WS-Resource Framework and CDDLM

Revision 0.2
Steve Loughran
23 March 2004

Summary

From a Web Services perspective, WS-Resource Framework (WS-RF) is both unusual and hence controversial. Its proposed mechanism for adding state to web services runs contrary to what is widely perceived as the way forward for *Service Oriented Architecture*, the conceptual model that is currently being advocated as the best way to write large-scale distributed systems. However, from a Grid Services perspective, there is little new in the framework, as it is very much like OGSI. By refactoring the OGSI specifications, the framework does make it easier for Web Services and Grid Services to co-exist.

This is important, as it will be some time before an implementation of WS-RF will be available. We can develop a SOAP interface that incrementally moves towards richer WS-RF support.

Things that WS-RF depends on

XML, XML Schema, XML namespaces.

These are foundational and (mostly) non-controversial.

SOAP1.2

This is the latest version of the SOAP specification. It describes two XML representations of data that can be sent between systems. The first, SOAP Section 5 "rpc/encoded" encoding is a pre-XML Schema representation that, while good for describing graphs of serialized objects, is now strongly depreciated by most of the SOAP community. The second encoding format is merely XML Schema, and is known as document/literal, "doc/lit".

SOAP messages comprise an Envelope containing zero or more SOAP Headers, and one SOAP Body, the nominal 'payload' of the messages. There is a well defined protocol for submitting SOAP messages over HTTP, which consists of

1. Open an HTTP connection to the server
2. Make a POST request to the URL of the *endpoint.*
3. Set a SOAPAction HTTP header declaring the 'verb' to act, effectively the name of the message to send (in rpc/enc, it maps more closely to the intended method to invoke).
4. Post the envelope containing optional headers and the body.
5. Send any attachments (assuming Soap with Attachments is used)
6. Get the response or the exception.

SOAP has a good model for representing faults as SOAPFaults; XML data structures that can contain arbitrary data. SOAP implementations often insert elements containing a stack trace and other interesting information.

WSDL1.1

WSDL, the Web Services Description Language, is the equivalent of an IDL for SOAP. It is not mandatory for a SOAP endpoint to have a WSDL description, though it makes it a lot easier to use. WSDL is a very ugly language to write and work with, which makes writing good WSDL hard. Often people cheat, write the implementation classes and let the runtime create the WSDL.

WSDL is extensible –it is possible to insert arbitrary XML in other namespaces into the WSDL to provide extra information to different components. This is the means through which WS-RF information is inserted into the service description.

WS-Addressing

This specification is outside WS-RF, but WS-RF depends on it strongly and it is not part of the basic SOAP specification –indeed, it is a new specification that is not broadly implemented.

WS-Addressing describes how to address multiple SOAP endpoints *inside a single URL*, through the introduction of a new, XML-based URI successor, the *EndPointReference*.

A SOAP endpoint supporting WS-Addressing looks at the <wsa:To> header in a SOAP message, and directs it to the recipient described inside. A SOAP client supporting WS-Addressing has to know to set the header before submitting a message. It may also need to know that the equality test for two endpoints is no longer a case-sensitive comparision of the endpoint URL, their WS-Addressing EndpointReference values need to be compared too.

The WS-ResourceFramework specifications

WS-BaseFaults

This is not published yet.

It is apparently a simple descendant of the fault specifications in OGSI, intended to be the base underlying fault that all Grid Services should raise. As such it is non-controversial and relatively simple for a service producer to support –once the relevant schema is published.

WS-ResourceLifetime

Of all the specifications, this may be the most controversial.

Building on the WS-ResourceProperties specification, the lifetime of a WS-Resource (a WS-Addressed endpoint following this lifecycle pattern), is made finite. This is done first by having a mechanism to destroy a resource; a `<wsrl:DestroyRequest>` message sent to an endpoint telling it to destroy the current state.

For Web Services this is a radical change, and one that may encounter much resistance. For Grid Services, it is nothing new.

WS-ResourceProperties

Of all the specifications, this may be the second most controversial.

This specification adds the notion of Resource Properties to a SOAP endpoint. These are attributes of the endpoint, each identified by its own WS-Addressing EndpointReference, and which can be get and set, individually or in bulk.

Each resource of an endpoint can produce or consume arbitrary XML; the XSD and WSDL of a service can describe what constitutes valid XML for a resource. There does not seem to be any explicit support for binary content, though the PASWA hack for embedding attachments into XML entries would work.

Implementing this specification is a hard undertaking, even with an underlying J2EE EJB application server underneath.

WS-Base Notification

The core specification describes notifications and how they are sent. A central item is the subscription mechanism; subscribers can specify how they want stuff delivered from two options (one looks good for chaining brokers), what preconditions they want on messages to be delivered, and any WS-Policy specified QoS restrictions (like a maximum delivery rate).

All subscriptions have a termination time; this is essential to keep the number of notifications down. It is optional for subscribers to specify this termination time. As with WS-RenewableReferences, Clients can subscribe and unsubscribe to notifications, and may be able to get a look at the current message on a topic. A Subscription Manager interface is used to manage these subscriptions.

There are two ways to deliver messages. The raw delivery simply forwards the SOAP message received by a notification broker. The embedded notification wraps this XML in a new Notify message, that can include extra data, such as topic information. This information could always go in a SOAP Header; the sole rationale for using a new message is that this message permits simultaneous delivery of multiple messages. As raw delivery does not impose any structure on received messages, the message producer can send whatever SOAP message it wants to a subscribing endpoint. The endpoint is, however, described in a wsa:EndpointReference XML structure, so the recipient must be able to understand WS-Addressing headers in the message.

WS-Topics

This is a simple specification that says message channels form heirarchical topics; subscribers can use wildcards when subscribing.

WS-Brokered Notification

This specification describes what service API a notification broker has to offer; what a service that acts as a messaging intermediary must do.

The introduction of intermediaries vastly complicates the messaging infrastructure, from being a simple Peer-to-Peer callback subscription protocol, to a complex message routing framework.

It may be that this broker is the means by which firewalls are pierced; every participating organisation runs one or more externally visible notification brokers, through which messages are routed.

WS-Service Group

This is still unpublished; it is a migration of the OGSI Service Group concept.

A ServiceGroup is a web service that knows about a group of web services which may have some kind of common relationship.

WS-RenewableReferences

This is still unpublished.

It is presumably a rework of the Grid Service Handle to Grid Service Reference mapping; in the WS-RF framework, it is no longer an essential part of the process of finding and binding to a service endpoint.

Analysis

There are some profound features to WS-RF, as it adds many aspects of the a distributed objects pattern to the Service Oriented Architecture (SOA) that was the WS-* protocol stack. This may or may not be a good thing; anyone who has used GT3 will see terminology and code change, but the core concepts will stay. Anyone who has stayed in the Web Service world will find lots that is new. This is going to encounter resistance, especially from companies and organisations who have alternate visions of the future direction of WS-* protocol stack. It means that standardisation, implementation and adoption of the new features is not going to be immediate.

Implementing WS-RF

Implementing some aspects of WS-RF does appear to require a non-trivial amount of effor. Implementing WS-ResourceProperties is one such task –a direct mapping to and from Entity JavaBeans would seem the easiest implementation path, but one that is only viable if a J2EE application server is running on every node.

Notification is even more complex, especially the brokered notification part of the system. Trying to do a reliable store-and-forward messaging system is a serious undertaking.

With all Web Service components, interoperability, *interop*, is a key issue. It is only if a specification is broadly implemented and highly interoperability that it can be used to full advantage. Even today, in 2004, we still have problems marshalling xsd:dateTime references between .NET and Java SOAP stacks [see: Loughran, S., *The Wondrous Curse of Interoperability*, 2003]. If there is still trouble at the SOAP level, consider what issues there must be in the higher up components, those that are newer and more complex.

We also have to consider specification change: the OASIS TC which now has ownership of the specifications will make changes, so any early implementation will have to track the specifications as they move towards becoming a standard. This will extend the time that complete implementation of the framework will be available.

Even if it takes time for WS-RF to stabilise, and longer for GT4 to be ready, because the specifications are somewhat self-contained, some specifications are likely to be implemented before others. The CDDLM service can selectively support those components of WS-RF that are deemed necessary, as and when implementations become available. The convergence of Grid Services and Web Services lets us implement CDDLM as a distributed Web Service, using today's technologies, and take advantage of WS-RF features when we can.

Implications for CDDLM

By factoring out the OGSI specification into separate pieces, a CDDLM service can pick-and-mix those parts of the WS-RF framework that are appopriate. As it turns out, many of them are either not needed, or can be implemented using the classic SOAP patterns.

The lifespan of the CDDLM service is likely to be that of the underlying fabric; it is only if the CDDLM service is undeployed that it goes away. Thus there is no need for the leased resource pattern. Nor is there any need for it to model custom endpoints for any deployed services, as again the lifespan of the endpoints is defined by the lifespan of the services.

The WS-ResourceProperties pattern may be of use when reading and writing properties, but as there is a perfectly valid and uncontroversial SOAP alternative –get and set methods- we do not need to use it.

The unpublished specifications are problematic primarily because they are unpublished, though their OGSI equivalents makes it easier to determine their potential role. Using WS-ServiceGroups to aggregate CDDLM service endpoints does not seem immediately valuable. Having renewal references to CDDLM endpoints could be useful, but in the absence of both a specification and an implemenation, it is hard to demonstrate the value. Fortunately, there are many other discovery mechanisms for Web Services, and WS-RenewableReferences is no longer required.

That leaves three primary specifications, WS-BaseFaults, WS-Addressing and WS-Notification.

Supporting WS- BaseFaults is good citizenship: it will make it easier to work with the service when things go wrong, and should not take much effort (unless there is a surprise awaiting us in the unpublished specification).

WS-Notification would also seem to be good citizenship. If we are going to provide any lifecycle events to anything that cares, we should not be trying to invent our own means of doing so. The sole reason against adopting the specification immediately is:-

(a) There is no implementation,

(b) It depends upon a lot of the other parts of WS-RF, and will take a long time to come into existence

(c) the specification may well evolve a lot in the process too, as people try and bring it together with WS-Eventing from Microsoft.

Because of the timeliness, I would argue in favour of starting a simple model for callbacks, with a long term plan to move to WS-Notification when that becomes possible. Our simple callback model should be designed to be able to segue into WS-Notification with good backwards compatibility. For example, if we export an operation for subscribing to notifications, the operation should have the same message format as a WS-Notification subscription request. Similarly, notification messages should match the WS-BaseNotifications specification. Changing specifications on the road to standardisation may complicate this plan, but the vision is appealing.

WS-Addressing is the odd one out. It may be controversial with the 'one URI' axiom of the W3C, and there is litte apparent need in our CDDLM service for multiple service endpoints under a single URL. Its core value is in forward compatiblity with the WS-Notification standards. Therefore, any callback mechanism should be WS-Addressing aware as soon as possible, while the CDDLM endpoints themselves can remain classic single-URL SOAP endpoints.

Roadmap

Pure SOAP

The CDDLM Service interface is a classic Web Service, described with a WSDL file. Callbacks would have to be via a direct invocation of a public SOAP service at a remote URL-specified location.

If anything goes wrong, SOAPFaults containing custom (XSD-described) elements can be used for faults.

WS- BaseFaults faulting-enhanced SOAP service

The CDDLM Service interface is still a classic Web Service. The WSDL is now enhanced to describe the faults raised by the service; all such faults are extensions of the WS-BaseFaults base faults.

Prerequisites: WS-BaseFaults documentation and schemas

Effort: minimal; helper classes and our own extensions to the base faults

SOAP secured with WS-Security

There are two ways of securing a SOAP call. One is at the transport layer, using HTTPS or simply digest authentication. The other is to secure individual messages in a transport independent manner. WS-Security does the latter.

Prerequisites: WS-Security implementation integrated with the OGSA security model (and CA hierarchy).

WS-Addressing enhanced SOAP

The service now uses WS-Addressing to export endpoints within the service. This is only needed if the endpoint *needs* to export multiple endpoints. Callbacks can also use WS-Addressing EndpointReference addresses for notification, which is where the benefit may be most immediate.

Prerequisites: WS-Addressing support classes (Apache WS-FX?)

Effort: minimal if the WS-Addressing implementation is good.

WS-Notification based callbacks

Any events raised by the system are now raised using WS-Notification.

Prerequisites: A WS-Notification implementation

Effort: unknown.

WS-RenewableReferences support

Assuming each node in the fabric can also act as a SOAP endpoint for the CDDLM deployment infrastructure (this is the expected SmartFrog design), an end user should not have a single wsa:EndpointReference that maps to a single instance of the service –it creates a point of failure in the addressing mechanism, rather than in the infrastructure. Having a mechanism that could map from any one of the endpoints to equivalent endpoints in the rest of the allocated hosts in the fabric would eliminate this point of failure, although of course the mapping service needs to be fault tolerant itself.

The mapping service could be completely independent of the underlying deployment infrastructure. It could, alternatively, be implemented on the allocated resources itself, though hosted on different hosts from the primary endpoints for obvious reasons. Again, deploying onto all allocated nodes would be best, at least when the number of allocated nodes is greater than one.

Prerequisites: WS-RenewableReferences specification and implementation.

Effort: some, but it is independent of the rest of the CDDLM service.

CDDLM Requirements of SOAP/WS-RF?

Are there any special requirements of the underlying communications infrastructure that are not in WS-RF?

From the perspective of the public service, this seems unlikely. Whatever external API we design should be relatively simple.

A more interesting question arises if we ask "could you use WS-* as the protocol stack for implementing the framework itself?" That would depend upon the architecture of the implementation. The fact that WS-RF essentially enables leased distributed-objects over SOAP, should make it possible to mirror the RMI-based design of SmartFrog on a SOAP-based communications layer. I am not, however, convinced that this is the the correct approach to take –certainly not in the short term, but possibly not in the long term either.

Standards List

XML Schema
SOAP1.2
WSDL1.1
PASWA
OGSI1.0
WS-Security
WS-Policy
WS-Addressing
WS-BaseFaults
WS-Resource
WS-ResourceLifetime
WS-ResourceProperties
WS-RenewableReferences

WS-Notification
WS-BaseNotification
WS-BrokeredNotification
WS-PubSub
WS-Topics
WS-ServiceGroup

Added note: Source and Attribution

This was produced as part of the GGF CDDLM WG activity; the document lives canonically at

https://forge.gridforum.org/projects/cddlm-wg/document/WS-RF_and_CDDLM/en/1

Steve Loughran [steve_loughran@hpl.hp.com] indicated in a note of April 29, 2005 that since the time of writing (March 2004), the team has actually done a wsrf version, though it has not yet been implemented.

This document is cited as reference #4 in the *Configuration Description, Deployment, and Lifecycle Management: Component Model Specification* Draft 2005-04-18, edited by Stuart Schaefer; see:

http://xml.coverpages.org/CDDLM-ComponentModel041805.pdf