred standards-based way to realize SOA [9].

RTI implementations for previous HLA standards (HLA 1.3, IEEE 1516) did not provide desired level of interoperability and reusability because of dependence on programming languages, platforms, etc. The most recent HLA standard, IEEE 1516.2010-HLA Evolved, includes

a Web Service API definition for SOA integration. This necessity emerged from couple of reasons. First of all, HLA and Web Services integrate heterogeneous simulations and legacy models, which means that distributed M&S (Modeling and Simulation) is available in a network centric environment. Secondly, though RTIs' performance is satisfying on LAN, same performance cannot be achieved when communication is over WAN/Internet. Meanwhile Web Services has better programming interoperability on WAN/Internet. To allow users of web services communication frameworks to access to the full HLA functionality over Wide Area Networks Web Services support through the new WSDL API is necessary [12]. Also the HLA standard does not support interoperability of different programming languages by itself.

Although SOAP based Web Services has the advantage of providing a formal definition language -WSDL, they are known as "Heavy Weight" services. Designing RESTful Web Services is another way of SOA integration which is widely used for providing cloud services.

In this paper we provide a prototype for RESTful Web Services API for HLA.

The remainder of the paper is organized as follows. Section 2 gives brief information about High Level Architecture (HLA), Web Services and Service Oriented Architecture (SOA), and integration of HLA with SOA. Section 3 describes the related work. Section 4 gives detailed information on motivations and proposed research. Section 5 presents design and implementation of study. Section 6 concludes the paper and gives the plan of future work.

2. Background Information

2.1 High Level Architecture

Developers can describe their simulation systems and interoperate with other simulation systems within a general framework provided by High Level Architecture (HLA). The aim of the HLA is flexibility. HLA comes forward between other distributed simulation about two key issues: promoting interoperability between simulations and reusability of models in different contexts. Three main components of HLA are: HLA Framework and Rules, the HLA Interface Specification, and the Object Model Template. The proper interaction of federates in a federation and responsibilities of federates and federations are defined by ten rules of the HLA Framework and Rules Specification [3]. A documentation standard describing the data used by a particular model is formed by the object model template (OMT) [5] which is a necessary basis for reuse. A generic communications interface that allows simulation models to be connected and coordinated is described by the federate interface specification [4] which addresses interoperability. HLA is an architecture, not software. But to support operations of a federation execution use of a software is essential. The runtime infrastructure (RTI) software is the implementation of the federate interface specification. It provides a set of services used by federates to coordinate operations and data exchange during a runtime execution [3].

With the evolution of technology and expansion of requirements revision of the HLA standards development process is inevitable. The result is the HLA Evolved standards that provide even more features to better meet the needs of modeling and simulation communities such as training, analysis, testing, acquisition and experimentation [12]. Major technical improvements of HLA Evolved standard are[12]:

- Web Services Support
- Modular Federation Object Models (FOMs) and Simulation Object Models (SOMs)
- Fault tolerance support
- Smart update rate reduction
- Dynamic link compatibility

2.2 Web Services and Service Oriented Architecture

A web service supports interoperability between distributed systems over a network. The basic platform of software system provided by Web Services is XML and HTTP. XML provides a language that enables expression of complex messages and functions. This language can be used between different platforms and programming languages. The HTTP protocol is the most used Internet protocol [21].

2.2.1 SOAP Based Web Services

Web Services platform elements are SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery and Integration) and WSDL (Web Services Description Language). SOAP is an XML-based protocol. Applications can use SOAP to exchange information over HTTP. UDDI is a directory service. Companies can register and search for web services from UDDI. WSDL describes interface of web service in a machine-processable format [21].

SOAP applies RPC (Remote Procedure Call) on top of HTTP. The SOAP engine converts RPC into a SOAP message and wraps it as a HTTP request sent via the HTTP server. RPCs have heterogeneous interfaces, and each procedure purpose along with each single parameter

is needed to be learned by programmers. WSDL expresses RPCs in machine-processable format, but does not describe how to use them. Therefore composing and mashing-up services are difficult, and reuse is constrained [14].

2.2.2 RESTful Web Services

Based on the Roy Fielding's theory; the "Representational State Transfer (REST) attempts to codify the architectural style and design constraints that make the Web what it is. REST emphasizes things like separation of concerns and layers, statelessness, and caching, which are common in many distributed architectures because of the benefits they provide. These benefits include interoperability, independent evolution, interception, improved scalability, efficiency, and overall performance."

REST is based on resource oriented architecture. Web service consumer does not send a message to web service provider. Instead of this consumer tries to create/update/get/delete a resource using following HTTP methods:

GET - Requests a specific representation of a resource

PUT - Updates a resource with the supplied representation

DELETE - Deletes the specified resource

POST - Creates a resource with the supplied representation

Following example helps to understand difference between SOAP and RESTful Web Services. To get information of a student SOAP based approach provides a message like *GetStudentInformation()* by using WSDL and SOAP. WSDL is used to define the interface of message in XML format. Then SOAP is used to send message to the server on top of HTTP. The REST approach defines an object which name is *Student*. HTTP GET method is used to get information about *Student* object. This object is uniquely identified and accessed by URI (Uniform Resource Identifier).

2.3 Integrating High Level Architecture with Service Oriented Architecture

RTI Initialization Data (RID) file specifies address and port used by federates and RTI to communicate with each other directly. When an HLA-based simulation application runs in the environment protected by firewall, its communication is blocked by firewall, which even makes the whole simulation fail. Web services may not be blocked by firewall at all due to the use of HTTP. If web services can be introduced in HLA-based simulation application, the limitations mentioned above will be resolved effectively, which will greatly improve its reusability and interoperability [15]. In this context the most recent HLA standard, IEEE 1516.2010-HLA Evolved, includes a Web Service API definition for SOA integration [3].

3. Related Works

According to Morse [17], the goal of the Web Enabled RTI is to enable a simulation to communicate with an HLA RTI through web-based services. In this way multiple federates are able to reside as web services on a Wide Area Network (WAN), and an end-user is able to compose a federation from a browser in the long term. For each federate, there are a client platform and a server platform. Client platform is user of web service. Server platform is a proxy that receives messages from client and convert into HLA calls. Communication between these platforms is supplied by SOAP/BEEP (Blocks Extensible Exchange Protocol) mechanism.

According to Wu [16], with the increase of simulation scale and complexity, distribution of simulator components geographically and distributed simulation on WAN are very essential. In traditional HLA model, there is high coupling between the federate code and LRC (Local RTI Component). Wu [16] proposes two emerging layers to enable decoupling: Application and Messaging layers. At the application layer, HLA service API and Federate Ambassador of Callback Processor replace RTI Ambassador and Federate Ambassador calls. At the messaging layer, messages are received by RTI Ambassador Services module with SOAP and converted into actual RTI Ambassador call. Federate Ambassador Services module transfers callbacks to the federate. To test performance of proposed model, a prototype has been developed and comparison between traditional RTI (pRTI) and Web-enabled RTI (pRTI-WS) has been performed. Results showed that pRTI-WS has worse performance of real-time than pRTI because RTI messaging proxy brings extra overhead, WAN has bandwidth limits and the encoding/decoding process of RTI calls/callbacks to SOAP messages causes more computation time.

According to approach explained in [15], RTI and all necessary web services are deployed at the RTI side. Each federate in client side has a corresponding proxy in RTI side that communicates with RTI on behalf of the federate. An experiment has been performed, and results of WSHLA-based simulation application and traditional HLA-based simulation application are compared. According to results time spent by any version of WSHLA-based simulation application is much more than time spent by traditional HLA-based simulation application because the size of SOAP packet used in

WSHLA is much larger than the packet used in HLA. In addition, marshaling or unmarshaling of the necessary parameters decrease the efficiency of WSHLA when a service is requested or responded.

All investigated related works uses SOAP based web services and uses one proxy per federate that communicates with RTI on behalf of a federate. Results of studies show that Web-enabled HLA has worse performance than traditional HLA because of overhead based on SOAP.

4. Motivations and Proposed Research

Recent related works like [13], [14], [15], [16], and [17] are agreed upon necessity of extending HLA standard with Web Services. New IEEE HLA standard, HLA Evolved, contains Web Service API to meet this requirement already. In this way HLA-based simulation applications will be more interoperable, reusable and interactive over WAN. Applications will be more interoperable because independency between interface and implementation supplied by web services makes developers free for choosing any programming language, any platform or even integrating a legacy system. Applications will be more reusable because web services provide loosely coupled systems. Applications can interoperate over WAN because security restrictions such as firewalls are not applied for HTTP operations.

To design and develop web services, one approach is SOAP-based approach. All investigated related works have preferred SOAP-based Web Services. Another approach is RESTful Web Services. These two approaches have some pros and cons and it should be a design decision to choose which approach is appropriate for the problem, to be able to meet all requirements. Followings are some advantages of REST over SOAP:

- Less overhead hence less bandwidth because of no SOAP envelope to wrap every call in.
- Less duplication because of no need of SOAP envelope to represent operations already represented by HTTP operations GET, PUT, POST, DELETE, etc.
- More standardized because HTTP operations are well understood and operate consistently.

Results of works show that web services enabled HLA based applications have worse performance than traditional HLA based applications. It brings out a tradeoff between more interoperable&reusable systems and systems with better performance. If we achieve improvement at performance of the web service enabled HLA applications, the performance gap between traditional and web service enabled HLA applications will decrease.

In this work we propose using RESTful Web Services for HLA - SOA integration and hope that results will be preferable to SOAP in context of performance. Also there are situations that RESTful Web Services are more suitable than SOAP-based Web Services such as mobile applications. Because of the constraints of mobile devices, “heavy-weight” approaches like SOAP are generally unacceptable. Our work will become a better alternative for this kind of applications.

In this work we developed a prototype for RESTful Web Services API for HLA.

Defining the API and designing web services are nested processes. Designing a RESTful Web Service essentially consists of two steps:

- decide on objects that will be exposed, and decide on representations of objects
- decide on reactions to GET, PUT, POST and DELETE on each of those objects

Development details are covered in following sections.

5. Design and Implementation

5.1 Architectural Overview

High level system architecture for providing RESTful Web Services API for HLA is shown in Figure 1. In our design there is a proxy federate that connects with other federates in LAN and provides communication between web clients and local federates. Modules in proxy federate will be detailed in following subsections. Briefly *HLA Communicator* abstracts HLA RTI services. *REST Connector* connects to one or more federations on behalf of REST clients.

5.1.1 HLA Communicator

HLA Communicator is an HLA RTI 1516.2010 Evolved abstraction framework that abstracts HLA RTI services. HLA Communicator is developed in Java. The framework can support more than one vendor-specific RTI implementations and lets developer switch between RTI implementations without updating underlying source code.

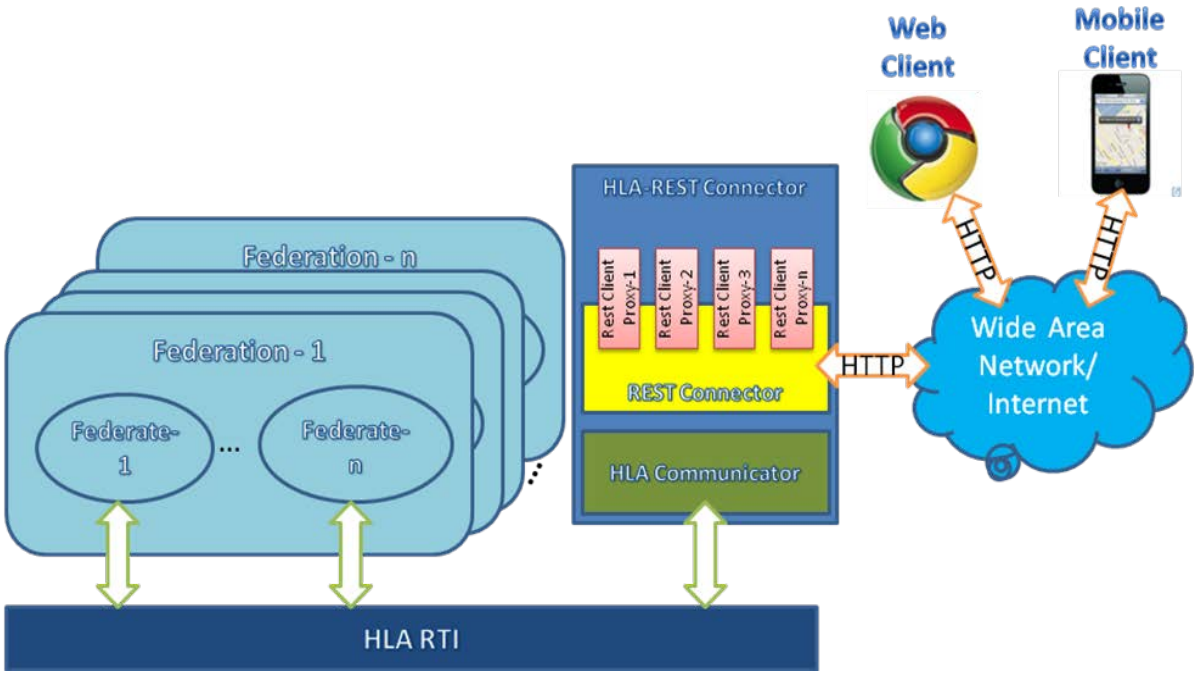

Figure 1 High level system architecture

HLA Communicator has a simplified API. It uses Java Reflection and Annotation APIs to provide a generic framework. In this way, the caller of the API passes names as defined in FOM (Federation Object Model) instead of handles generated by RTI. To achieve this mapping of FOM elements to Java equivalent artifacts is necessary. First of all a POJO (Plain Old Java Object) is created for each FOM Element. Then Java annotations are used for automatic mapping of FOM elements to Java classes and attributes. An illustration is shown in Figure 2.

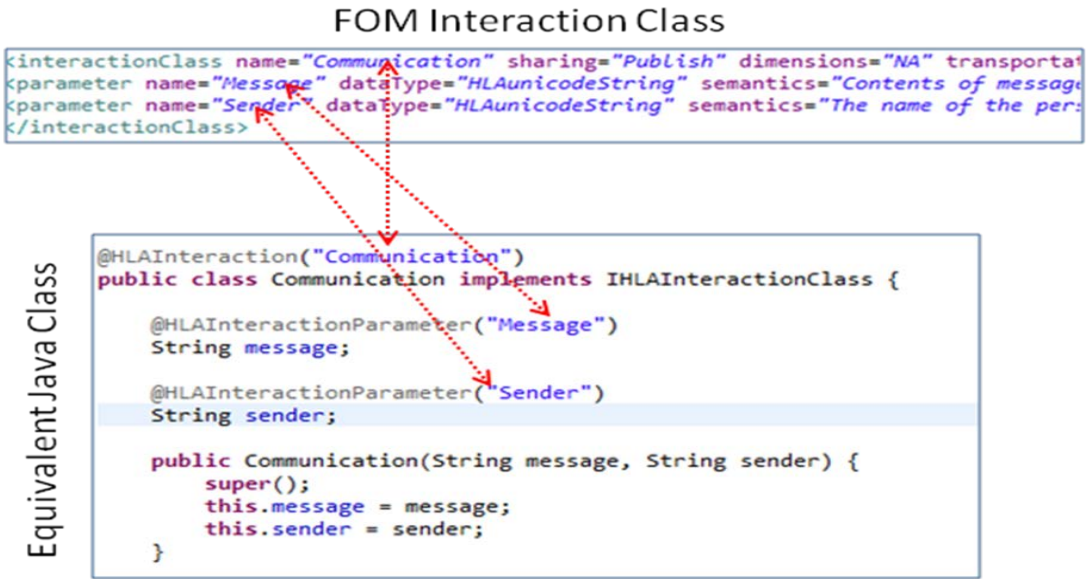

Figure 2 Mapping FOM elements to Java elements

The framework uses new Encoding/Decoding utilities of HLA 1516.2010 Evolved standard for serialization/deserialization of data. Each HLA element equivalent POJO has its own encode/decode methods. An example is shown in Figure 3.

```
public byte[] encode(String fieldName) {
    if(fieldName.equals("message")){
        return Encoder.makeHLAunicodeString(message);
    }else if(fieldName.equals("sender")){
        return Encoder.makeHLAunicodeString(sender);
    }else{
        System.err.println("Unknown field: " + fieldName);
    }
    return null;

}

public void decode(String fieldName, byte[] incomingData){
    if(fieldName.equals("message")){
        message = Encoder.stringFromHLAunicodeString(incomingData);
    }else if(fieldName.equals("sender")){
        sender = Encoder.stringFromHLAunicodeString(incomingData);
    }else{
        System.err.println("Unknown field: " + fieldName);
    }
}
```

Figure 3 Encode/Decode methods of an HLA element

As mentioned before, HLA Communicator has a simple API. There are examples of using HLA Communicator in Figure 4, Figure 5, Figure 6, Figure 7, and Figure 8. These figures show capabilities of HLA Communicator interface like joining into a federation, publishing/subscribing FOM elements, registering and updating an object class, sending interactions, and resigning the federation.

```
communicator = new HLACommunicator(rtiHost, CRC_PORT, "Chat.xml", FEDERATION_NAME);

communicator.init();
```

Figure 4 Joining into a federation

```
// Subscribe and publish interactions
communicator.subscribeInteractionClass("shareddata.chat.interaction.Communication");
communicator.publishInteractionClass("shareddata.chat.interaction.Communication");

// Subscribe and publish objects
communicator.publishObjectClass("shareddata.chat.objectclass.Participant");
communicator.subscribeObjectClass("shareddata.chat.objectclass.Participant");
```

Figure 5 Classes can be published/subscribed with Java name

```
participant = new Participant(_username);

communicator.registerObjectInstance("shareddata.chat.objectclass.Participant", participant);

communicator.updateObjectClass(participant);
```

Figure 6 Object Class POJOs can be registered, updated

```
communicator.sendInteraction(new Communication(message, _username));
```

Figure 7 Interaction Class POJOs can be sent

```
communicator.resign(true);
```

Figure 8 Resigning from the federation

5.1.2 REST Connector

The first version of *REST Connector* is designed and developed over *HLA Communicator* (HLA/RTI Abstraction Framework) explained in previous section. REST Connector has two implicit modules: One is the HLA part that connects to HLA federation and communicates with other federates in the LAN on behalf of Web clients. The other module is responsible for serving to requests from Web clients (REST Clients). *REST Connector* creates one proxy for each REST Client. On HLA part it runs only one federate for each federation to improve performance. On Web server part, there are multiple proxies (one for each REST client) that exchange data via one federate. This one-to-many relationship is shown in Figure 9.

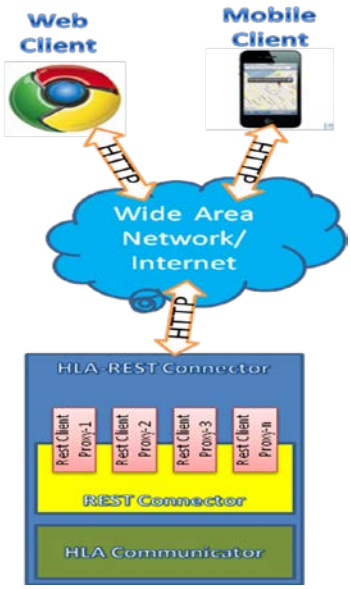

Figure 9 REST Connector

This kind of design (only one *REST Connector* manages proxies for all federations and REST Clients) has some advantages and disadvantages. It promises better performance, because when all/some clients are interested in same data it will be received once by REST Connector and distributed to all clients. But it has a drawback of single point of failure. When the federate code fails, all

clients will suffer from this failure and cannot be served until REST Connector is started over.

Implementation Details of REST Connector

REST Connector uses **Restlet** as RESTful web framework [18]. Restlet is developed on top of Servlet API and enables easier development of RESTful Web Services. Defining and mapping REST resources are simpler with Restlet.

Restlet has an edition for GWT (Google Web Toolkit) [20]. Developers can leverage the Restlet API from within any Web browser, without plugins through the use of GWT [18]. GWT is a widely used platform for rich internet application. Block diagram of relationship between Restlet and GWT is as shown in Figure 10 [18].

We use JSON (Java Script Object Notation) [19] as data exchange format among clients. It is because JSON has better performance than XML, and it is a more compact format than XML. So JSON is more bandwidth friendly. Using JSON to exchange data over WAN is more appropriate compared with the XML because JSON is data-centric, XML is document-centric exchange format.

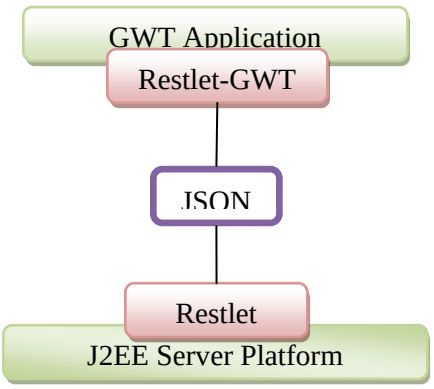

Figure 10 Restlet Edition for GWT

We define resources for Restlet which are related to HTTP URIs (Table 1). Also this is the first part of exposing an HLA REST API. For now Federation Management, Declaration Management and part of Object Management groups of HLA Interface Specification are defined. The following table describes how to access RESTful HLA web service.

Table 1 A Part of RESTful Web Service HLA API

Functionality	HTTP URI	Method	Request Body	Reply Body
Create Federation Join Federation	/hla	POST	Federation Name and Federate Name	Federate ID
Resign Federation Destroy Federation	/hla/{FederationName}/{FederateName}	DELETE	-	-
Publish Interaction Class	/hla/{FederationName}/{FederateName}/interactionpublications	POST	Interaction Class Name	-
Unpublish Interaction Class	/hla/{FederationName}/{FederateName}/interactionpublications/{InteractionClassName}	DELETE	-	-
Subscribe Interaction Class	/hla/{FederationName}/{FederateName}/interactionsubscriptions	POST	Interaction Class Name	-

Unsubscribe Interaction Class	/hla/{FederationName}/{FederateName}/interactionsubscriptions/{InteractionClassName}	DELETE	-	-
Publish Object Class	/hla/{FederationName}/{FederateName}/objectpublications	POST	Object Class Name	-
Unpublish Object Class	/hla/{FederationName}/{FederateName}/objectpublications/{ObjectClassName}	DELETE	-	-
Subscribe Object Class	/hla/{FederationName}/{FederateName}/objectsubscriptions	POST	Object Class Name	-
Unsubscribe Object Class	/hla/{FederationName}/{FederateName}/objectsubscriptions/{ObjectClassName}	DELETE	-	-
Send Interaction	/hla/{FederationName}/{FederateName}/objectpublications/{InteractionClassName}/Write	POST	Sample Data	-
Receive Interaction	/hla/{FederationName}/{FederateName}/interactionsubscriptions/{InteractionClassName}/Read	GET	-	List of read samples
Register Object Instance	/hla/{FederationName}/{FederateName}/objectpublications/{ObjectClassName}/Register	POST	-	Object Instance ID
Discover Object Instance	/hla/{FederationName}/{FederateName}/objectpublications/{ObjectClassName}/Register	GET	Object Instance ID	-
Update Object Instance	TBD	TBD	TBD	TBD
Reflect Object Instance	TBD	TBD	TBD	TBD
Delete Object Instance	/hla/{FederationName}/{FederateName}/objectpublications/{ObjectClassName}/Register	DELETE	-	-
Remove Object Instance	TBD	TBD	TBD	TBD
…	…	…	…	…

6. Conclusion

To make HLA-based simulation applications be more interoperable, more reusable and interactive over WAN, extending HLA standard with Web Services is essential. With the help of web services, we can build loosely-coupled simulation systems. Current HLA standard (1516 Evolved) and other related works use SOAP-based Web Services. SOAP based Web Services are known as heavy-weight web services and has some disadvantages like more overhead (SOAP envelope to wrap every call in), more bandwidth and necessity of learning a new API. Main drawbacks of SOAP Web Services are performance and scalability. Web-enabled HLA applications have worse performance than traditional HLA applications. Therefore our expectation is gaining performance by using RESTful Web Services.

An initial implementation of RESTful HLA Web Service API is finished for now. As future work, firstly missing services will be designed and implemented. After completing the necessary API definition, a RESTful Web Service test and performance benchmark application (a data intense, sample HLA application) will be developed. After that, an equivalent web based client will be developed using HLA Evolved Web Service API (SOAP/WSDL) and performance of the RESTful and the SOAP Web Service applications will be compared. Comparison metrics will be latency, throughput and bandwidth usage of the SOAP based web services and RESTful web services.

7. References

[1] F.Kuhl, R. Weatherly, J.Dahhman: "Creating Computer Simulation Systems: An Introduction to the High Level Architecture", ISBN-10: 0130225118, Prentice Hall, 1999

[2] Mike Lightner, Judith Dahmann: "The High Level Architecture for Simulations" SIMULATION 73: 264-265, doi:10.1177/003754979907300501, November 1999

[3] IEEE, (2010), IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Framework and Rules, 1516-2010

[4] IEEE, (2010), IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Federate Interface Specification, 1516.1-2010

[5] IEEE, (2010), IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Object Model Template (OMT) Specification, 1516.2-2010

[6] IEEE, (2003), "Recommended Practice For HLA Federation Development and Execution Process (FEDEP), IEEE Std 1516.3-2003"

[7] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec: The many faces of publish/subscribe. ACM Comput. Surv. 35, 2, 114-131, June 2003, DOI=10.1145/857076.857078 http://doi.acm.org/10.1145/857076.857078

[8] OMG, UML - Unified Modeling Language, www.uml.org (Last accessed at 01.07.2012)

[9] Service Oriented Architecture, SOA, http://java.sun.com/developer/technicalArticles/WebServices/soa/ (Last accessed at 01.07.2012)

[10] J.R. Noseworthy: "The Test and Training Enabling Architecture (TENA) Supporting the Decentralized Development of Distributed Applications and LVC Simulations", Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, 2008

[11] SISO, www.sisotds.org (Last accessed at 01.07.2012)

[12] Björn Möller, Katherine L Morse, Mike Lightner, Reed Little, Robert Lutz: "HLA Evolved – A Summary of Major Technical Improvements", Fall Simulation Interoperability Workshop, Simulation Interoperability Standards Organization, 2008.

[13] Wei Zhang, Lei Feng, Jiwen Hu and Yabing Zha: "An Approach to Service Provisioning of HLA RTI as Web Services", Asia Simulation Conference, 2008.

[14] Khaldoon Al-Zoubi, Gabriel Wainer: "Using REST Web-Services Architecture for Distributed Simulation", 23rd Workshop on Principles of Advances and Distributed Simulation, No. 114-121, 2009.

[15] Hengye Zhu, Guangyao Li, Liping Zheng: "Introducing Web Services in HLA-Based Simulation Application", Proceedings of the 7th World Congress on Control and Automation, China, 2008.

[16] Zebin Wu, Huizhong Wu, Weiqing Li, Xu Zhang: "Extending Distributed Simulation's Run-Time Infrastructure with Web Services", Proceedings of the IEEE International Conference on Automation and Logistics", 2007.

[17] Katherine L. Morse, David L. Drake, Ryan P.Z. Brunton: “Web Enabling HLA Compliant Simulations to Support Network Centric Applications”, Proceedings of the 2004 Symposium on Command and Control Research and Technology (No. 172), 2004.

[18] http://www.restlet.org/ (Last accessed at 01.07.2012)

[19] http://www.json.org/ (Last accessed at 01.07.2012)

[20] http://code.google.com/webtoolkit/ (Last accessed at 01.07.2012)

[21] http://www.w3schools.com/webservices/ (Last accessed at 01.07.2012)

Author Biographies

RUKIYE CELIK received the B.S. degree and the M.Sc. degree in Department of Computer Engineering from the Hacettepe University, Ankara, Turkey, in 2003 and 2005, respectively. Since 2007, she has been studying Ph.D. in Middle East Technical University (METU), Ankara, Turkey in the area of distributed simulations. Since July 2005, she has been working at HAVELSAN A.S., at Department of Simulation and Training Systems.

DR. VEYSI ISLER received the B.S. degree in Computer Engineering from the Middle East Technical University (METU), Ankara, Turkey, and the M.Sc. degree in Computer Engineering and Information Science from the Bilkent University, Ankara, Turkey, in 1987 and 1989, respectively. He received Ph.D. degree in the Department of Computer Engineering and Information Science at Bilkent University in the area of parallel rendering in 1995. Then, he worked as a research associate at the Computer Graphics and Multimedia Laboratory, Department of Computing, The Hong Kong Polytechnic University. In 1996, he joined the Department of Computer Engineering of METU as an assistant professor. Between 2000-2005, he worked for industry directing a research and development team in the areas of simulation and game development.

Since July 2005, he has been with the Department of Computer Engineering of METU as an associate professor. He is director of Game Technologies master’s program at the same university. He is also director of Modeling and Simulation Research and Development Center of METU since 2006.