

Tight-coupling Web services in the SOA

Skill Level: Intermediate

Judith M. Myerson (jmyerson@bellatlantic.net)
Systems Engineer and Architect

24 Jan 2008

Look at the pros and cons of both tight and loose coupling Web services and the resulting change in scale that comes from tight coupling. This article includes examples of criteria to measure performance of tightly coupled Web services during the testing process.

Introduction

My developerworks series, [Use SLAs in a Web services context](#) covers mitigating risks for vulnerabilities and integrating Web services into Enterprise Application Integration (EAI) with a service-level agreement (SLA) guarantee. My other developerworks series, [Work with Web services in enterprise-wide SOAs](#), talks about load-balancing Web services and integrating radio frequency identification (RFID) Web services into EAI applications. This series also explores developing risk management Web services, migrating legacy service components as discoverable Web services, and developing Web services to integrating SAP with IBM® DB2® and Oracle using IBM WebSphere® MQ.

In each of these, I've attempted to show how Service-Oriented Architectures (SOAs) are associated with loose coupling among Web services and other interacting software agents. In a general sense, I've proposed that you might need to tight-couple some Web services as a result of resources becoming scarce due to changes in scale and where execution speed is critical.

Applications, systems, and networks are generally growing more rapidly than their given resource capacity, and this includes message queues for Web services. This creates security and performance concerns where anything over the maximum capacity limits at any time results in system overloads of message-based Web services.

In this article, learn about:

- Tight coupling versus loose coupling.
- Why you need to tight-couple Web services.
- How synchronous business functions can appear as an asynchronous, loosely coupled Web service.
- How the coupling of a Web service can be switched from loose to tight.
- What criteria should be used to measure performance.
- What the constraints on measurement are.

Tight coupling versus loose coupling

Most large, complex systems are built as small collections of large subsystems instead of as large collections of small, independent subsystems. This is because of the potential for increased performance, security, economy, or some other key property that you can't get by decoupling the system into relatively independent, small elements. The tight coupling characteristics of large-scale systems generally result from optimizing the overall design and from minimizing redundancies and inefficiencies among the system's components. This results in closer coupling among the system's components and large numbers of critical interdependencies.

One disadvantage of tight coupling among a system's components is that failures within the individual components tend to disable the entire system. Loosely coupled Web services are seen as a better alternative; a Web service failure doesn't disable the entire system, provided a failover Web service server is in place.

You can change details in loosely coupled Web services as long as those changes don't affect the functionality of the called Web services. The tight-coupled systems can be difficult to maintain, because changes in one system subcomponent usually require the other subcomponent to adapt immediately.

Loosely coupled Web services require substantial redundancies unlike tight coupling between clients and service, which minimizes redundancies. The listening Web service and the requesting Web service might not trust each other. This means security and trust standards must be added to get both the listener and requester to trust each other. On the other hand, tightly calling and called coupled systems assume that both have the knowledge of what each requires to trust one another.

Why use tight coupling for Web services?

Web services are normally message based and loosely coupled whether the

resource is scarce or not; they wait for an answer via message queuing before they take further action, if any, based on the contents of these messages. They have the advantage of messages being passed instead of method invocations and provide a degree of independence between the sending and receiving Web services.

It takes a large number of Web services to result in system overloads if they collectively exceed the resource capacity of message-queuing systems and if the capacity to handle peak loads wasn't considered in the planning stage of building or reusing Web services with new functionalities. It's time to start planning for the tight coupling of some Web services.

Asynchronous versus synchronous

A loosely coupled Web service application can communicate asynchronously and synchronously. Asynchronous Web services communicate with one another as long as the applications support the same standard message-based interface and speak a standard, interoperable language (SOAP, for example). Answers are guaranteed with exception-handling and error-recovery mechanisms. Synchronous Web services require that they be compatible to communicate.

Asynchronous loosely coupled Web services spend time waiting for a response. Synchronous loosely coupled Web services can't wait for an answer, locking up the program until the response comes back. Synchronous calls can be programmed, but with timeouts so that if the call can't come back in a certain time, the calling Web service can be unlocked to exit from the system.

One problem with asynchronous loosely coupled Web services is that for some business functions, it can exceed its resource capacity for the message queuing servers or system.

Business functions

While business is usually asynchronous because of reliability and scalability reasons, some business functions require synchronous transactions, such as some queries for inventory, schedule changes, credit card transaction approvals or disapprovals, and so on. Let's take a look at an online book Web service scenario where loosely coupled synchronous business functions appear to be in the asynchronous mode when they aren't.

Asynchronous mode

Let's use the example of an online bookstore and the process for making a transaction. First you select your choices, then put them into the shopping cart. Maybe you remove some or add others. Ultimately, though, the shopping cart Web

service has been waiting for you in asynchronous mode to finish your shopping. When you're done, you choose a credit card payment option to make the purchase.

Then, the system begins to check with an outside source for verification of card information. If you get a message asking you to wait for a response for a few minutes, the system is in asynchronous mode. On the other hand, if the system instantly verifies and approves your credit information, it's in synchronous mode.

Synchronous mode

The synchronous mode locks up the Web service application while waiting for a quick response. If the application has set a timeout for several seconds, it exits in a given amount of time, unlocks the application, then asks you to return later to finish your transaction. If you get a response, you know if your credit card information is approved or disapproved. Upon approved information, you get a thank-you note, then you wait asynchronously for the books to arrive.

Lack of switching mode

The problem is that some asynchronous functions in this Web service don't have the capability to switch from loose to tight coupling mode in response to an event alert mechanism indicating that the system change in scale exceeds the system's resource capacity.

Change in scale

Volumes of resources can change from low to high, and vice versa, in the background while applications are running. Theoretically, when volumes are high, loosely coupled Web services wait for a response from a message queuing server. In reality, the message queuing resource is either scarce or isn't scarce while Web services are waiting to send or receive a message

Coupling switching

To get a certain Web service to respond to a change in scale, consider a Web service that can switch from loose coupling to tight coupling when triggered by event alert mechanisms when its corresponding resources have reached certain levels. In making the switch, the resources can be dynamically allocated when needed and deallocated when not needed by Web services.

WS-Addressing

When a Web service is tightly coupled, it should turn on the WS-Addressing EndpointReference with ReferenceProperties while turning off WS-Context. The WS-Addressing session model provides coupling between the Web service endpoint information and the session data, which is analogous to object references in

distributed object systems.

WS-Context

When the Web service switches to loosely coupled mode, it should turn on WS-Context while turning of WS-Addressing. WS-Context provides a session model that's an evolution of the session models found in HTTP servers and transaction systems, allowing a service client to more naturally bind the relationship to the service dynamically and temporarily.

Measuring performance

Whether you plan to tight-couple some Web services or redesign some existing Web services to be able to make the coupling switch, you should set a performance measurement goal. To set the goal, consider what performance criteria to use at both network and application levels and what constraints may impact the performance.

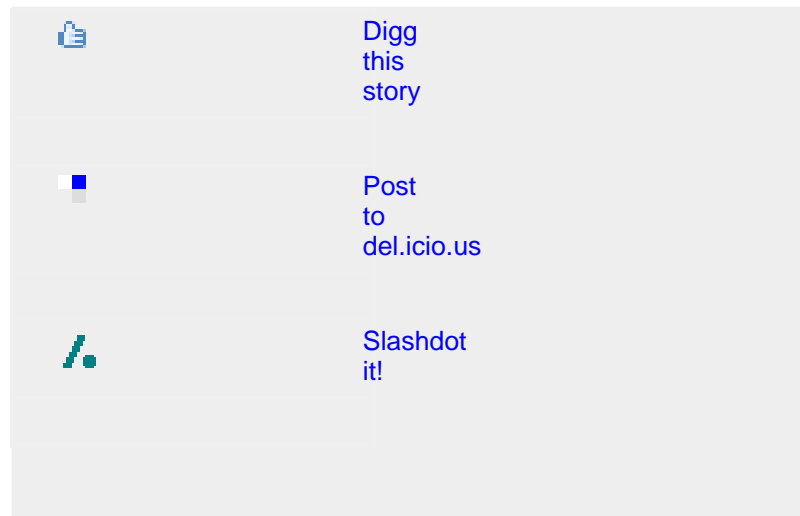
Here are some things to consider when setting performance criteria:

- Maximum time each Web service has waited for a message before receiving it
- How long and how often the range of maximum times has occurred within a given period of time
- Length of time to make the mode switch and to complete the process of switching
- Frequency of the range of maximum times (for example, after dinnertime and during the weekend)
- Peak times during the first month the Web services are deployed
- Impact of the change of scale on guaranteed availability in the SLA
- Range of guaranteed service availability to achieve during the testing process and the production

All of these recommendations are subject to constraints, such as SLA, security policies, testing process, corporate policies, and government regulations.

Conclusion

Share this...



You need a team of developers, testers, and system administrators to consider tight coupling of Web services. You must plan ahead for issues of time development, testing, and deploying tightly coupled Web services and those loosely coupled Web services that can switch to tight coupling mode to avoid system overloads of scarce resources. Resolving these issues helps with the jobs of design, development, and deployment of tightly coupled Web services. You can use IBM Rational® ClearQuest® and IBM Rational Functional Tester to increase productivity by reducing testing time, defect tracking time, and the costs of test labs in an enterprise SOA development project.

Resources

Learn

- Check out the [Work with Web services in enterprise-wide SOAs](#) series by Judith M. Myerson.
- Read Judith's series [Use SLAs in a Web services context](#) for details on service-level agreements.
- Want to know more about tight coupling? Read it in IBM Systems Journal's "[Developing XML Web services with WebSphere Studio Application Developer.](#)"
- Read Judith Myerson's [The Complete Book of Middleware](#), which focuses on the essential principles and priorities of system design and emphasizes the new requirements brought forward by the rise of e-commerce and distributed integrated systems.
- Read [Enterprise Systems Integration, Second Edition](#) to get the business insight and the technical know-how to ensure successful systems integration.
- Bring your organization into the future with [RFID in the Supply Chain](#), which explains business processes, operational and implementation problems, risks, vulnerabilities, and security and privacy.
- IBM Redbooks®: Go into the nuts and bolts of developing a service-level agreement in [Tivoli® Manager for Domino V2.1: Fulfilling Service Level Agreements Using Tivoli Technology](#).
- Visit the [Developer Bookstore](#) for a comprehensive listing of technical books, including hundreds of [Web services titles](#).
- The [SOA and Web services zone](#) on IBM developerWorks hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.
- Play in the [IBM SOA Sandbox!](#) Increase your SOA skills through practical, hands-on experience with the IBM SOA entry points.
- The [IBM SOA Web site](#) offers an overview of SOA and how IBM can help you get there.
- Stay current with [developerWorks technical events and webcasts](#).
- Browse for books on these and other technical topics at the [Safari bookstore](#).
- Check out a quick [Web services on demand demo](#).

Get products and technologies

- Innovate your next development project with [IBM trial software](#), available for download or on DVD.

Discuss

- [Participate in the discussion forum for this content.](#)
- Get involved in the developerWorks community by participating in [developerWorks blogs](#), including the following SOA and Web services-related blogs:
 - [Service Oriented Architecture -- Off the Record](#) with Sandy Carter
 - [Best Practices in Service-Oriented Architecture](#) with Ali Arsanjani
 - [WebSphere SOA and J2EE in Practice](#) with Bobby Woolf
 - [Building SOA applications with patterns](#) with Dr. Eoin Lane
 - [Client Insights, Concerns and Perspectives on SOA](#) with Kerrie Holley
 - [Service-Oriented Architecture and Business-Level Tooling](#) with Simon Johnston
 - [SOA, ESB and Beyond](#) with Sanjay Bose

About the author

Judith M. Myerson

Judith M. Myerson is a systems architect and engineer. Her areas of interest include middleware technologies, enterprise-wide systems, database technologies, application development, network management, security, RFID technologies, and project management.

Trademarks

ClearQuest, DB2, IBM, the IBM logo, Rational, Redbooks, Tivoli, and WebSphere are registered trademarks of IBM in the United States, other countries or both.