Web Services using Asynchronous Communication

G M Tere
Department of Computer Science,
Shivaji University, Kolhapur,
Maharashtra - 416004, India

B T Jadhav
Y.C. Institute of Science,
Satara, Maharashtra - 415001,
India

R. R. Mudholkar
Department of Electronics,
Shivaji University,
Kolhapur,
Maharashtra – 416004, India

ABSTRACT

Asynchronous interactions are becoming more important in the implementation of complex B2B Web applications. This paper addresses correlation and coordination issues involved with asynchronous Web services, by studying different mechanisms and metadata structures for supporting them; in addition, several interaction patterns for building asynchronous computations are discussed, and the trade-offs between the various patterns are shown. In conclusion, we illustrate the use of asynchronous Web services in the context of some concrete B2B applications. This paper discusses how to build a web services architecture that handles requests and responses as separate transactions. Not all web services work synchronously; in some situations, responses to web service requests are not provided immediately, but rather sometime after the initial request transactions complete. Such asynchronous operations aren't explicitly supported by web services specifications and standards. In some situations, responses to web service requests are not provided immediately, but rather sometime after the initial request transactions complete. Such asynchronous operations aren't explicitly supported by web services specifications and standards; however, those standards do include the infrastructure and mechanisms on which asynchronous operations can be based. In this paper, several design patterns for asynchronous web services are discussed.

General Terms

Performance, Design

Keywords

Asynchronous patterns, asynchronous transports, synchronous transports, request/reply operations, web services

1. INTRODUCTION

Web services are programmable application logic that is accessible using standard Internet protocols. Web Services provide well-defined interfaces [1], or contracts, that describe the services provided. Invocations of web services are asynchronous in nature in that the service provider must be capable of accepting requests from clients without notice. However, sometimes the response to the web service request is available on the same thread of execution as the invocation; such operations are often labeled as synchronous. This discussion of asynchronous operations will not focus on the initiation of request messages by clients or the consumption of request messages by service providers; rather, this paper focus on how to handle responses to web service requests that are not provided immediately but at a time after the initial request transactions complete. Such asynchronous behavior is common for services that require complex processing that may take minutes or even days to complete - when, for example, the web service implementation is dependent on batch processing or manual steps requiring human intervention.

The designer of a web services client needs to decide how to handle asynchronous responses [3] and how to ensure that his or her implementation is compatible with the way in which a service provider supports asynchronous operations. One option for the client is to issue a request and then block its thread of execution waiting for a response, but for obvious reasons this is not a good alternative; among other problems, it results in resource inefficiencies and raises transactional and scalability issues. The preferred solution is to build asynchronous behavior into the client. The client makes a request as part of one transaction and carries on with the thread of execution. The response message is then handled by a different thread within a separate transaction. In this model, the client as a service requestor requires a notification mechanism and a registered listener component to receive responses. Likewise, there must be a correlator (a correlation or transaction ID) exchanged between the client and service provider for associating responses with their requests. A typically asynchronous scenario would include the following:

- Production and transmission of a request message by a client.
- Consumption of the request message by the service provider.
- Production and transmission of a response message by the service provider.
- Consumption of the response message by the client.

The messages exchanged may be thought of as datagrams for which no reply is needed or expected in order for the transaction to be processed. Through the use of such datagrams, the sending or initiating party of the messages can be fully decoupled from the receiving party, allowing for a truly asynchronous relationship between the two.

1.1 Challenges to be addressed

To support asynchronous operations, one must address many issues that do not exist when responses are synchronous. The tasks that need to be addressed by asynchronous implementations include:

- Defining a correlator and a mechanism for its exchange.
- Defining a reply-to address specifying where the response should be sent, and ensuring that the service provider is informed of this destination.
- The generation of a response by a service provider as a transaction separate from the request.
- The receipt of an asynchronous response by the client.
- The correlation of response with request by both the client and service provider.

1.2 Transports and local interfaces

The transports that can be used for web services communications [3] vary in their capabilities to facilitate the support of asynchronous operations. Thus, it's not only web services behavior that can be described as either asynchronous or synchronous; the transport used for exchanging web services messages also falls into one category or the other. Transports whose interfaces inherently support the correlation of response messages to request messages for application use and support a push and pull type of message exchange are often described as being asynchronous transports. Synchronous transports do not provide these facilities and, when used for asynchronous operations, require that the applications (the client and service provider, for the purposes of this discussion) manage the correlation of messages exchanged by not only defining how the correlator will be passed within each message, but by also matching responses with requests. Examples of transports that can be used in support of asynchronous operations include [7]:

- Asynchronous transports
    - HTTPR
    - JMS
    - IBM MQSeries Messaging
    - MS Messaging
- Synchronous transports
    - HTTP
    - HTTPS
    - RMI/IIOP
    - SMTP
- Regardless of the transport being used for an asynchronous operation, the client (or service proxy used by the client) and the service provider are responsible for generating a correlator that the transports can use in managing the requests and responses.
- Typically, when business partners are utilizing web services to integrate their business processes, they will prefer to use HTTP, HTTPS, and HTTPR as transports for communications across the Internet; within an enterprise, when there are similar application platforms, native transports and interfaces will be used, such as JMS, RMI/IIOP [5], and JCA (Java Connection Architecture).
- The asynchronous transports enable a client to continue processing on its thread of execution immediately after requesting a service invocation; they also provide mechanisms to enable a client to determine the status of its web service requests, and to retrieve responses to those requests.
- Web service implementations that do not provide the ability to initiate the transmission of a response on a separate thread of execution cannot be used for asynchronous operations. Examples of such implementations would be those that use EJBs to front-end database applications or implementations that provide access to enterprise systems through the use of local interfaces such as JCA.

2. ASYNCHRONOUS PATTERNS

The four patterns for support of asynchronous web service operations discussed here are based on the four transmission primitives that an endpoint can support, as defined in version 1.1 of the Web Services Descriptor Language (WSDL) [2] specification:

- One-way: The endpoint receives a message.
- Request/response: The endpoint receives a message, and sends a correlated message.
- Solicit/response: The endpoint sends a message, and receives a correlated message.
- Notification: The endpoint sends a message.

It should be noted that the number of transmission primitives and the number of patterns discussed in this paper are totally independent of each other. Each of the patterns introduces a correlator exchanged between the client and service provider for use in associating responses with requests. The correlator can be provided by either end of the exchange and its creator may be determined based on the underlying transport. For example, when using HTTPR and JMS [8], the source of a message provides the correlator: a transaction ID or a combination of JMSMessageID and JMSCorrelationID. For single-direction operations, if HTTP or HTTPS is used and the reception of the service invocation needs to be confirmed by the client, the client's HTTP protocol handler should block on the invocation waiting for the HTTP response to ensure that the request has been successfully received by the service provider's HTTP listener. For a case in which a service proxy is used by the client, any error conditions associated with the request should result in an exception being thrown. It's important to model the interface of the service proxy to match the WSDL operations defined by the service provider[2]. For example, if a client invokes a one-way operation, the proxy should never return a parameter to the client. Such an exchange would in effect make the operation a request/reply operation, with the reply information not coming from the service provider. The W3C's WSDL working group is expected to expand the language's support for asynchronous operations by providing the ability to define callback mechanisms formally within WSDL. In the meantime, the four primitives listed above can be used in support of asynchronous operations. However, the IDEs and other web services tooling currently available to automate the generation of client-side service proxies typically only support the request/response model.

2.1 One-way and notification operations

In this pattern, the request and the response are two messages defined within separate WSDL operations [2]. The request is modeled as an inbound one-way operation and the response is modeled as an outbound notification operation. Each message is sent as a separate transport-level transmission. This pattern as shown in Figure 1, provides a high level of decoupling between the client and service provider, as it supports the use of two datagrams exchanged between the parties, one for the request and one for the response.

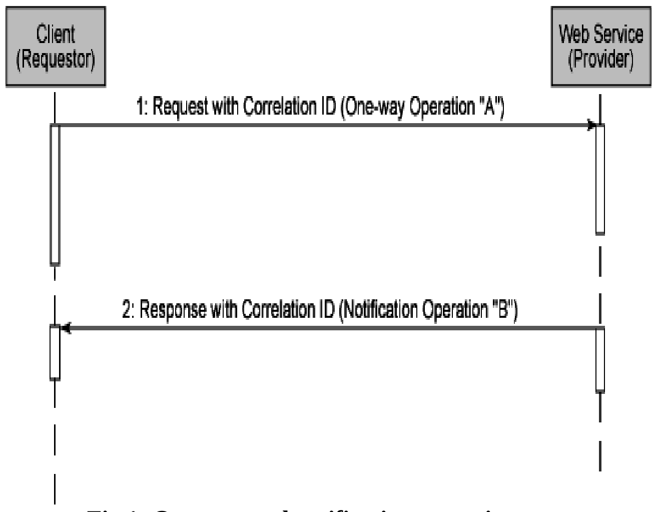
Client (Requestor)
Web Service (Provider)
1: Request with Correlation ID (One-way Operation "A")
2: Response with Correlation ID (Notification Operation "B")

Fig 1: One-way and notification operations

For this pattern, the client is responsible for creating the correlation ID and passing it to the service provider via whatever mechanism has been agreed upon by the two parties. SOAP headers, HTTP headers, and JMSCorrelationIDs would all be acceptable mechanisms [5].

Defining the reply-to address, which indicates where the response should be sent, is also the responsibility of the client, and the means for informing the service provider of this address is determined by how the WSDL is defined for the operations. If the client has a published a notification listener service supporting a one-way operation, its WSDL will contain the port address for the service. Likewise, the service provider will need access to the WSDL of the client's service to determine where to send the response. Access to the WSDL of the notification listener service can be provided when the provider's web service is deployed or at runtime by passing a reference to the WSDL on the initial request. Alternatively, the specific address (for example, the URI) denoting where the response is to be sent can also be provided explicitly as a parameter on the request.

This pattern is also applicable for publish and subscribe (pub/sub) and event notification types of services. A market index update application would be a good example of a pub/sub service; examples of event notification services include applications that notified interested parties about the completion of or exceptions in business process tasks, the completion of a long-running report, or the meeting of certain inventory thresholds. Providing the reply-to address information as a parameter on the request (a request to subscribe to a topic or event, for instance) will enable a service provider to support a large number of subscribers with little administrative support.

For this pattern, in which messages are sent using separate transport-level transmissions without application-level acknowledgements being exchanged between the client and service provider, the transport used should be one considered reliable if the business process that's being supported by the message flows is critical.

2.2 Request/Reply operations

In this pattern, request and response are two messages defined within a single request/reply operation and sent as two separate and unrelated transport-level transmissions.

This pattern, as shown in Figure 2, can also provide a high level of decoupling between the client and service provider, as it supports the use of two datagrams exchanged between the parties for the request and response. However, to use this pattern, the service provider must be a little more sophisticated in processing information at runtime. For example, the service provider will need to be able to handle as an input parameter the address to which it should send the response (for example, the reply-to address).

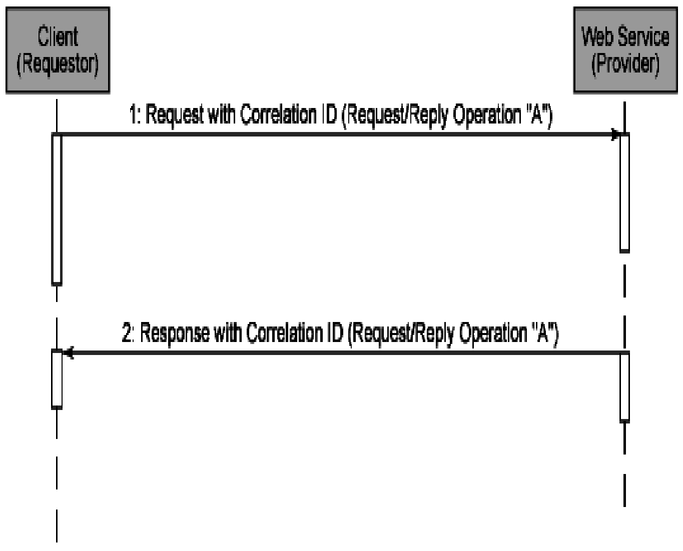
Client (Requestor)
Web Service (Provider)
1: Request with Correlation ID (Request/Reply Operation "A")
2: Response with Correlation ID (Request/Reply Operation "A")

Fig 2: Request/reply operation

For this pattern, the client is responsible for creating the correlation ID and passing it to the service provider via whatever mechanism has been agreed upon by the two parties. Again, SOAP headers, HTTP headers, and JMSCorrelationIDs are among the acceptable mechanisms.

Defining the reply-to address denoting where the response should be sent is also the responsibility of the client. Since a single operation is used for this pattern, a reference to the address or the explicit address itself must be provided as a parameter on the request. For example, if the client has a published an asynchronous response listener service supporting one-way operations, a reference to the service's WSDL can be provided on the initial request.

This pattern is applicable for general-purpose services where the request results in a single response; examples of such services include the persistence or retrieval of data, or the initiation of a business process consisting of a single unit of work, such as an electronic payment. Pattern 2 is similar to Pattern 1, in which messages are sent using separate transport-level transmissions without application-level acknowledgements being exchanged between the client and service provider. Thus, the transport used for this pattern should also be one considered reliable if the business process that's being supported by the message flows is critical [10].

2.3 Request/reply operations with polling

In this pattern, request and response are handled using four messages defined within two separate WSDL operations. The initial request is modeled as a request/reply operation, with two messages (a transmission with a reply) sent as a single transport-level exchange. The response is retrieved by a second request, also modeled as a request/reply operation with two messages sent as a single transport-level exchange [8]. The two operations are meant to be implemented as synchronous flows, with information being returned from the service provider for each request providing the client with a level of acknowledgement per request.

This pattern shown in Figure 3. It enables the client-side implementation to be simpler in support of self-service based solutions, in which the client application initiates all interactions, while also providing a level of decoupling between the client and service provider. However, it's assumed that the request/reply operations are synchronous such that the flows for the reply messages use the native transport reply mechanism (for example, an HTTP Response operation).

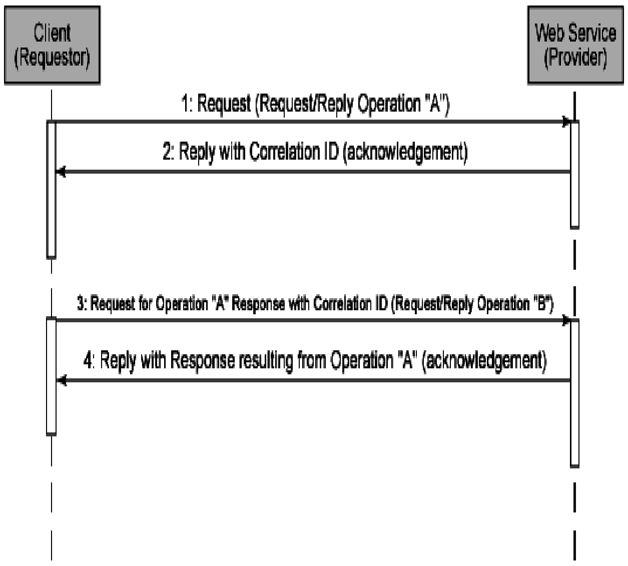
Client (Requestor)
Web Service (Provider)
1: Request (Request/Reply Operation "A")
2: Reply with Correlation ID (acknowledgement)
3: Request for Operation "A" Response with Correlation ID (Request/Reply Operation "B")
4: Reply with Response resulting from Operation "A" (acknowledgement)

Fig 3: Request/reply operation with polling

In the example shown in Figure 3, the service provider generates the correlation ID and the client is responsible for using it to retrieve the response; however, theoretically either side of the exchange could create the correlation ID.

This pattern results in a simpler client implementation, as the notification mechanism and listener components are not required. However, the client must implement a facility with which it can periodically poll for the response from the service provider [11]. This pattern may not be the most efficient, as more than one request per response may be needed to retrieve the response if the initial service request hasn't completed, but it makes for simpler implementations. Thus, this pattern is applicable in cases where simplicity has a priority and where the expected load for the service is low. Examples of service types that could benefit from polling include the initiation of long-running business processes, requests for generation of complex reports, and services used by browser-based customer facing solutions.

3. Request/reply operations with posting

Under this pattern, request and response are handled using four messages defined within two separate WSDL operations. The initial request is modeled as a request/reply operation, with two messages sent as a single transport-level exchange. The response is modeled as a solicit/reply operation, with two messages sent as a single transport-level exchange. The two operations are meant to be implemented as synchronous flows with information being returned from the consuming party for each request providing the requesting party with a level of acknowledgement per request.

This pattern as shown in Figure 4, is similar to Pattern 1 and useful for pub/sub or event notification services when a synchronous transport is used and when the client and service provider require an application-level acknowledgement. Because of this similarity, the example situations given under Pattern 1 could also be addressed by Pattern 4; other examples of service types requiring explicit acknowledgements include services used to exchange business-critical information or confidential information for medical or financial industries -- funds transfers or the initiation of insurance claims, for example.

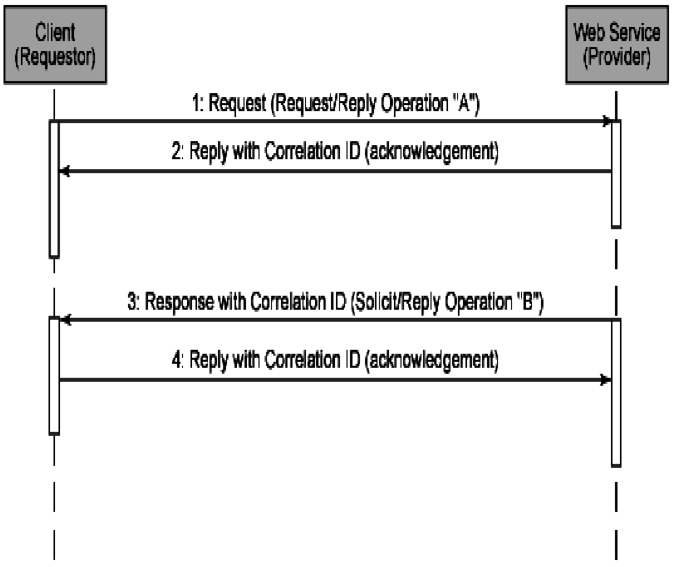
Client (Requestor)
Web Service (Provider)
1: Request (Request/Reply Operation "A")
2: Reply with Correlation ID (acknowledgement)
3: Response with Correlation ID (Solicit/Reply Operation "B")
4: Reply with Correlation ID (acknowledgement)

Fig 4: Request/reply operations with posting

If the roles of the client and service provider are reversed for the handling of the response, the WSDL operation for sending the response can be defined as a request/reply operation for the client. The role reversal is simply for convenience; it facilitates the development of the pattern, as today's tooling does not support solicit/reply operations. The messages that flow between the two parties are not changed; only this article's model for describing the client's and service provider's perspectives is different. Main features of asynchronous patterns [4][7] are summarized in Table 1.

Table 1 : Main features of asynchronous patterns

Pattern	Use cases	Features
Polling	E-commerce B2C, web portals	Very simple to use but not efficient
Callback	Long running async information retrieval services (not business critical)	Hard to implement, but more efficient. Suitable for P2P.
Publish-subscribe	Useful in news, business-information	Same as Callback pattern
Callback with ack.	Variant of Callback, useful to verify published data reception	Inherits Callback features. Adds initial synchronous information.
Publish-subscribe with ack.	Variant of Publish-Subscribe, useful for tracking purposes and in B2B WS transactions	Inherits Publish-Subscribe features. Adds initial synchronous information.

4. CONCLUSIONS

As the industry further develops specifications that determine how to coordinate flows between web services and how to describe dependencies between web services that realize business processes, support for asynchronous operations will be simplified. However, today's web services specifications and standards do not directly describe the support of asynchronous operations, though they do include the infrastructure and mechanisms on which asynchronous operations can be based. Asynchronous web services patterns will serve as foundations on which one can build advanced asynchronous web patterns. The support of asynchronous web service operations can be implemented using both synchronous and asynchronous transport protocols. The use of asynchronous transports, which inherently provide the correlation of request and response messages and provide the mechanisms to query status and retrieve response messages independently, makes the support of asynchronous operations on both the client and service provider sides easier, as message-oriented middleware provides reliable messaging for the transport of web service requests and responses. Likewise, synchronous transports can be used to support simpler implementations that support asynchronous operations, especially where a self-service style is preferred by the client application. Thus using asynchronous patterns we can achieve faster communication. Asynchronous invocations provide a great deal of flexibility for web services users and for the people who write and host web services. With more web services becoming generally available and getting more complex at the same time, more applications are able to integrate those web services into their applications. Because most of the final applications ultimately interact with users, the challenge is to provide a better response to user.

5. ACKNOWLEDGMENTS

We thanks teachers of Department of Computer Science, Shivaji University, Kolhapur for motivating us for this research work. We wish to thank the Principal of Thakur College of Science and Commerce, Mumbai and Principal, Y.C. Institute of Science, Satara for providing resources required to complete this paper.

6. REFERENCES

[1] A. Bosworth et al., Web Services Addressing http://msdn.microsoft.com/ws/2003/03/wsaddressing/, 2003, Accessed on 20th Dec 2011

[2] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl

[3] G. Alonso, F. Casati, H. Kuno, V. Machiraju, Web Services - Concepts, Architectures and Applications. Springer Verlag, October 2003

[4] I. Manolescu, M. Brambilla, S. Ceri, S. Comai, P. Fraternali, Exploring the combined potential of Web sites and Web services. WWW'03 (poster), Budapest, 2003.

[5] JavaTM RMI-IIOP Documentation, http://download.oracle.com/javase/1.3/docs/guide/rmi-iiop/index.html, Accessed on 20th Dec 2011

[6] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, Model driven development of Web Services and hypertext applications. SCII03, June 2003, Orlando

[7] Matt Powell, Asynchronous Web Service Calls over HTTP with the .NET Framework, Microsoft Corporation, 2009

[8] Oracle WebLogic Server, Programming Advanced Features of WebLogic Web Services Using JAX-RPC, 10g Release 3 (10.3), August 2008

[9] Osamu Takagiwa, Adrian Spender, Anthony Stevens, Julien Bouyssou, Programming J2EE APIs with WebSphere Advanced, IBM, 2001

[10] P. Yendluri, Web Services Reliable Messaging, http://webservices.org/index.php/article/articleview/1148/1/24, Accessed on 22nd Dec 2011

[11] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera. Designing Data-Intensive Web Applications.Morgan-Kaufmann, Dec. 2002. IBM, BPEL4WS Version 1.1, http://www-106.ibm.com/developerworks/library/ws-bpel/, Accessed on 22nd Dec 2011