Asynchronous operations and web services, Part 1: A primer on asynchronous transactions

Build a web services architecture that handles requests and responses as separate transactions

Holt Adams

April 01, 2002

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; however, those standards do include the infrastructure and mechanisms on which asynchronous operations can be based. In this article, Holt Adams explains why any web services architect needs to understand how asynchronous operations work. This article will help you begin to adapt your own services for an asynchronous environment.

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, I'll 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.

In this paper, I'll examine the theoretical basis for asynchronous web services. You'll learn what constitutes an asynchronous transaction, and what transports can carry it. I'll also explore the specific challenges that developers face when building support of asynchronous operations into their web services.

The designer of a web services client needs to decide how to handle asynchronous responses 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.

**Peer-to-peer and proxy invocations**

For the purposes of this series, I will consider a direct invocation of a provider’s web service by a client (that is, by a web service request) a peer-to-peer invocation, where the requestor and provider support the same transport and messaging protocol (for example, SOAP/HTTP or SOAP/JMS). This invocation style is a typical use of a web service.

Invocation of a provider’s web service by a client through an intermediary, where the intermediary performs data transformation, transport conversions, security, and logging functions, will be considered a proxy invocation. The target service is actually invoked by the intermediary, which passes the request onto the target service using its native transport and forwards the response back to the client. The gateway is transparent to the client and web service provider, so from their perspective the invocation style is peer-to-peer.

**Challenges to be addressed**

To support asynchronous operations, you 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.
**Transports and local interfaces**

The transports that can be used for web services communications 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:

- **Asynchronous transports**
  - HTTPR
  - JMS
  - IBM MQSeries Messaging
  - MS Messaging
- **Synchronous transports**
  - HTTP
  - HTTPS
  - RMI/IIOP
  - SMTP

You can find more information on many of these transports in the [Related topics](#) section below.

**IBM's Web Service Invocation Framework and asynchronous operations**

The Web Service Invocation Framework (WSIF) provides a client API for invoking web services asynchronously using a local proxy (for example, a stub) with the capability to use different transports based on the context of the invocation. WSIF components allow true support for asynchronous web services operations.

For more information on WSIF, see the [Related topics](#) below.

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, 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 can not 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.

**Conclusions and coming attractions**

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. You should now have an idea of how you can start building support of asynchronous operations into your web services today.

In the second installment of this series, I'll take a look at some asynchronous web services patterns. These patterns will serve as foundations on which you can build advanced asynchronous web patterns. You can prepare yourself for that article by mastering the concepts outlined here.
Related topics

- Read the second part of this article, *Asynchronous operations and web services, Part 2.*
- You can find a [Java Messaging Service API tutorial](http://java.sun.com/products/jms/2.0/docs/tutorial/) at Sun Microsystems' web site.
- The Java Community Process's site has more information on the [Java Connector Architecture](http://java.net/projects/jca).
- For more on programming J2EE APIs with WebSphere, with a particular look at messaging with JMS, WebSphere, and MQSeries, check out this [IBM Redbook](http://www.redbooks.ibm.com/redbooks/pdfs/sg246962.pdf) (in PDF format).
- Check out IBM's [Web Services Invocation Framework](http://www-1.ibm.com/software/webservers/services/), an alphaWorks technology.

© Copyright IBM Corporation 2002
[Trademarks](http://www.ibm.com/developerworks/ibm/trademarks/)
(www.ibm.com/developerworks/ibm/trademarks/)