Using WS-Addressing To Perform Asynchronous Web Service Calls

J.Hayward and A.Phippen

Network Research Group, University of Plymouth, Plymouth, United Kingdom
e-mail: info@network-research-group.org

Abstract

The Simple Object Access Protocol (SOAP) specification does not define an asynchronous message exchange pattern for web services. Business Process Execution Language (BPEL), however, does provide a mechanism for using web services asynchronously, but it relies on hard-coding the end points for the request and the response. This results in tight coupling between the client and the service. WS-Addressing provides a mechanism for defining end points and relating messages with each other. By using the Microsoft Web Service Enhancements the project proposes a mechanism with which web service responses can be routed to an available application using information defined with the WS-Addressing specification. This mechanism loosens the coupling between the client and service and can improve system reliability.

Keywords

Web Services, Asynchronous, WS-Addressing, BPEL

1. Introduction

With the introduction of Business Process Execution Language (BPEL) there has been interest in composing web services into new web services. One example is that of a travel company that uses booking services of airlines, hotels, and car rental companies to provide a new service that enables a consumer to book a complete trip. This new composite service could use BPEL to create a fully automated service where the appropriate services are used and the results compared to provide the consumer with the best option.

Processes that are controlled by computer systems provide a number of benefits. They provide flexibility enabling quick responses to changes in required services. New services can be created to meet new business requirements. Through auditing of completed processes compliance can be assured and costs calculated and controlled. However a majority of processes within companies require human involvement. These human activities stop the automatic process and can take a variable amount of time to complete. For example, in a document editing process an author may take months to write a document. Once the author has finished his task the process needs to continue.

This paper researches mechanisms by which web services can be invoked asynchronously by using open standards and not relying on bespoke interfaces. The WS-Addressing specification provides information that enable routing and messaging information to be included in Simple Object Access Protocol (SOAP) messages. The SOAP standard itself does not specify an asynchronous message exchange pattern. However by including WS-Addressing information within the SOAP message headers asynchronous messaging can be supported. By including the WS-Addressing information in SOAP headers rather than in the published web service interface synchronous web services can ignore the routing information. From the information included in the response from a web service it can be determined if the web service is synchronous or asynchronous.

2. Asynchronous Calls with BPEL

BPEL provides a mechanism for asynchronous calls. The <receive> element following an <invoke> element provides an advertised entry point that the response from the remote web service can use. Juric (2004) describes a typical asynchronous callback as a "*[call to a web service] providing a port type through which the web service invokes the callback operation*". The port type used by the callback as defined by the <receive> element is publicly declared within the BPEL process Web Service Description Language (WSDL) document (Christensen *et al*, 2005). The BPEL asynchronous callback mechanism is illustrated by Figure 1, shown below:

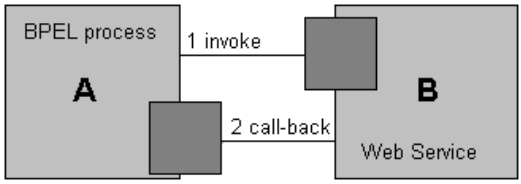

Figure 1: Example BPEL Asynchronous Call (Adapted from figure by Juric (2004))

Juric (2004) explains that in the above diagram the client (A) and web service (B) are BPEL processes. Because the client BPEL process (A) is itself advertised as a web service then a "callback" port type must also be defined within the WSDL. The BPEL process that is the invoked web-service (B) then performs an <invoke> to use the callback to the client (A). This model assumes that all the web services a client will be dealing with will themselves be a BPEL defined process as there is a "partner link" between the two processes. Indeed Andrews *et al* (2003) describe within the BPEL specification that partner links "*represent dependencies between services*". In the airline example supplied by Juric (2004) the callback is hard-coded into the third-party web service (B). Weerawarana *et al*. (2005) give an example whereby the client (A) supplies the partner-link to the web-service (B) which is then used to perform the callback. This reduces the hard-coding, but still relies on the web service (B) being a BPEL process and a tight-coupling exists whereby the web service (B) interface requires a partner-link definition.

The issue still remains of performing a callback from a web service which is either not a BPEL process or contains a hard-coded callback. As with the Weerawarana *et al* (2005) example, the interface of any web service needing to be invoked asynchronously must define a parameter for receiving at least one location of a web service where the response can be sent. This infers that the interface of the web service must be compromised to include callback information. Any client using the client must understand the format of the callback information and provide it along with other parameters the service requires.

Consider a task management system that is used by a BPEL defined processes to assign manual tasks to personnel. The BPEL definition will invoke the task management system to create a new task for a particular person. When the person completes the task within the task management system they acknowledge that the task is complete. The task management system must then perform the callback to the BPEL engine so that the process can continue. In order for the task management system to interact with the BPEL engine in this asynchronous way callback information must be supplied.

There are various categories of information that is required to be included within the callback information. These are:

- Details of where to send the reply
- A unique identifier for the message
- Details of where faults should be sent

Zdun *et al* (2003) details a framework which uses an asynchronous web service call proxy external which provides a simple API to client code. The approach of using a proxy is similar to that provided by the Microsoft .NET framework, where it is the proxy that waits for a response from the service and then performs the provided call back to the client code (Microsoft, 2003). Zdun *et al* (2003) use the proxy to either poll the service to determine if a result is available or it waits for the response from the service and then performs the required callback to the client. With the proxy approach there are a number of issues that arise when the length of time for the reply from the service could be more than a few minutes, especially when human interaction is involved. These issues include:

- Each poll of the server generates at least one network message and response. With many instances running this could unnecessarily flood the network and downgrade performance.
- It is unclear what occurs if the client crashes. Does the proxy continue, or does it crash too? If the proxy stays up what happens to the callback go?

3. Defining Callbacks with WS-Addressing

The Web Service Addressing (WS-Addressing) specification provides a framework for supplying information that would be required for a callback mechanism between remote services. WS-Addressing provides transport-neutral mechanisms to address

Web services (Gidgin *et al*, 2005). This provides the means to identify a web service endpoint and a way to use these in SOAP messages for the exchange of messages between Web Service providers and requesters (Weerawarana *et al*, 2005).

WS-Addressing contains the headers *ReplyTo* and *FaultTo* which define locations where response messages are to be sent (Weerawarana *et al*, 2005). Additionally a *MessageID* provides a unique identifier for the message so that a message can be related to a specific request by the sender. The *Source* header defines where the request came from. When callback information is supplied to the service the WS-Addressing headers can be supplied in the <header> element of the SOAP message.

Below is an example of a request SOAP message and the response that utilise the WS-Addressing specification (Adapted from example by Weerawarana *et al*, 2005).

WS-Addressing Request Example

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"

xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <S:Header>
    <wsa:Source>
      <wsa:Address>http://source.com/sender</wsa:Address>
    </wsa:Source>
    <wsa:MessageID>guid:7hf8dhycd-f8djdi9-difcjdkfd
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://reply-machine.com/replyreceiver
      </wsa:Address>
    </wsa:ReplyTo>
  </S:Header>
  <S:Body>
    <!--- BODY CONTENTS --->
  </S:Body>
</S:Envelope>
```

WS-Addressing Response Example

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"

xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <S:Header>
    <wsa:RelatesTo>guid:7hf8dhycd-f8djdi9-difcjdkfd
    </wsa:RelatesTo>
  </S:Header>
  <S:Body>
    <!--- RESPONSE CONTENTS --->
  </S:Body>
</S:Envelope>
```

The WS-Addressing specification states that the *RelatesTo* header contains the ID passed to the service with the *MessageID* header in the request (Weerwarana, 2005). It is header that enables the application receiving the response to match the response to the process that generated the original request. By using WS-Addressing in the SOAP headers the response can be routed to an appropriate location. However, the SOAP specification (Box *et al*, 2000) does not contain an inherent asynchronous messaging exchange pattern. The web service still needs to be written to specifically read the WS-Addressing headers in order to provide asynchronous calling.

As described earlier, the sender of the request may not be available to receive the asynchronous result. This is especially true where the asynchronous call is to allow human interaction which could take months to complete. Therefore the callback address supplied in the WS-Addressing *ReplyTo* header may be an un-reliable one. One solution is for the *ReplyTo* address to point to a host that acts as a mediator or broker. The broker will receive the response and forward it onto either the source of the request or, if it is unavailable, any available host. The source of the request is supplied to Web Service as the *Source* header and so needs to be included in the SOAP message sent to the broker. Figure 2 illustrates this sequence of events:

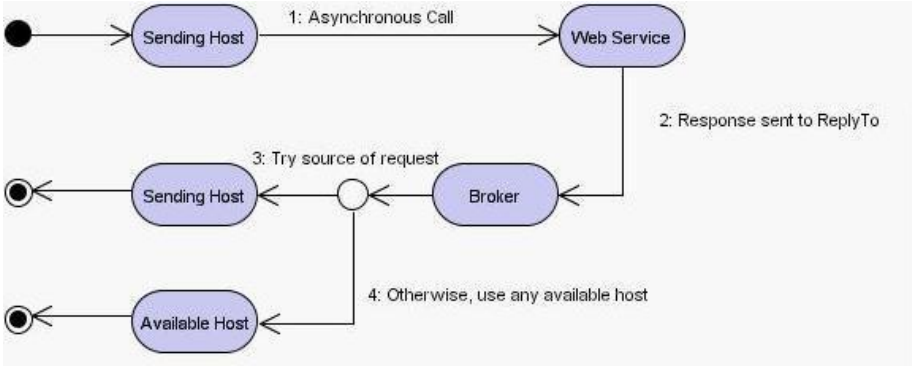

Figure 2: Using a broker to route asynchronous call responses.

In this architecture the Broker needs to contain a level of intelligence. It must interact with a directory, such as a Universal Description Discovery and Integration (UDDI) server where hosts capable of processing the response are advertised, and select an applicable host.

4. Testing Using WS-Addressing for Asynchronous Messaging

Both Microsoft .NET and Apache Axis 2 support asynchronous calls to published web services that are not specifically designed to be accessed in synchronous manner. The Visual Studio .NET (Microsoft, 2003) help topic "*Asynchronous Design Pattern Overview"* states that "*One of the innovations provided by the asynchronous pattern is that the caller decides whether a particular call should be asynchronous.*" This is confirmed for invoking web services in the help topic "*Communicating with XML Web Services Asynchronously*" which states "*note that an XML Web service does not have to be specifically written to handle asynchronous*

requests to be called asynchronously". This feature is only achievable through using the Microsoft .NET client API and not through generic SOAP calls. Also this API only allows the callback to the calling object instance. The callback cannot be routed to a different host or run-time instance. The Apache Axis 2 asynchronous support works in the similar way, in that it is achievable through a client API using a non HTTP transport (Zdun *et al* 2003).

The Microsoft Web Service Enhancements (WSE) (Microsoft, 2005) provides additional classes to the .NET framework providing support for the additional web service specifications such as WS-Security, WS-Trust, WS-Policy and WS-Addressing. To use the WSE to test the support of WS-Addressing in providing the callback information a simple web service was created using Microsoft Visual Studio .NET 2003 and deployed on host running Microsoft Advanced Server 2003. The WSE provides classes that enable the SOAP headers to be accessed from the deployed web service. This enables the service to access the values of the WS-Addressing headers.

The first step was to install the WSE on the Windows Advance Server 2003 computer where the web service is hosted. Once the WSE was installed the supplied Configuration Editor was used to enable the web service project to operate with the Web Service Extensions. The code for the Web Service project was then edited to include the *Microsoft.Web.Services2* namespace. The Microsoft.Web.Services2 namespace contains all the classes required to access the SOAP headers such as *ReplyTo* and *MessageID*. Within the web service the following code retrieves the *SoapContext* object and gets the string containing the supplied *ReplyTo* address:

```
SoapContext ctxt = RequestSoapContext.Current;
string replyTo = ctxt.Addressing.ReplyTo.Address.Value;
```

The web service is still called using the Request/Response pattern and on completion, when the *return* statement of the web service is reached, a standard SOAP response is returned. This response can be interpreted as a confirmation message that the SOAP message was successfully received. Below are the headers included in the response generated by a request to the test WSE enabled web service.

```
<soap:Header>
  <wsa:Action>http://tempuri.org/AddResponse</wsa:Action>
  <wsa:MessageID>uuid:e40fccdf-3af9-4856-b65c-f3e3db1ad6a7
  </wsa:MessageID>
  <wsa:RelatesTo>uuid:92bccf15-b71d-4a34-b05e-c967b17830bf
  </wsa:RelatesTo>
  <wsa:To>http://uop-project:13000/</wsa:To>
</soap:Header>
```

As part of the standard request/response pattern the above response is sent back to the requesting host when the web service completes. When a web service is called

that is not WSE enabled then the WS-Addressing headers are not included within the response. The *To* header in the above response matches the *ReplyTo* header as defined in the request. When using the WSE the response to the request is not routed via a new connection to the supplied *ReplyTo* header. The response, including the headers, is returned to requesting application. Therefore, the *To* header in the above example response does not contain the correct location. For the web service to reply asynchronously it generates its own SOAP request message that is sent to the location defined by the received *ReplyTo* header.

The application has to determine whether the called web service provides an asynchronous or synchronous service. The application has to determine if the response received is the result of the service being executed, or if an asynchronous response delivered in the future will contain the result of the service. The BPEL specification provides the statements <invoke> and <receive> to explicitly state that a call to a web service must be performed asynchronously. Therefore for BPEL processes it must be assumed that developer of the process can recognise whether the required service is to be accessed asynchronously or not.

Following the initial test a proof of concept for the proposed broker architecture earlier was created. The proof of concept required four separate elements. These were:

- An application that would call a web service with WS-Addressing information contained within the SOAP headers. The application would also be capable of receiving SOAP messages through an open port. When this application started it would register itself on a UDDI server advertising a URL where SOAP messages could be sent. When this application closed it would un-register itself from the UDDI server.
- A simple web service that reads the WS-Addressing headers and creates a response SOAP message. That response SOAP message is sent to the endpoint defined by the supplied *ReplyTo* header. The *ReplyTo* header contains the address of the broker, this is where the asynchronous reply is sent.
- An application acting as the broker. The broker reads the WS-Addressing headers. messages and firstly attempts to forward the whole SOAP message to an available application as advertised on a UDDI server.
- A UDDI server with publishing permissions enabled.

The proposed mechanism uses open standards to reduce the coupling between client software and an asynchronous web service. The elements described above, when used together, provide a proof of concept that an asynchronous web service can be implemented and still be de-coupled from the client. This demonstrates that by passing the routing information in the SOAP headers that the public interface of web service was not unnecessarily cluttered. Additionally, by routing the asynchronous reply via a broker, further benefits are gained. The final end point of response is not determined at the time of the original request. Any number of end points may be available if originator of the request is no longer available. This is a distinct possibility where the time between request and response could be measured in weeks or months, and would improve the reliability of the overall system. The reliability

can be further improved by the broker queuing responses and retrying where no end points are available.

5. Conclusions

With the advent of the web service standards greater collaborations between technologies is possible. Now business processes can use these distributed services and combine them to create new services. However, business processes often require human interaction. When a human becomes part of the process the process must stop and wait for a response. This could take seconds, minutes, or even months. Therefore asynchronous calls to web services are vital for including human interaction within a business process.

BPEL supports this by providing hard-coded links between services. In an example given by Juric (2004), process A uses service B, but service B must eventually callback to process A. This paper proposes that WS-Addressing is an open standard that enables routing information to be defined within SOAP messages, therefore, enabling a web service to be accessed asynchronously. By supplying the WS-Addressing information to the web service through SOAP headers, the web service interface remains unaffected, and web service need not read the headers if the information they contain is not required. This means that the application calling the web service does not need explicitly specify that the interaction should be asynchronous or not. The web service itself can provide asynchronous callbacks if required.

The mechanism proposed in this paper was tested through developing an application that uses WS-Addressing to enable a web service to be called asynchronously. Microsoft WSE was used to enable a web service to read the supplied WS-Addressing headers allowing the service to be used asynchronously. The test web service was created using .NET and published on Microsoft IIS 6.0. Although this was an entirely Microsoft environment it was used solely to test optionally including the WS-Addressing within the headers of a SOAP message and allowing the web service to read these headers and act upon them. These tests also proved that when a web service is able to perform asynchronously that the asynchronous reply can be routed to a host supplied within WS-Addressing headers that is different for the requesting host. This mechanism, therefore, offers a peer-to-peer approach to web service interaction, using open standards, rather than a traditional client/server approach. The peer-to-peer approach negates the need to poll the service for results and allows the response message to be routed to an available client or clients if required.

6. References

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klien, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S. (2003) “Business Process Execition Language For Web Services 1.1” ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., Winer D. (2000) "*SOAP 1.1*", http://www.w3.org/TR/2000/NOTE-SOAP-20000508 W3 Consortium (accessed 3rd December 2004)

Christensen, E.; Curbera, F.; Meredith, G.; and Weerawarana, S. (2001) "*Web Services Description Language (WSDL) 1.1.*" http://www.w3.org/TR/wsdl (access 3rd December 2004)

Gidgin, M., Hadley M. (2005) "*Web Service Addressing 1.0 – Core*", http://www.w3.org/TR/ /2005/WD-ws-addr-core-20050331 (Accessed 7th July 2005).

Juric, M. B., Mathew, B., Poornachandra, S. (2004) *Business Process Execution Language for Web Services*, Packt Publishing, ISBN 1-904811-18-3, First Published October 2004.

Microsoft (2003) "*Asynchronous Design Pattern Overview*" http://msdn.microsoft.com/ library/default.asp?url=/library/enus/cpguide/html/cpconasynchronousdesignpatternoverview.asp (Accessed 25th July 2005)

Microsoft (2005) "*WSE version 2.0 SP3*" http://www.microsoft.com/downloads/details.aspx?familyid=fc5f06c5-821f-41d3-a4fe-6c7b56423841&displaylang=en (Accessed 26th July 2005)

Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D. F. (2005) *Web Service Platform Architecture*, Prentice Hall, ISBN 0-13-148874-0

Zdun, U., Voelter, M., Kircher, M. (2003) "*Design and Implementation of an Asynchronous Invocation Framework for Web Services*" in proceedings at International Conference on Web Services Europe – 2003.