A First Look at the HLA Evolved Web Service API

Björn Möller
Clarence Dahlin
Pitch Technologies
Nygatan 35
SE-582 19 Linköping, Sweden
+46 13 13 45 45
bjorn.moller@pitch.se
clarence.dahlin@pitch.se

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

ABSTRACT: *A new API is being added to the IEEE 1516 HLA standard for simulation interoperability. The new API is based on Web Services, an implementation of the Service Oriented Architecture. This paper provides a brief technical introduction and an overview from a strategic and architectural viewpoint.*

A Web Service is a service that can be accessed from another system using an http request and response, similar to fetching a web page. The request and response is described using XML. A service can be described using the Web Service Description Language (WSDL). This is currently the predominant implementation of the Service Oriented Architecture.

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. 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 (scenario). It also provides unique capabilities for synchronizing the data exchange between systems using scenario 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. Simulation system functionality may now be accessed in three ways:

> *- Web Services systems can access the functionality of a federation on a coarse-grained service level using Web Services.*
>
> *- Web Services systems that need a closer interaction, synchronization and more efficient access to the shard scenario can use the HLA Web Service API to fully participate as federates in the federation.*
>
> *- Federates requiring the highest level of performance and scalability can use the C++ or Java API.*

The Web Service API will allow simulation developers to reap the benefits of both the Web Service and the HLA technologies. As the Service Oriented Architecture evolves beyond the current http and request-response implementation these two technologies can be expected to blend, potentially into new concepts and interoperability architectures.

1. Introduction

The High Level Architecture (HLA) [1] was initially developed by the US DoD with the purpose of supporting simulation interoperability. 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. Two techniques for doing this are

described in this paper and the more powerful one, the HLA Evolved WSDL API, is covered in detail.

2. Technical Description of Web Services

Web Service technology is a way for a software module 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.

2.1 Requests and Responses

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

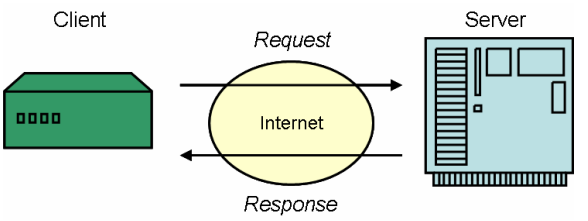

Figure 1: Web Service Request-Response

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

2.2 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.

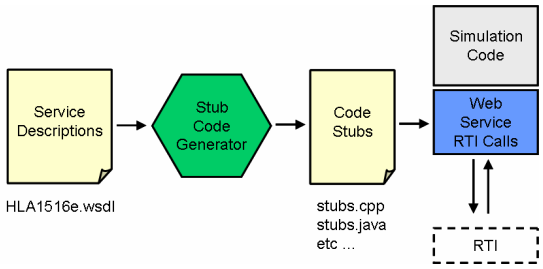

Figure 3: Code Generation from WSDL

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

There are also a number of complementary techniques such as UDDI [7] which provide directory mechanisms to make it possible to discover new services. Other available techniques include point-to-point security.

2.3 Web Services Architectural Styles

Today two major architectural styles are commonly used with Web Services:

• Remote Procedure Call (RPC) style calls. This means that a functional call is performed between two software modules just like if they were implemented in the same program. Software components in the same program usually share a large state, i.e. a large amount of data. The outcome of one call usually depends on previous calls.

• Document style calls. This case can be thought of as a document or a form where you fill in all required information. The form is then processed and a resulting document is returned. The processing service will handle requests, one by one and not assume any shared or preserved context, except for the knowledge and data that it specializes in. This is the currently promoted style for Web Service calls.

Calling HLA using web service can be regarded as somewhere in between these two styles. One of the

most common uses of HLA is actually to share a large common state. In many cases it is not supposed to be well known what service to expect or what data will be produced by another federate when an entity is updated. In this regard a business service can be seen as "backward chaining" (determining only requested variables) and an HLA federation as "forward chaining" (producing new data dynamically based on state change) from a computer science perspective.

2.4 Message Exchange Patterns

In addition, Web Services promote the idea of Message Exchange Patterns (MEPs) which describe a more or less complex sequence of requests and responses. It is possible to fully describe HLA service groups using a MEP notation. As an example the HLA "Send Interaction" service is described using a MEP below.

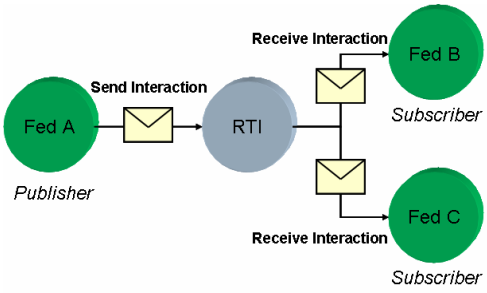

Figure 4: Sample MEP for Send Interaction

3. 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.

3.1 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 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 to other organizations.

3.2 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 now 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 to exchange data with. Traditional meetings, negotiations and paperwork usually take place before any actual web services are invoked.

4. Choosing a Mechanism for Data Exchange

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[1] 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 5: 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. It's all about striking a balance!

5. Two Approaches for M&S and Web Services

Web Services as they stand today provide no dedicated functionality for supporting interoperability between M&S systems. In simpler cases it may be possible to use Web Services for a basic data exchange. If any higher level of M&S interoperability is required there are basically two ways to use Web Services to integrate M&S systems (see Figure 6):

[1] Data for RTI with C++, Java and Web Services measured on 3.6 GHz HP Intel workstation with built-in networking on LAN and Windows XP. A commercial RTI was used but with a prototypical Web Service RTI implementation. Data for the local C++ call frequency was measured on the same configuration.

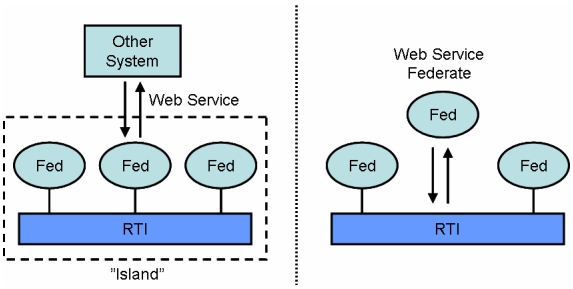

Figure 6: Two Web Service Approaches

• Provide a simulation-based service that in turn is based on an HLA federation. This can be regarded as an M&S subsystem or an "island" that provides a certain service. An example is a C4I system that wants to employ a sub-system for "what-if" analysis.

• A more powerful way, is to allow a system to interoperate natively in a federation through the Web Service API that has been added to the HLA Interface Specification . This is known as the WSDL API for HLA. Such an integration will allow a system to fully share the context with the rest of the federation and to benefit from all M&S oriented services that HLA provides.

This paper will now focus on the later, more powerful approach. The principles used for providing an M&S "island" are well described in current Web Services literature.

6. Overview of the HLA Web Services API

The HLA Web Services API 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. It is worth noticing that the semantics of what an LRC is has thereby shifted somewhat from a "fatter" to a "thinner" approach.

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).

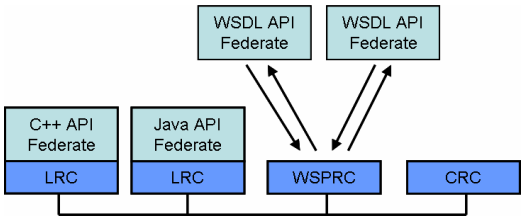

Figure 7: The Web Service Provider RTI Component

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. Different RTI implementations and deployments are likely to differ about how many simultaneous sessions can be supported. 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 Coming to Grips with the WSDL Code

The WSDL Code provides a set of XML Schemas that define the data types that are used. Some examples are provided below.

```
<!-- Many types are just strings -->
<xsd:simpleType
       name="FederationExecutionName">
       <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="OrderType">
   <xsd:restriction base="xsd:string">
         <xsd:enumeration value="Receive"/>
         <xsd:enumeration value="TimeStamp"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="FomDocumentDesignator">
       <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>

<-- Binary values are encoded -->
<xsd:simpleType name="AttributeValue">
   <xsd:restriction base="xsd:base64Binary"/>
</xsd:simpleType>
```

Figure 8: Sample HLA API Data Types

It then defines XML Schema parameter sets for each HLA Service, both requests and responses.

```
<xsd:element name="createFederationExecution">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element
       name="federationExecutionName"
       type="wshlae:FederationExecutionName"/>
   <xsd:element
       name="fomDocumentDesignator"
       type="wshlae:FomDocumentDesignator"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element
   name="createFederationExecutionResponse">
  <xsd:complexType/>
</xsd:element>
```

Figure 9: XML Schemas for Parameter Lists

Based on this a set of requests and responses are defined.

```
<!-- 4.2 CREATE FEDERATION EXECUTION -->
<wsdl:message
  name="createFederationExecutionRequest">
  <wsdl:part name="parameters"
  element="wshlae:createFederationExecution"/>
</wsdl:message>

<wsdl:message
 name="createFederationExecutionResponse">
  <wsdl:part name="parameters"
  element=
  "wshlae:createFederationExecutionResponse"/>
</wsdl:message>
```

Figure 10: Requests and Responses

These are then grouped into the HLA operations which group requests, responses and faults (that are defined in a separate section of the WSDL code).

```
<!-- 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 11: A Sample HLA Operation

The two final sections of the WSDL are:

• Binding. The operations are bound to the Simple Object Access Protocol (SOAP) [8] and http.

• Port. The resulting service is finally bound to a port where it can be accessed. In this case a placeholder is used in the standard. When a WSPRC is deployed it shall be replaced by the actual physical location of the service, as shown in the previous examples.

The HLA WSDL API definitions have been tested with several code generation frameworks. As previously described we found that several frameworks did not behave according to the standard or only implemented parts of it. In some cases we have taken this into account. We have thus avoided structuring the WSDL API into several files/modules to maximize compatibility with a small penalty. In some other cases the impact would be too large and alternate frameworks exist. An example is that Axis 1.3 [9] cannot correctly encode data types described as base64Binary[2].

6.2 Major Differences from the C++ and Java APIs

The most striking difference between the WSDL API and the existing C++ and Java APIs lies in the process model used. The WSDL API only provides mechanisms for one way calls. This means that callbacks have to be "pulled off" the server. They are fetched using the Evoke Multiple Callbacks service. This service returns a list of callbacks as well as an indicator of remaining callbacks. The number of callbacks returned can be limited using a parameter if there is a risk of getting too much data back. The need for this was discovered during practical experimentation.

Another prominent feature is that handles have been replaced by strings in many service calls in order to adapt it to common Web Service style. This eliminates the need for a number of support service calls to translate calls into services. The result is a simpler and clearer programming style as can be seen from the following example:

```
RTI.CreateFederationExecution("myExercise",
          "http://myserver.com/MyFOM.xml")
RTI.JoinFederationExecution("myExercise")
RTI.PublishInteractionClass(
          "InteractionRoot.RadioMessage")
```

Figure 12: Sample WSDL Federate Code

Note that the FDD (FOM) is provided using a URL to make sure that it is available to other federates across a WAN.

When a connection is established to a WSPRC a session is initiated. An http cookie is used to maintain the state, just like a web shop maintains a certain customers' session and shopping basket. Most frameworks generate code to handle this automatically behind the scenes.

Another important thing to keep in mind is that data still needs to be encoded as usual according to the 1516 OMT standard. Before the data is sent over the network it is encoded using base64 encoding which is the common way in Web Services to transmit binary data. In order to facilitate time management a number of standardized logical time representations have been introduced in HLA via the HLA-Evolved activity.

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

[2] A workaround for AXIS 1.3 is to modify the parameter list schemas. If the type of a parameter is "xsd:base64Binary", like for "wshlae:AttributeValue", simply write "xsd:base64Binary" explicitly in the parameter schema instead of using the type.

- 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 DMSO's 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 Europe. While Pitch was setting up their booth the staff in DMSO's 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:

- Create / Join / Resign
- Publish / Subscribe / Sending / Receiving updates and interactions
- Time Advance Request / Time Advance Grant
- Ownership
- DDM
- Selected MOM Object Instances/Attributes and Interactions

An example of a test script for DDM is given below.

```
FOM EXCERPT:

Dimension: Country
Interaction  NewsMessage with parameter
HeadLine and dimension Country

TEST SCRIPT:

Federate A:
Create Region Country1 and set ranges (1..2)
CreateRegionSet
Create Region Country2 and set ranges (2..3)
CreateRegionSet

Federate B:
Create Region and set ranges (1..3).
CreateRegionSet
SubscribeWithRegions

Federate A:
SendInteraction NewsMessage(
  "News in Country 1") Country 1
SendInteraction NewsMessage(
  "News in Country 2") Country 2

Federate B:
//Check that both news are received

Federate B:
SetRangeBounds(1..2)

Federate A:
SendInteraction NewsMessage(
   "News in Country 1") Country 1
SendInteraction NewsMessage(
   "News in Country 2") Country 2

Federate B:
// Check that only the first
// interaction is received

Federate B:
SetRangeBounds(5..6)

Federate A:
SendInteraction NewsMessage(
   "News in Country 1") Country 1
SendInteraction NewsMessage(
   "News in Country 2") Country 2

Federate B:
// Check that nothing is received
```

Figure 13: DDM Test Script Used for WSDL testing

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 one that could be used (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 clear 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. Note that it is relatively easy to implement a "cap" on top of such a generated API to achieve link compatibility with current LRCs. It is unclear how common the need to swap between traditional LRCs and WSDL LRCs is.

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 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.

Acknowledgements

We would like to acknowledge the following organizations and individuals:

The Swedish Defense Materiel Administration provided funding for the early experiments with mixing HLA and SOA Architectures. The other members of this project, Staffan Löf, Björn Löfstrand and Lennart Olsson made valuable contributions during this project.

The US Department of Defense / Defense Modeling and Simulation Office (DMSO) provided funding for an initial HLA WSDL study as well as the contract for developing the full WSDL specification and supporting experimentation.

Katherine Morse and Ryan Brunton have contributed with many valuable experiences from their initial HLA and Web Services work as part of the XMSF project.

During the HLA Evolved effort a Tiger Team was formed and several SISO members have contributed valuable information. The members were Katherine Morse, Reed Little, David Drake, Roger Wuerfel, Ryan Brunton, Ben Watrous, Keith Snively, Annette Wilson, Reggie Cole, Brian Spaulding, Len Granowetter, Miles Fidelman, Clarence Dahlin, Russell Lane and Björn Möller.

During our work we have discussed design questions with several external Web Services experts. This includes questions about limitations and compatibilities in particular Web Services frameworks as well as discussions about how to implement "stateful" Web Services using cookies. We would in particular like to mention Guy Pritchard (Microsoft Distributed Services Team), Dennis Sosnoski, Bob Swart, Kevin Jones as

well as several members of the Apache Axis User mailing list.

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.

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.

CLARENCE DAHLIN is a software developer and consultant at Pitch. He specializes in web technologies, communication and collaborative systems. Key technologies in these areas include http, XML, Web Services and corresponding development and server technologies. Clarence Dahlin studied Computer Science and Engineering at the Royal Institute of Technology (KTH), Stockholm.