A Management Overview of the HLA Evolved Web Service API

Björn Möller
Staffan Löf
Pitch Technologies
Nygatan 35
SE-582 19 Linköping, Sweden
+46 13 13 45 45
bjorn.moller@pitch.se
staffan.lof@pitch.se

Keywords:
Interoperability, HLA Evolved, Web Services, Service Oriented Architecture, WSDL, RTI, WAN

ABSTRACT: *A new API has been added to the HLA IEEE 1516 standard for simulation interoperability. The new API is based on Web Services, an implementation of the Service Oriented Architecture. This paper provides an overview from a strategic and architectural viewpoint targeted at program and project managers as well as technology strategists.*

The capabilities of HLA and Web Services may at first seem to overlap. A closer look reveals some fundamental differences:

- Web Services provides a loosely coupled mechanism for performing coarse-grained services with modest performance over both LAN and WAN. Services are supposed to be stateless and context independent. It is based on technologies that are familiar to most enterprises. A wide range of supporting vendors and software is available.

- HLA provides extremely high performance and scalability for interactions and information exchange in a shared, complex state. It also provides unique capabilities for synchronizing the data exchange between systems using logical time, as opposed to wall clock time.

The Web Service API for HLA combines the best of these two worlds, bringing the above capabilities to a shared approach.

Major benefits of the WSDL API include the support for numerous newer and older languages and operating systems as well as the ease of deployment across wide area networks. Major limitations are performance and scalability.

The major long term benefit of the WSDL API is that it promotes and supports the concept of simulations as readily available services provided within and between enterprises.

1. Introduction

The High Level Architecture (HLA) [1] was initially developed by the US DoD with the purpose of supporting simulation interoperability and reuse. Since the need for simulation interoperability extends outside of the defense community the HLA standard was later taken to IEEE for open, international standardization [2].

At the same time Web Services [3], an implementation of the Service Oriented Architecture (SOA) [4], started to become popular for making business systems interoperable. HLA and Web Services provide interoperability at very different levels and one is not likely to replace the other. On the other hand, they can be amalgamated to provide simulation builders the best of two worlds. Three approaches for doing this are described in this paper and the more powerful one, the HLA Evolved WSDL API, is covered in detail.

2. Interoperability for M&S Systems

Interoperability is today becoming a crucial feature of information systems in almost any domain. While many requirements are similar there are still fundamental differences between for example the business domain and M&S systems. Some of the more important requirements for M&S interoperability are:

1. The need to share a large state, typically consisting of a more static synthetic environment (for example geo data) as well as a more dynamic set of entities (for example vehicles). Due to the amount of data available each system may need to choose how much of the state that it wishes to receive updates for.
2. A high degree of autonomy in the systems. There is usually no "master" triggering all individual events by a call. Instead most

participating systems may freely react to changes in the shared state.

3. The presence of a logical time of the scenario that differs from the real life ("wall-clock") time. This time applies to the data exchange between participating systems but the actual time resolution may vary between systems. Time may be handled using fixed time steps or it may be driven by events.
4. The need for additional synchronization operations, for example for start-up and scenario loading.

Some of these characteristics will be revisited when analyzing how Web Services can be used for M&S.

3. Technical Description of Web Services

Web Service technology is a way for a software component to call other software modules, usually across a wide area network, in order to make use of their services. The calls are performed using the http protocol, originally developed for transferring web pages. The content that is transmitted is formatted using XML.

3.1 Requests and Responses

The structure of a web service call is illustrated by the following simplified example:

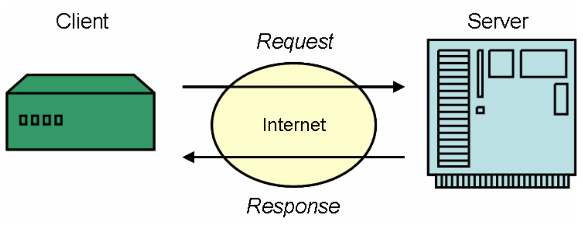

Figure 1: Web Service Request-Response

A simplified example of a web service request to http://webservice.hardwarestore.com from a client at Tom's Repairworks and the response received can look like this:

```
<request webservice="order">
   <customer
      name="Tom's Repairworks"
      account="2634" />
   <item
      partno="56673"
      name="Bolt M3"
      quantity="5000"/>
</request>

---
<response>
  <orderstatus
      cost="56"
      currency="USD"
      deliverydate="010207" />
</response>
```

Figure 2: Simplified Request - Response

Since the http protocol is used, this transaction may easily be performed between networks of different companies or even between continents. Since text formatting and XML are used, this type of call can be expected to work between any combination of platforms.

Note also that the transaction is usually performed on a coarse grained level, typically on the business level. The entire context of the request is provided in the call.

3.2 WSDL and Programming Languages

Using the Web Service Description Language (WSDL) [5], a set of services can be described to support various application domains. There are a number of software frameworks that make it possible to generate all necessary program code to perform the Web Services calls. Code can be generated in a variety of languages (C++, C#, Java J2SE and J2ME, Visual Basic, ADA, even PL/1, Fortran and Cobol) for deployment in a large number of technical environments.

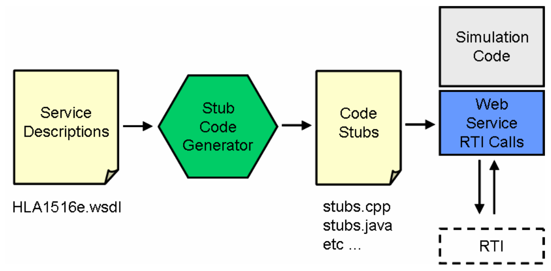

Figure 3: Code Generation from WSDL

There is also Web Service Interoperability organization (WS-I) [6] that promotes interoperability across platforms, operating systems and programming languages.

4. Analysis of Web Services

To fully understand the success of Web Services it is necessary to look at the business context from where it stems. It is heavily promoted by companies like IBM, Microsoft and Sun as a way to make business systems interoperate. In addition to this, it is supported by open source projects like Apache.

4.1 Success Factors

Some of the other success factors behind Web Services include:

Smooth evolution of technologies. Web Services builds on the same principle as a web browser that sends a request from a client to a publicly available server. The request is now initiated by another system instead of a human being. The resulting data is formatted using XML instead of an html-encoded web page. Since web browser applications are usually made publicly available, servers and external access through firewalls more widely accepted, Similar types of access are now more likely to be permitted for remote computer systems as well.

Hardware/software independence. The data exchange is based on the commonly available TCP/IP, http and XML standards and is not associated with any particular vendor, operating system or programming language. This allows everybody to meet on neutral ground.

A technology with a business model. One of the most prominent aspects of Web Services is that it also provides a business model that is easily communicated to both middle and upper management. The idea is of course to provide a business service either within an organization or to other organizations.

4.2 Limitations of Web Services

Some Web Services limitations include:

Poor implementations of the standard. While the speed of the standards development has sometimes been impressive we have found the actual implementations, interoperability between vendors and support for various Web Services constructs to be less impressive. This means for example that a certain development framework may not be able to generate calling stubs for a particular WSDL definition. It also means that the resulting calls may not be compatible between different vendor's clients and servers. During the development of the HLA WSDL standard we unexpectedly had to spend a number of extra man-months trying to figure out how to limit our usage of web services constructs to stay compatible with major frameworks.

Modest performance. The use of the request-response model and XML formatting seriously limits the performance compared to raw socket or multicast based transfer of data with compact representation.

Fragmented information model and limited scalability. If a larger number of systems need to interoperate the point-to-point connection model introduces problems. As connections are often developed ad-hoc the same information may be shared in several ways (for example Fahrenheit in one service and Celsius in another). There is no way to easily hook in new systems that need to listen to the same information without modifying senders. Update mechanisms using multicast or broadcast are generally not available. The above seriously limits interoperability and reuse. This has been recognized by some of the SOA community and approaches like Enterprise Service Bus (ESB) and Contemporary SOA are under development.

Exaggerated expectations on new business models. In practice very few businesses go out to an UDDI directory to look up new business partners. Traditional meetings, negotiations and paperwork usually take place before any actual web services are invoked.

4.3 More about Performance

It is important to understand that any exchange of data will face a trade-off between performance on one hand and modularity/reuse/interoperability through loose coupling on the other hand. The table below shows some examples of a data exchange of a 100 byte data block that is performed using a local C++ method call, an RTI call over a network or using a Web Service API. These are sample, real-life, numbers rather than hard limitations.

Mechanism	**Calls per second**	**Coupling**	**Call Topology**
Local C++ function call	10,000,000	Tight	Local only
RTI exchange using C++/Java API	100,000	Loose	Local or remote
RTI using WS API	3,000	Very loose	Local or remote

Figure 4: Interoperability Method Comparison

What this means in practice is that you can implement a faster and more fine grained interplay between components in the same program than between different processes or hosts. The drawbacks of large, monolithical systems are well known however.

If on the other hand we are striving towards a higher degree of modularity and reuse, we need to look at more loosely coupled systems.

Note also that for the test cases above more or less all CPU cycles were used for the data exchange. Any real system would require a substantial portion of the CPU cycles for its own calculations.

5. Three Approaches for M&S and Web Services

Three approaches for combining M&S and Web Services are explored in this paper, together with an analysis of some pros and cons, especially related to the M&S interoperability requirements described earlier.

5.1 Use Web Services As Is

In this case each pair of systems that needs to exchange data communicates directly.

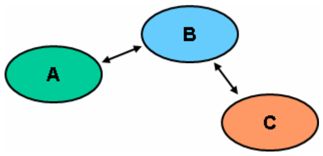

Figure 5: Use Only Web Services

Some advantages are that it is usually quick and easy to get the first couple of systems to communicate. It is possible to take advantage of several Web Services features like support for many operating systems, programming languages and wide-area network (WAN) communications.

Some disadvantages are the fact that there is very limited support for a large shared state as well as time and synchronization handling. This architecture will provide little support for easy expansion (like addition of new systems) and reuse, making it costly in the long run. Certain Web Services limitations, like modest performance, will also be apparent.

5.2 Bridge to an "HLA Island" using Web Services

The second approach is to put crucial M&S systems in an HLA federation and use one federate as a bridge to other systems using Web Services. This approach could for example be used to integrate "What-if" analysis into C4I systems.

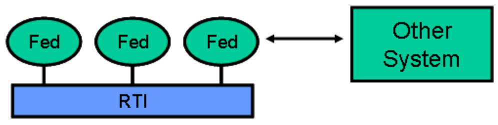

Figure 6: "HLA Island" with Web Services bridge

In this case we will get partial support for M&S requirements like shared state and time handling in the HLA part of the total system. We will also be able to benefit from the Web Service support for many operating systems, programming languages and wide-area network (WAN) communications.

At the same time the Web Services based part of the overall system will suffer limited capabilities as before. It will also run with limited performance.

5.3 Use the HLA Evolved Web Services API

The third approach is to make all systems true HLA federates. To enjoy the benefits from Web Services some federates can use the new HLA Evolved Web Services API.

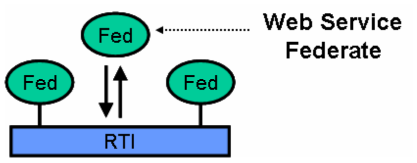

Figure 7: Use the HLA Web Services API

In this approach all systems will enjoy full support for shared state, time handling and synchronization. Federates using the Web Services API can benefit from the support for many operating systems, programming languages and long-haul communications. The only remaining drawback is that these federates will suffer from the performance limitations that are inherent in Web Services.

This paper will now focus on the third approach that is more powerful from an interoperability perspective.

6. Overview of the HLA Web Services API

The HLA Web Services API [13] is described using the WSDL language, which is a precise description of the services, not an actual programming API. Using the WSDL it is possible to generate code that carries out the correct calls. This provides one of the most important benefits of the WSDL API. A wide range of development and deployment platforms are supported. Note that the portion of the federate that communicates with the rest of the federation, the "Local RTI Component", is not necessarily specific to any particular RTI implementation.

To connect to an RTI you must find somewhere to connect to, which is known as the Web Service Provider in the Web Service world. For this purpose a new RTI component is necessary, known as the Web Service Provider RTI Component (WSPRC).

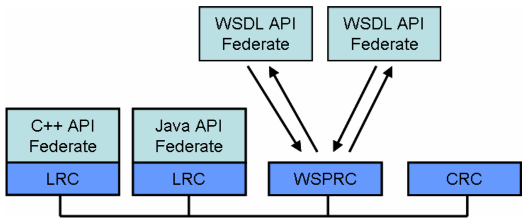

Figure 8: The Web Service Provider RTI Component

The WSPRC provides one or more ports. An example of a port is:

```
http://rti.providercompany.com
```

This port is where you point your federate, just like you would point your web browser to a specific web site/web server. An RTI could also provide multiple WSPRCs, a technique that is known in Web technology as “server farms”.

It is also possible to establish a secure and authenticated connection using the https protocol, in which case the port may be described as:

```
  https://rti.secureprovidercompany.com
```

The full set of HLA Services (such as Join Federation Execution and Update Attribute Values) are available using the HLA WSDL API. During execution it is actually transparent to a federation which API (C++, Java or WSDL) a particular federate uses.

6.1 An example of the WSDL Code

The WSDL Code provides a set of XML Schemas. One example of a HLA service call is provided below.

```
<!-- 4.2 CREATE FEDERATION EXECUTION -->
<wsdl:operation
 name="createFederationExecution">
 <wsdl:input
   name="createFederationExecutionRequest"
   message=
   "wshlae:createFederationExecutionRequest"/>
 <wsdl:output
   name="createFederationExecutionResponse"
   message=
  "wshlae:createFederationExecutionResponse"/>
 <wsdl:fault name="ErrorReadingFDD"
   message="wshlae:ErrorReadingFDD"/>
 <wsdl:fault name="CouldNotOpenFDD"
   message="wshlae:CouldNotOpenFDD"/>
 <wsdl:fault
   name="FederationExecutionAlreadyExist"
  message=
    "wshlae:FederationExecutionAlreadyExist"/>
 <wsdl:fault name="NotConnected"
   message="wshlae:NotConnected"/>
 <wsdl:fault name="RTIInternalError"
   message="wshlae:RTIInternalError"/>
</wsdl:operation>
```

Figure 9: A Sample HLA Operation

6.3 Some Engineering Support

The first major experiment was carried out as part of the XMSF effort using HLA 1.3 as described in an earlier paper [10] Some of our early experiments from mixing HLA and SOA have been described in a paper [11].

As part of the HLA 1516 WSDL development we have collected several types of engineering support:

- The IITSEC 2005 demo
- HLA service group tests
- J2ME tests
- https tests

These tests are summarized below.

A federation that mixes several CGF federates, management federates as well as Google Earth viewer federate was shown at IITSEC 2005. The Google Earth federate connected through a WSDL HLA API. The WSDL federate was running in the DMSO booth and the rest of the federates were running in Pitch’s booth. One particularly interesting experience was that two copies of the federation were actually available: one at Pitch’s booth and one on a server on the Internet in Sweden. While Pitch was setting up their booth the staff in the DMSO booth (who had an Internet connection) chose to connect to the European federation to start the demo. It was completely transparent whether they played in a federation a few hundred feet away or a few thousand miles away.

As part of the WSDL development a number of tests of HLA Service groups using the WSDL API have been performed. These include performing sequences of subsets of the HLA Interface specification, usually between a WSDL federate and a traditional federate. Such test sequences have been executed for:

An additional engineering proof is a small federate that was developed using Java 2 Micro Edition (J2ME) in just two days. The federate was deployed on a Java enabled consumer grade Sony Ericsson cell phone (Z800). This federate was able to connect over the regular GPRS or 3G cell network to a WSPRC and successfully join the federation and send interactions. The only major problem encountered was that the TCP/IP port 80 was the only allowed port (possibly because of security settings).

Yet another test was an approach using https between frameworks from different vendors. The major effort here was to correctly produce and manage the certificates. Once they were in place the https communications worked well.

7. Discussion

The HLA Web Services API provides a number of obvious improvements:

• A substantially larger set of development environments and languages than ever before are now supported by HLA. The very “thin” Local RTI Component can be automatically generated in practically any language and can easily be made vendor independent.

• It is now considerably easier to deploy HLA federates in a wide area network environment. One reason is technical since the http protocol (over TCP/IP) is generally supported. Another reason is the fact that the service can be provided using a web server using a TCP/IP port that an IT security manager is more likely to approve of. The importance of the latter is generally underestimated by people who have little experience in dealing with firewall administrators.

• New communities that today use Web Services to connect systems can be approached with a well-known solution. This has significant potential both for them as well as the HLA/M&S community.

• One, just as important, advantage in the long term is that it promotes the view of and implementation of HLA-based, interoperable simulations provided as services within or between enterprises. This may actually give a deeper meaning to the term "web-based simulations" in the long term.

There are however some limitations that need to be considered:

• Web Service based HLA in no way takes away the effort required in understanding why, how and when to exchange information in a federation. Synchronization operations such as time management and save/restore still require substantial insights. This can be expected to be a problem in particular with people who only have experience in less complex Web Service applications.

• The WSDL API has a lower performance than the C++ and Java API. This can be expected to be a smaller problem on wide area networks than on LANs. A lower message rate may impose effects like lower spatial resolution or even slower time advance rate on the entire federation. One possible remedy is to use the new Smart Update Rate Reduction [12] The performance limitations may be particularly difficult to accept for current users of high-performing federations.

• When calling stubs are generated from the WSDL API in C++ or Java the resulting API is likely to differ from the corresponding standard APIs. The semantics will be the same however.

8. Conclusions

Web Services is an increasingly popular way to make business systems interoperate. While Web Services in no way provides the full set of services to make M&S systems interoperate, it is still important to find ways to make these standards play well together.

One approach is to let an HLA federation serve as a sub-system to another system. A domain-specific or problem-specific service can thus be provided by the federation.

A more general approach is to implement a Web Service API for HLA. This enables web service based systems to participate in a federation and benefit from the full set of HLA services. Such a Web Services API for HLA has been developed as part of the HLA Evolved effort. The API is described using the Web Service Definition Language (WSDL). Based on the WSDL code, it is possible to generate code in different programming languages. It is transparent to a federation whether a specific federate uses the C++, Java or WSDL API.

Major benefits of the WSDL API include the support for numerous newer and older languages as well as the ease of deployment across wide area networks.

The major long term benefit of the WSDL API is that it promotes and supports the concept of simulations as readily available services provided within and between enterprises.

References

[1] DMSO: : "High Level Architecture Version 1.3", DMSO , www.dmso.mil, April 1998

[2] IEEE: "IEEE 1516, High Level Architecture (HLA)", www.ieee.org, March 2001.

[3] Doug Barry: "Web Services and Service Oriented Architectures", www.service-architecture.com

[4] Thomas Erl: "Service-Oriented Architecture, Concepts, Technology and Design", Prentice-Hall, July 2005, ISBN 0-13-185858-0

[5] W3C: "Web Services Description Language (WSDL) 1.1", W3C, URL: www.w3.org/TR/wsdl

[6] W3C: "WS-I Basic Profile", http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

[7] "UDDI", OASIS, URL: www.uddi.org

[8] W3C: "SOAP Version 1.2 Primer" W3C, URL: http://www.w3.org/TR/soap12-part0/

[9] The Apache Software Foundation: "AXIS", http://www.apache.org/axis

[10] Katherine L. Morse, Ryan Brunton and David Drake, "Web Enabling an RTI - an XMSF Profile", Proceedings of the 2003 European Simulation Interoperability Workshop, June 2003.

[11] Björn Möller, Staffan Löf: "Mixing Service Oriented and High Level Architectures in Support of the GIG", Proceedings of the 2005 Spring Simulation Interoperability Workshop, April 2005

[12] Björn Möller, Mikael Karlsson: "Developing well-balanced federations using the HLA Evolved smart update rate reduction." Proceedings of 2005 Fall Simulation Interoperability Workshop, 05F-SIW-87, Simulation Interoperability Standards Organization, September 2005.

[13] Björn Möller, Clarence Dahlin: "A first look at the HLA Evolved Web Service API", Proceedings of the 2006 European Simulation Interoperability Workshop", June 2006

Author Biographies

BJÖRN MÖLLER is the Vice President and co-founder of Pitch, the leading supplier of tools for HLA 1516 and HLA 1.3. He leads the strategic development of Pitch HLA products. He serves on several HLA standards and working groups and has a wide international contact network in simulation interoperability. He has twenty years of experience in high-tech R&D companies with an international profile in areas such as modeling and simulation, artificial intelligence and Web based collaboration. Björn Möller holds an MSc in Computer Science and Technology after studies at Linköping University, Sweden and Imperial College, London.

STAFFAN LÖF is the President, CEO and co-founder of Pitch Technologies (Pitch). He leads the business and strategic development of Pitch. He is active in several international, as well as national, groups for strategic development of distributed simulation. He has over thirty years experience from running high-tech companies and managing R&D type of programs and projects. During 1995-1996 he served as the Attaché of Science and Technology at the Swedish Office of Science and Technology in San Francisco. Staffan Löf holds an MSc in Computer Science from Uppsala University. He has PhD-level education and research experience from Stanford University and SRI International.