Developing Web Centric Federates and Federations using the HLA Evolved Web Services API

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

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

ABSTRACT: *The new Web Services API of HLA Evolved offers several new opportunities for federation developers, both from an architectural and a programming perspective. This paper provides an introduction and a number of recommended practices for federation designers and federate programmers, including some references to FEDEP.*

The first and maybe most important step is to take a web-centric approach to the overall federation design. This includes deciding upon how the federation is to be provided as a service. How will executions and potential participants be managed and how will authentication take place. When are federates allowed to join? Performance characteristics and fault tolerance need to be taken into considerations. It also includes determining reasonable update and interactions rates based on the expected available bandwidth. This in turn may introduce the need for dead-reckoning and similar methods.

The next step is to set up a productive development and debugging environment. A wide array of code generation and WSDL analysis tools are available. An RTI with a Web Service Provider component that follows the HLA WSDL standard is of course also needed.

When developing web services federates, some features may require special attention, including:

- *Connection and authentication*
- *Session handling, time-outs and fault tolerance*
- *Bandwidth management and tuning*
- *Use of handles*
- *Data encoding and time representation*
- *Callback delivery*
- *Support services*
- *Long-haul deployment*

Finally the pros and cons of Web Services based federates versus C++ and Java API federates are discussed.

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. Such a solution is provided through the

Web Services API in HLA Evolved, the upcoming version of HLA IEEE 1516.

This paper provides detailed technical information for designers and developers who intend to build federates using the new Web Services API. The Web Services API is also referred to as the WSDL API since it is described using the Web Services Description Language [5].

1.1 More HLA Web Services API Background

Two earlier papers provide information about the HLA Evolved Web Services API but with a different focus:

"A Management Overview of the HLA Evolved Web Service API" [6] is targeted at managers, system architects, and technology evaluators. It provides a high-level comparison of different interoperability technologies and avoids implementation details.

"A First Look at the HLA Evolved Web Service API" [7] is targeted at project managers and system designers. It contains a description of Web Services and the HLA WSDL API definitions but doesn't provide details about how to develop Web Services based federates.

Depending on your focus you may want to refer to these papers for information that complements this paper.

2. Designing a Web Centric Federation

Systems that use the HLA Web Services API can be compared to a web browser and a Web Server.

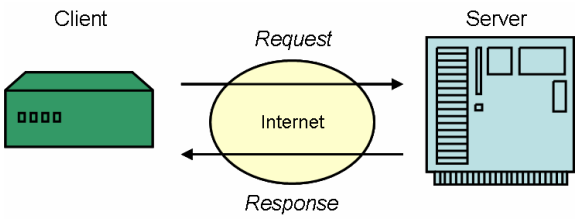

Figure 1: Web Services Topology

The web browser (or federate) can be located anywhere in a network. The web server (or Web Service Provider RTI Component, WSPRC) is located in a well-known place, usually using a DNS name like www.company.com or rti.company.com. The client connects by issuing a request. The server now becomes aware of the client and sends a response. A series of requests and responses forms a session.

The above pattern is significantly different from what we see in many of today's HLA federations where a number of users in the same room start up a well-defined set of simulators.

2.1 Considerations when using the Web Services API

First of all you need to understand why you are using the Web Services API, what features that you intend to take advantage of and what limitations that you potentially may have to handle. To do this it is important to understand the conceptual differences between traditional HLA federations and web services based HLA federations.

The federation is a service. A Web Services based federation may be provided as a service. This means that if someone needs to participate in a particular type of training they may be able to easily point their simulator to the training or analysis session that they want to participate in. The RTI providing the web services connection may provide a range of federations that can be joined.

Long-haul connections are easy. Using Web Services, federations may be made available over wide area networks and across firewalls in a considerably easier way than before. As with web sites there may be no significant difference between connecting to a local or remote federation[1]

Encryption and authentication may be required. Depending on the security zones (internal, external, DMZ, classified/unclassified, etc) that the communication needs to cross greater precaution needs to be taken about encryption and authentication.

More coordination of participants may be required. As participants connect from different places and may be offered several sessions to join more coordination is needed. It is necessary to decide who can join what session and during which parts of the scenario. Many traditional federations require every simulator to start up, connect, load scenarios and start executing in a highly coordinated manner. In web centric federations on the other hand, every federate may be a "late joiner" that needs to synchronize their operational picture before becoming a full member.

Some performance limitations exist. Since a Web Services is based on XML technology it has lower performance than HLA using a highly optimized RTI protocol. It is still possible to exchange hundreds of updates/interactions per second or even a few thousand. For long-distance connections the performance

[1] During the IITSEC 2005 demo of the HLA Web Services API, staff in the DMSO booth switched between connecting to a local federation and a federation on the other side of the Atlantic by typing in a different URL. For this particular federation there was no noticeable difference in functionality or performance.

limitations will usually depend on available bandwidth. For local connections the CPU required for XML processing will usually be limiting. Bear in mind that a dead-reckoned aircraft usually requires 1 – 10 updates per second. It may also be worth sending updates for as many attributes as possible in one update call since the XML payload for each update is large.

Also note that HLA Evolved introduces "smart update rate reduction" that may be used to provide a lower update rate to some federates [9].

Long-haul means high-latency. Compared to Local Area Networks where the latency between two hosts is usually lower than 1 millisecond most wide are networks will have much higher latency, in the range of tens to hundreds of milliseconds. If any type of encryption is applied this may add considerable latency.

2.2 Relation to FEDEP

The following table suggests how the above topics need to be considered during the development and execution of federations, based on the FEDEP standards [10].

Considerations	FEDEP steps
The federation is a service	1,2,3,4,5,6,7
Long-haul connections are easy	1,3,4,5,6
Encryption and authentication may be required	1,3,4,5,6
More coordination of participants may be required	3,4,5,6
Some performance limitations exist	3,4,5,6
Long-haul means high-latency	3,4,5,6

3. Setting up a Development Environment

To set up an environment for development and debugging you need to consider several things:

1) You need an RTI that implements the WSDL API. This API is usually implemented in addition to the C++ and Java API.

2) You need the WSDL definition that is provided as part of the HLA Evolved standard.

3) You need a suitable development environment that supports both the implementation language of your federate and Web Services.

4) You need test federates that can be used for stimulating and/or verifying the federate that you develop. Since C++, Java and Web Services based federates may be mixed freely you may use existing federates that don't use web services. There is no difference from how this is done with older HLA standard versions so this isn't covered in detail in this paper.

3.1 Features of RTIs that support the WSDL API

With HLA Evolved the Web Services Provider RTI Component is introduced.

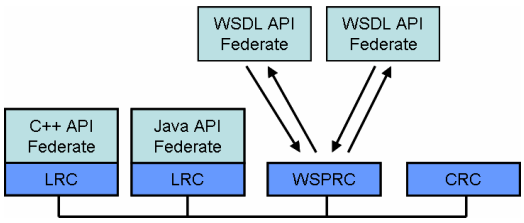

Figure 2: The Web Service Provider RTI Component

Earlier versions had the Local RTI Component (LRC), a local library for each federate as well as the Central RTI Component (CRC) for centralized coordination operations. The WSPRC should be installed on a computer with a well-known address or DNS name and use a well-known TCP-IP port. The default TCP-IP port for Web Services is 80. One or more federates using the Web Services API can then connect to the WSPRC, list available federations[2] and join one of them. Upon the first connection an HTTP session will be established. When the federate calls the disconnect service the session will be disposed of.

The federate needs to call the WSPRC at regular intervals to collect any pending callbacks and (optionally) to make additional calls. If this doesn't happen the WSPRC may at some point in time consider the federate session to be invalid and terminate the session. This is a highly desirable behavior since a crashed federate will never perform a clean disconnect. In a development environment the huge number of abandoned session would soon exhaust the system resources of WSPRC. The session time-out pattern is familiar to most people who have used on-line services on the web.

The WSPRC may optionally provide other functionality such as monitoring of current sessions, error or diagnostic logging and configuration of parameters such as the session time-out.

When you configure the WSPRC you need to make sure that the address and port is available to any federates that need to access it. This may involve opening firewall ports and in some cases setting up "one-to-one" NAT (Network Address Translation) if

[2] HLA Evolved adds services for listing available federation executions for an RTI.

the WSPRC is running in a private address range behind a firewall.

3.2 Web Services Frameworks

Web Services based HLA Federates can be developed in almost any language and on almost any operating systems, assuming that Web Services are supported. Examples of supported programming languages are C++, C#, Java, Visual Basic, FORTRAN, COBOL, ADA, Perl and Delphi.

One major difference between federates using the classic HLA C++ and Java API is that a Web Services federate may be built without any library from any specific RTI supplier. The WSDL definitions can be used to generate code (known as "stubs") that perform RTI calls remotely using a Web Services connection.

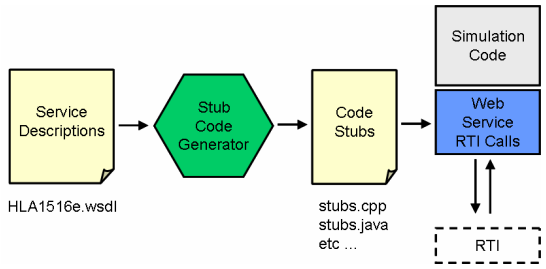

Figure 3: Code Generation from WSDL

While the WSDL is an open standards specification the actual support for code generation for various WSDL constructs in different code generation frameworks may vary. This was a major challenge during the WSDL design. The HLA Evolved WSDL has been tested with several Web Services frameworks, according to the following table:

Framework	Target language
Apache Axis	Java
Microsoft .NET	C#, Visual Basic
IONA	C++
IBM WebSphere	C++, Java
BEA WebLogic	Java

Figure 4: Tested Web Services frameworks

The result of the code generation step is a substantial amount of code (files) that can then be imported into the federate development project in most IDEs (Integrated Development Environments).

Depending on the Web Services framework used the code generation may not always be successful or produce functional code. One example is Microsoft's generation of C++ code where enumerated values in WSDL are not supported. Another example is earlier versions of Axis that could not produce correct code for data that is encoded as byte arrays (for example HLA user defined tags).

You may also need to examine and account for the namespace and package structure of the generated code.

4. Writing Federate Code

The structure of the generated code varies from framework to framework. In this paper we will use Java code generated in the Axis framework since both Axis and Java are generally available to all potential users.

4.1 Code for Joining a federation

The generated code is centered around a service object that can be seen as an RTI ambassador proxy. If Axis is used the federate startup code, including the instantiation of such an object may look like this.

```
URL RtiUrl =
    URL("HTTP://rti.foo.com:80/wsprc");
rti = new WSAmbassadorServiceBindingStub(
    RtiUrl), _service);
rti.connect("");

FomDocumentDesignatorSet foms =
   new FomDocumentDesignatorSet(
    new URI[]{myFomURI});

rti.createFederationExecution(
    "DemoSim",
    foms,
    new URI(),
    "HLAinteger64Time");

String federateDesignator =
    rti.joinFederationExecution(
        "My Boeing 747 federate",
        "mySimulatorType",
        "DemoSim",
        new FomDocumentDesignatorSet());
```

Figure 5: The RTI Ambassador Proxy

The RTI ambassador "proxy" is initialized with a URL pointing to the WSPRC. The connect service (new in HLA Evolved) is then called. It establishes the connecting session to the RTI using RTI connection settings as specified in the settings designator, in this case the default settings of the WSPRC. After this the federation execution is created with the FDD/FOM specified using a URL. Best "web centric" practice is to use an HTTP URL but a file URL can also be used, assuming that the WSPRC (not the federate) can actually access the specified file system. The federation execution can then be joined. Notice that HLA Evolved

introduces a federate name in addition to the federate type in the Join call.

4.2 Session handling and time outs

As describes earlier a session is maintained between the federate and the WSPRC. Remember that the WSPRC may be supporting many sessions at the same time. There are several mechanisms in Web Services for maintaining services and the one used in HLA is the HTTP cookie, which can be found among the HTTP headers in the call. The main advantage of HTTP cookies for sessions is that they are supported by almost all Web Services frameworks and also compatible between different frameworks.

The session (HTTP cookie) is maintained automatically by most frameworks so there is no need to do anything in particular with the cookie. Just remember to resign, disconnect and delete the RTI ambassador proxy before exiting the program.

It is of course possible for one program to create several RTI ambassador proxies and connect as several federates to one federation or to act as a bridge between different federations.

A call may of course time out, just like an attempt to fetch a web page. While this can be seen as an error from the calling federate it doesn't automatically mean that the session has been lost. The connection may be temporarily broken (or bandwidth challenged) and later calls may succeed. How to handle this may need to be decided from federate to federate. This can be regarded as an advantage with the Web Services API since it makes it possible to implement federates that are tolerant to temporary network connection faults for both best-effort and reliable operations.

4.3 Calling RTI services

Calling RTI services in general is no different than with the C++ and Java APIs. Actually it may be a little bit easier since most handles for FOM data have been replaced by the corresponding string. This is an example of registering an object instance:

```
myObjectInstanceDesignator =
  rti.registerObjectInstance
   ("HLAobjectRoot.AirCraft", null);
```

Figure 6: Registering an object instance

4.4 More about handles

As can be seen in the above example the actual FOM data string is used in most cases where a handle is used in the C++ and Java API. This applies to object classes, attributes, interaction classes, parameters, dimensions, transportation types and update rates[3]. For federates and object instances the name is used instead of the corresponding handle. In a few cases, like region designators and retraction handles an opaque byte array is used.

The services for getting the handle from a string are still supported but they will return the same values as supplied when successfully called. If for example an object class doesn't exist in the FOM it will throw an exception as usual.

4.5 Callback delivery

One major difference from other APIs is that callbacks will need to be "pulled" off the WSPRC using the EvokeMultipleCallbacks. Even for small attribute updates the resulting XML Unicode string will be big. The returned data is actually an XML array of callback. A maximum number of callbacks to be returned can be specified in the EvokeMultipleCallbacks call.

```
Callback[] callbacks =
rti.evokeMultipleCallbacks(10).
   getCallbackArray().getCallback();
if (callbacks != null) {
   parseCallbackArray(callbacks);
}

/* Parsing callbacks */

for (callback : callbacks) {
 if (callback.getDiscoverObjectInstance()
    != null) {
      discoverObjectInstance(callback.
           getDiscoverObjectInstance());
    }
 else if (callback.getReflectAttributeValues()
    != null) {
      reflectAttributeValues(callback.
           getReflectAttributeValues());
    }
 else if (callback.getReceiveInteraction()
    != null) {
      receiveInteraction(callback.
           getReceiveInteraction());
    }
    // ... Catch all callbacks ...
  }
```

Figure 7: Processing callbacks

It is a recommended "best practice" to process all pending callbacks before issuing additional service calls. One example of a problem that might otherwise occur is the following: The callback queue contains a "Remove Object Instance" callback. Instead of processing the callback the federate tries to get ownership of an attribute of that object instance. This will, unnecessarily, result in a failure. This situation is not unique to Web Services based federates. It may occur in any federate where callbacks are queued, and even without queuing, since the object instance removal may be in transit on the network.

[3] New in HLA Evolved

4.6 Data encoding and time representation

As described above there is no general need to have any RTI libraries locally available during development or on the host where the federate is deployed. The RTI libraries may come in handy anyway to provide the encoding helpers [9].

HLA Evolved introduces two standardized time representations: 64 bit float and 64 bit integer. When joining, the time libraries are specified as a string in C++ and Java, as well as in Web Services. In order to be able to work with the time values in the Web Services, federate a number of services are provided for converting between the real integer and float values and encoded values, as well as for working with Epsilon values. A list of functions for handling 64 bit float time values is provided here.

- encodeHLAfloat64Time
- decodeHLAfloat64Time
- encodeHLAfloat64Interval
- decodeHLAfloat64Interval
- increaseHLAfloat64TimeByEpsilon
- decreaseHLAfloat64TimeByEpsilon

The integer time type has corresponding operations. Since epsilon equals 1 for integer time types the increase and decrease by epsilon services may be less useful in practice.

A Web Services API based federate can choose to use the above convenience functions. It may alternatively do the corresponding operations locally, with our without RTI-provided time-class implementations. This will result in fewer WSDL calls and higher performance.

4.7 Secure and authenticated connections

It is possible to use secure and authenticated communications using the HTTPS protocol. The following will need to be done:

1) Specify the URL prefixed by HTTPS instead of HTTP.

2) Install server side and possibly client side certificates as specified by the web services frameworks.

There is no need to modify the federate. As part of the development of the WSDL API server side certificates were used.

5 Testing and debugging

The following tools may be useful when testing a Web Services HLA federate:

Most modern RTIs have a **CRC GUI** that can be used for inspecting the state of the Web Services federate as well as the entire federation.

The WSPRC may provide particular debugging functionality as well as **Web Services monitoring interfaces and event logs**. This is useful especially for monitoring the connection status over the Web Services link.

The communication on-the-wire between the WS federate and the WSPRC may be monitored using **SOAP tools** (for example SOAP Scope). This is particularly useful for monitoring the correct formatting and content of the SOAP and XML structure in the Web Services call.

Lower level **network monitoring tools** like TCPtrace may be used to check the raw HTTP call. This is useful for debugging any problems with the HTTP sessions.

In addition, the Web Services frameworks and corresponding tools may provide further debugging and monitoring tools.

The federates themselves may of course be debugged using standard development tools.

6. Discussion

While developing the Web Services API on the detailed, programming level, several incompatibilities and bugs were found in the web services frameworks and the code that they generated. For some WSDL constructs, several frameworks were incompatible. There was no general agreement between the framework vendors on how sessions were to be handled except for using the basic HTTP cookie.

While all of this was somewhat disappointing, the WSDL API as it has been submitted to HLA Evolved works well in most environments. The above information and guidelines apply to most frameworks although the syntax may be slightly different. For some environment, for example J2ME (found in PDAs, cell phones and built-in systems) less XML processing functionality may be available but federates can still be implemented in very short time.

The guidelines in section three may not always result in a federate that always works if the design issues in section two (bandwidth, update rate, etc) haven't been properly handled.

7. Conclusions

Implementing HLA federates that use the HLA Evolved Web Services API does not present any major additional difficulty compared to traditional federate development. A number of guidelines have been provided on how to set up a development environment, how to write the code and how to debug.

Still there are several important things to consider in the federation design. There are several new opportunities to take advantage of, not the least to provide federations as services and to deploy over long distances. At the same time, some limitations, for example in bandwidth and update rate need to be handled.

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] Björn Möller, Staffan Löf: “A Management Overview of the HLA Evolved Web Service API” Proceedings of 2006 Fall Simulation Interoperability Workshop, 06F-SIW-024, Simulation Interoperability Standards Organization, September 2006.

[7] Björn Möller, Clarence Dahlin. A First Look at the HLA Evolved Web Service API. Proceedings of 2005 Euro Simulation Interoperability Workshop, 06E-SIW-061, Simulation Interoperability Standards Organization, June 2006.

[8] 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.

[9] Björn Möller, Mikael Karlsson, Björn Löfstrand: “Reducing integration time and risk with the HLA Evolved encoding helpers.” Proceedings of 2006 Fall Simulation Interoperability Workshop, 06S-SIW-42, Simulation Interoperability Standards Organization, September 2006 Spring.

[10] “Federation Development and Execution Process (FEDEP)”, IEEE 1516.3, www.ieee.org, 2003

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.

MIKAEL KARLSSON is the Chief Architect at Pitch overseeing the world’s first certified HLA IEEE 1516 RTI as well as the first certified commercial RTI for HLA 1.3. He has more than ten years of experience of developing simulation infrastructures based on HLA as well as earlier standards. He also serves on several HLA standards and working groups. He studied Computer Science at Linköping University, Sweden.