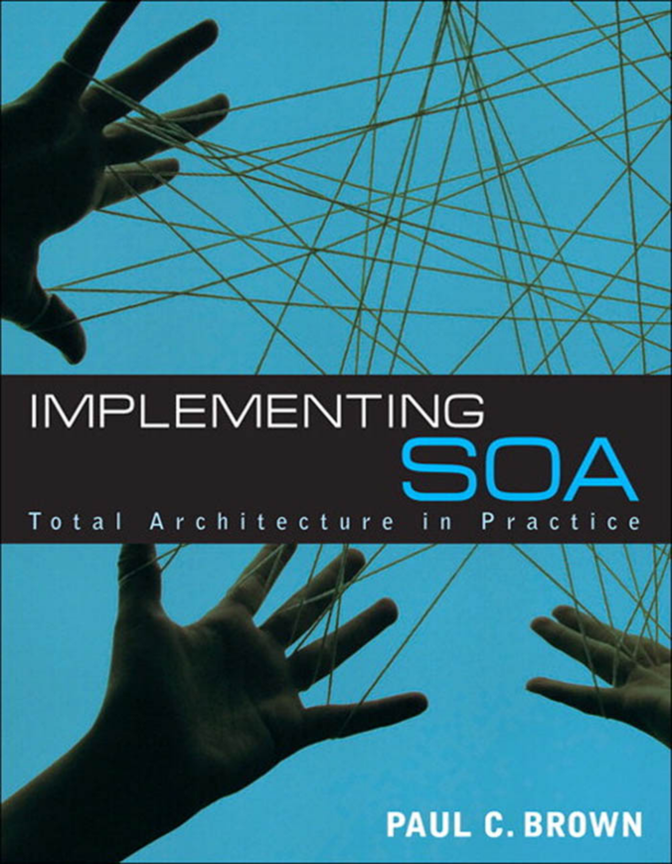

# IMPLEMENTING SOA

Total Architecture in Practice

PAUL C. BROWN

Visit us on the Web: informit.com/aw

*Library of Congress Cataloging-in-Publication Data*

Brown, Paul C.
Implementing SOA : total architecture in practice / Paul C. Brown.
p. cm.
Includes index.
ISBN 0-321-50472-0 (pbk. : alk. paper)
1. Computer architecture. 2. Computer network architectures. 3. Business enterprises—Computer networks—Management. 4. Web services. I. Title.
QA76.9.A73B82 2008
004.2'2—dc22 2008005257

ISBN-13: 978-0-321-50472-2
ISBN-10: 0-321-50472-0

Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.
First printing, April 2008

# Preface

If you are an architect responsible for a service-oriented architecture (SOA) in an enterprise, you face many challenges. Whether intended or not, the architecture you create defines the structure of your enterprise at many different levels, from business processes down to data storage. It defines the boundaries between organizational units as well as between business systems. Your architecture must go beyond defining services and provide practical solutions for a host of complex distributed system design problems, from orchestrating business processes to ensuring business continuity. Implementing your architecture will involve many projects over an extended period, and your guidance will be required.

In *Succeeding with SOA*, I discussed the need for an enterprise to pay close attention to its architecture, the role of its architects, and the importance of setting the right organizational context for their success. In this book, *Implementing SOA*, I turn to the work of the architects themselves—your work—guiding you through the process of defining a service-oriented architecture at both the project and enterprise levels. Whether you are an architect putting SOA into practice or you are an engineer aspiring to be an architect and wanting to learn more, I wrote this book for you.

Doing SOA well can be very rewarding. Done properly, your enterprise will comprise a robust and flexible collection of reusable business and infrastructure services. The enterprise will be able to efficiently recombine these services to address changing business needs. On the other hand, if you do SOA poorly, your enterprise will be encumbered with a fragile and rigid set of functionality (which I hesitate to call services) that will retard rather than promote enterprise evolution. You don't want to end up there. *Implementing SOA* will show you the pitfalls as well as the best practices. In short, it will guide you to doing SOA well.

## The SOA Architectural Challenges

Doing SOA well presents you with four interrelated architectural challenges.

1. Services define the structure of both business processes and systems. Business processes and systems have become so hopelessly intertwined that it is no longer possible to design one without altering the other. They have to be designed together, a concept I call *total architecture*. Thus, building your service-oriented architecture is not just a technical exercise, it is also a business exercise that requires the active participation of the business side of the house.
2. You are not building your SOA from scratch. Your enterprise today operates using a working set of business processes and systems. You can't afford to disrupt business operations just because you want to build an SOA. Practically speaking, you need to evolve your existing business processes and systems into an SOA. During this transition, individual projects must continue to deliver tangible business value, independent of your SOA initiative.
3. Your SOA is a vision that requires a consistent interpretation as it is put into practice. The actual implementation of your SOA will happen piecemeal, project by project. Services that are developed in today's project must satisfy future needs, and today's projects must leverage the services developed in yesterday's projects. Ensuring that existing services are appropriately used, and that new services will meet future needs, requires coordination and planning across multiple projects, both present and future.
4. A service-oriented architecture is actually a distributed system. As such, your SOA must incorporate self-consistent solutions to all of the classic distributed system design problems: trading off service granularity against communications delays, coping with communications breakdowns, managing information that is distributed across services and sites, coordinating service execution and load distribution, ensuring service and business process availability and fault tolerance, securing your information, and monitoring and managing both business processes and services. The requirements driving your solution choices stem from the needs of the business processes involved and are thus tied in with business process design as well as systems design. As before, solutions to these problems require consistent approaches across all of your projects.

At the end of the day, your challenge as an architect is to organize your enterprise's collaboration between business processes, people, information, and systems, and to focus it on achieving your enterprise's goals.

## About the Book

*Implementing SOA* is a comprehensive guide to addressing your architectural challenges. It shows you how to smoothly integrate the design of both business processes and business systems. It will tell you how to evolve your existing architecture to achieve your SOA objectives, maintaining operational support for the enterprise during the transition. It demonstrates how to use a proactive enterprise architecture group to bring a consistent and forward-looking architectural perspective to multiple projects. Finally, it shows you how to address the full spectrum of distributed system design issues that you will face.

This book is organized into nine parts. Part I presents the fundamental concepts of architecture, services, and the total architecture synthesis methodology. Parts II through VIII discuss a series of architectural design issues, ranging from understanding business processes to monitoring and testing your architecture. Part IX then builds on these discussions to address the large-scale issues associated with complex business processes and workflow, concluding with a summary discussion of the workings of the enterprise architecture group.

In Parts II through VIII, each of the architecture topics is discussed from two perspectives: the project perspective and the enterprise architecture perspective. Each part first discusses the design issues as though the project architect were creating the entire architecture from scratch. The last chapter in each part then addresses the realities of a multi-project environment and the role that the enterprise architecture group must play to ensure that the design issues are appropriately addressed throughout the total architecture. This separation highlights the relative roles of the project and enterprise architects as well as the manner in which they need to collaborate. The enterprise architecture group chapter in Part IX then summarizes the activities of this group.

The book as a whole, and each individual chapter, can be approached in two ways. One way is prescriptive. The book presents a structured approach to tackling individual projects and managing the overall enterprise architecture. The other way is to use the book as a review guideline. Each chapter discusses a topic and concludes with a list of key questions related to that topic. Use the questions as a self-evaluation guide for your current projects and enterprise architecture efforts. Then use the content of the individual chapters to review the specific issues and the various ways in which they can be addressed. Either way, you will strengthen your enterprise architecture.

*Implementing SOA* is a comprehensive guide to building your enterprise architecture. While the emphasis is clearly on SOA, SOA is just a style of distributed system architecture. Real-world enterprise architectures contain a mixture of SOA and non-SOA elements. To reflect this reality, the discussions in this book extend beyond SOA to cover the full scope of distributed business systems architecture.

The pragmatic approach of *Implementing SOA* will guide your understanding of each issue you will face, your possible solution choices, and the tradeoffs to consider in building your solutions. The key questions at the end of each chapter not only provide a convenient summary, but also serve as convenient architecture review questions. These questions, and the supporting discussions in each chapter, will guide you to SOA success.

## Acknowledgments

This book is dedicated to my wife, Maria. Without her love and support, neither this book nor the previous one would ever have come into existence. She picked up the slack for many things I should have been doing and gave me encouragement when I grew weary of the task. Mere words are inadequate to express my love and appreciation.

There are many who have helped along the way as well. I thank my mentors who helped me learn how to explore uncharted territory: John Reschovsky, Joel Sturman, David Oliver, David Musser, and Mukkai Krishnamoorthy. I thank my colleagues who have provided an intellectual foil for these ideas: Jonathan Levant, John Hutchison, James Rumbaugh, Michael Blaha, and William Premerlani. For their support of my enterprise methodology work, I thank Brian Pierce, Bruce Johnson, Paul Beduhn, and Paul Asmar. For their help in sharpening the real-world architectural concepts, I thank Paul Asmar, David Leigh, Saul Caganoff, and Janet Strong. For helping me turn a concept into a book, I thank Michael Blaha and William Premerlani. For helping me make this book a reality, I thank Paul Asmar, Ram Menon, Roger Strukhoff, Scott Fingerhut, Peter Gordon, Michael Blaha, and Charly Paelinck.

PCB
Schenectady, NY
February 19, 2008

Chapter 4

# Using Services

Services, on their own, provide no benefit. To get benefit from your services, you need to employ them as an element of a larger process—a process that provides value to your enterprise. Other participants in the process need to interact with the service and benefit from the service results. Thus you need to be able to integrate the service with other services and nonservice functionality as well as to form your business processes. This chapter explores a number of choices that are available for integrating services into business processes.

## Service Interaction Patterns

### Synchronous Request-Reply

When you think of “using” a service, what probably comes to mind first is having the service user ask the service provider to perform the service—and then waiting for the result. This style of interaction is characterized by the synchronous request-reply pattern shown in Figure 4–1. It is perhaps the most common service interaction style.

The synchronous request-reply pattern is simple. The service provider only needs to supply a single operation interface (Figure 4–2). The service user calls this operation to submit the request, and this same operation provides the mechanism for returning the result.

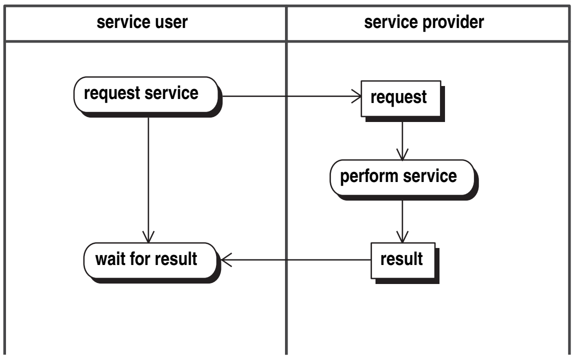

**Figure 4–1:** *Synchronous Request-Reply Interaction Pattern*

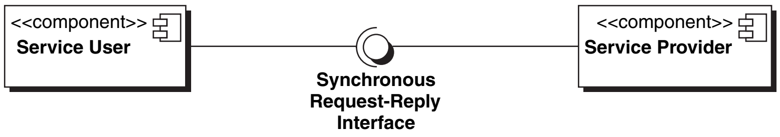

**Figure 4–2:** *Synchronous Request-Reply Interface*

## Asynchronous Request-Reply

The synchronous request-reply interaction pattern is not sufficient to build most real-world business processes. Much as you (and the retailer) might like it, when you order a book online, the book does not arrive while you are still sitting at the keyboard! You don't wait for the book, to the exclusion of all other activity, until it arrives. Instead, you go off about your business for a few days until the book is delivered. This is the style of interaction represented by the asynchronous request-reply pattern shown in Figure 4–3.

Since the asynchronous request-reply pattern delivers the result at some future point, it requires a mechanism—a second operation—for the service provider to deliver the result. If the service provider is to initiate the delivery of the result, then the *service user* typically provides an operation and corresponding interface for this purpose (Figure 4–4). Note that for the service provider to use this interface, the original request must tell the service provider about the interface to be used to deliver the result.

Alternatively, the service provider could supply a result retrieval operation with its corresponding interface (Figure 4–5). With this approach,

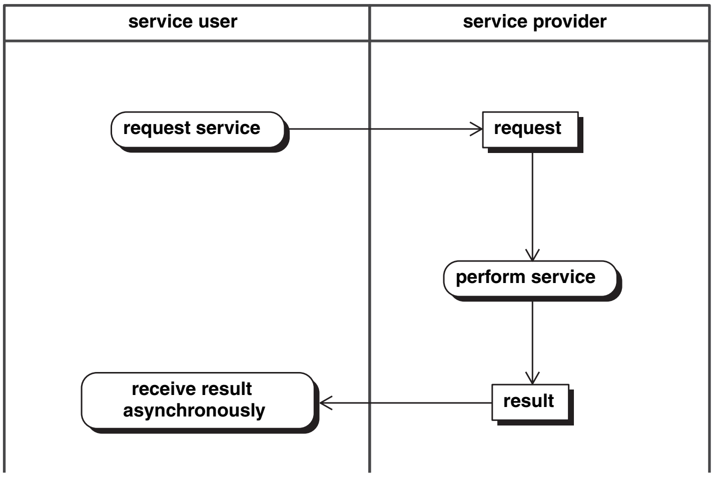

**Figure 4–3:** *Asynchronous Request-Reply Interaction Pattern*

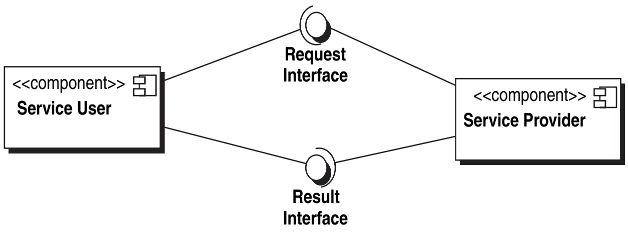

**Figure 4–4:** *User-Supplied Asynchronous Result Interface*

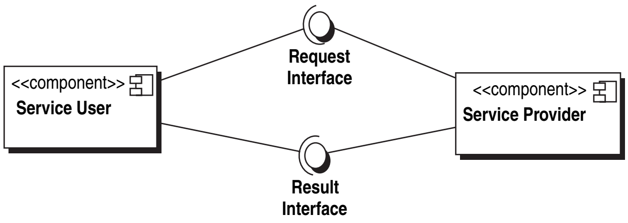

**Figure 4–5:** *Provider-Supplied Asynchronous Result Interface*

the service user invokes the result retrieval operation to determine whether the result is ready and retrieve it if it is. This is the style of interaction you use when you take your laundry to the dry cleaners. In the first interaction, you deliver the dirty laundry, and in the second you go back to pick up the cleaned items.

When the service provider supplies a result retrieval operation, the service user is the one who initiates the second interaction. For efficiency, the service provider typically specifies a time after which the results will be available. This response-time service-level agreement (SLA) is a promise, but there is no absolute guarantee that your laundry will be ready at that time. But if the service provider meets the promise most of the time (99%), it makes for an efficient interaction. In the absence of such an SLA, the service user can only guess at when the result will be ready. This guesswork leads to the repeated invocation of the interface to determine whether the results are ready—an inefficient process often referred to as polling.

Message-based service access, which involves a third-party communications intermediary, is inherently asynchronous. Thus message-based communications provides a convenient alternative for returning asynchronous results. Message-based approaches are discussed later in this chapter.

## Subscription

Subscription services (Figure 4–6) deliver more than one result. They provide an ongoing series of results spread out over time. In using a subscription service, the service user registers with the service provider to receive a series of results—asynchronously. Your newspaper subscription is an example of this type of service, as are the stock market activity alerts delivered to your phone or computer.

Like the asynchronous request-reply pattern, the subscription pattern also requires two operation interfaces (Figure 4–7). If the parties are interacting directly, generally one of the interfaces will be provided by the service provider and the other by the service user. The service provider supplies a synchronous request-reply subscription interface that results in acknowledgment of the subscription. The service user provides an interface for the subsequent delivery of the expected results. As with asynchronous request-reply, the subscription request must indicate how the results are to be delivered.

## Unsolicited Notification

The existence of the service user's delivery interface opens up the possibility for a fourth interaction pattern: the unsolicited notification (Figure 4–8). Once a service provider becomes aware of the presence of

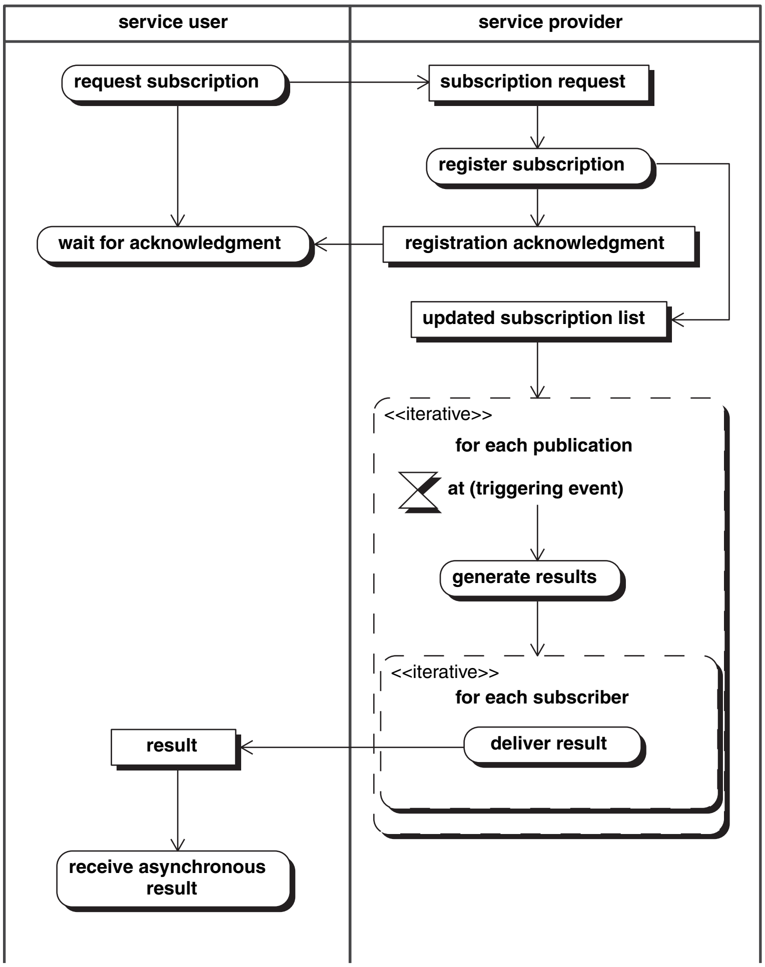

**Figure 4–6:** *Subscription Interaction Pattern*

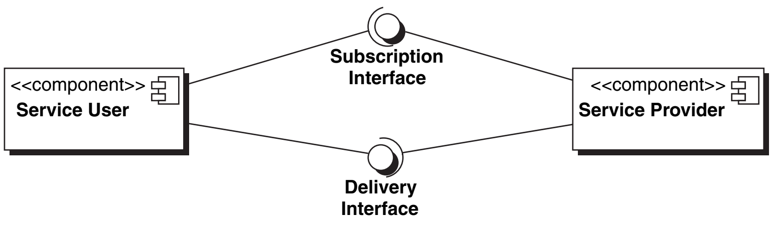

**Figure 4–7:** *Typical Subscription Interfaces*

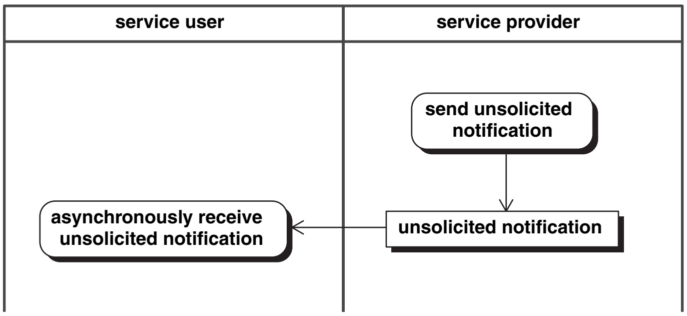
service user
service provider
send unsolicited notification
unsolicited notification
asynchronously receive unsolicited notification

**Figure 4–8:** *Unsolicited Notification Pattern*

a delivery interface, there is nothing stopping the service provider from sending results that were not explicitly requested. The most obnoxious form is, of course, junk mail and its electronic equivalent, spam. But unsolicited notification can be beneficial as well. Companies often notify their employees and customers of significant events using such notifications. If you see flames in an auditorium, yelling "Fire!" is an unsolicited notification as well. You (the service provider) are notifying others in the auditorium (the service users) of the presence of a dangerous situation.

Unsolicited notifications only require a single interface—the delivery interface on the service user (Figure 4–9). As you shall see shortly, the use of a messaging service provides another means of delivering unsolicited notifications.

## Interaction Pattern Summary

More complex interaction patterns than these are, of course, possible. But every interaction pattern can be assembled from these four basic patterns: synchronous request-reply, asynchronous request-reply, subscription, and unsolicited notification. Therefore, a detailed understanding of these four patterns will provide you with the tools to analyze any interaction pattern you may encounter. Chapter 27 will

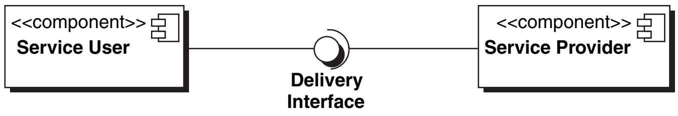

**Figure 4–9:** *Unsolicited Notification Interfaces*

explore the properties of these patterns in more detail and will elaborate on some of the more common complex interaction patterns as well.

## Service Access

On the surface, accessing a service sounds so simple—just call the interface! In reality, access can get very complicated as you try to control access to services and maintain flexibility concerning where service users and providers are deployed. The following sections explore a number of the design issues you will encounter and the design patterns that you can use to address them.

### Direct Service Access

The most obvious way to use a service is to directly access the service provider's interface (Figure 4–10). While this is straightforward, it requires that the service user be aware of both the functionality provided by the service interface (which you would expect) and the location of the interface. If you are using HTTP to access your service, for example, then the service user needs to know either the IP address or hostname of the machine on which the service is running as well as the specific socket corresponding to the service's interface.

This requirement to know about the interface's location makes the design of the service user dependent on the deployment specifics of the service provider. If the service provider is moved from one machine to another, then the service user's configuration must be updated to reflect the change. If the service becomes very successful and has many users, such dependencies make it difficult to move the service provider. Such movement might be required to add capacity by moving the service to a larger machine or recover from a machine outage by moving the service to an alternate machine. While network-level solutions such as virtual IP addresses and virtual hostnames pro-

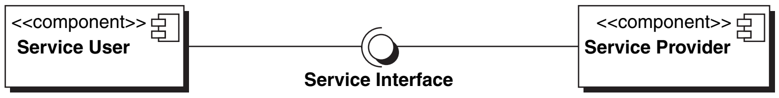

**Figure 4–10:** *Direct Service Access*

vide some relief, they have their own limitations and accompanying administrative costs as well.

## Variations in Direct Service Access

### *Service Lookup*

One means of avoiding this dependency between the service user and the location of the service interface is to employ a lookup service (Figure 4–11). The lookup service knows the actual location of the provider's service interface. In order for the service user to access the service, it first uses the lookup service to get the actual location of the service interface and then uses this information to access the service. The JNDI (Java Naming and Directory Interface)[1] is a service interface that is commonly used for this purpose. The advantage of this approach is that when a service provider changes location, only the lookup service needs to be updated. The service users will pick up the change from the lookup service.

When a lookup service is used, the lookup typically occurs just the first time a particular user wants to access the service. Once the service interface is located, the user continues to use that interface as long as it is operational.

Of course, the lookup service itself now presents exactly the same problem that it solves for other services: the service user must know the location of the lookup interface to be able to use it. But at least the problem is reduced in scope to just accessing the lookup service, which presumably will not change its interface location very often.

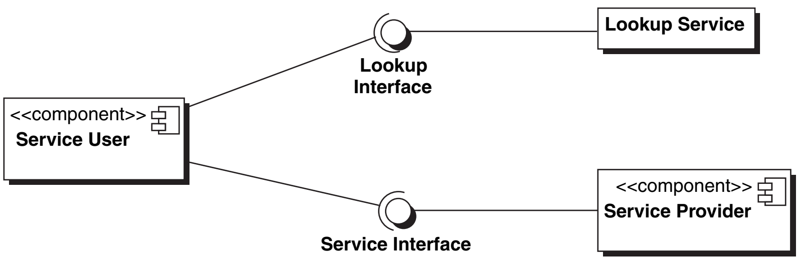

**Figure 4–11:** *Lookup Service*

1. http://java.sun.com/products/jndi/.

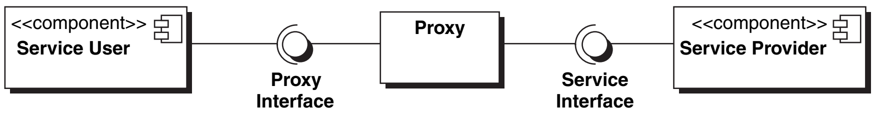

**Figure 4–12:** *Proxy Access Interfaces*

*Proxy Access*

Another approach that can isolate the service user from the details of the service provider's deployment is to employ a level of indirection—a proxy—for accessing the service (Figure 4–12). To the service user, the proxy presents what appears to be the service's interface without knowing that it is, in reality, a proxy interface. The proxy forwards all incoming requests to the real service interface and forwards replies from the service interface back to the service user through the proxy interface.

## Direct Access Limitations

There are a couple of drawbacks to the direct access approach and its service lookup and proxy variants. One is that any change in the location of the service provider's interface requires changes to the configuration of some other component. In direct access, the component requiring update is the service user. With service lookup, it is the lookup service, and with a proxy it is the proxy. This dependency of other components on the physical location of the service provider's interface complicates changing the service provider's location. It adds administrative overhead and adds to the time it takes to implement such changes.

This administrative overhead affects more than just the normal deployment of services. It comes into play when you want to provide a fault tolerant or highly available service as well. In such cases, when something happens to the existing service provider you need to bring up a replacement copy of the service provider. This replacement copy will be in a different location, generally on a different machine. For local failover, this different machine may be in the same data center, but in the case of a site disaster it will be in a different data center. In such cases it becomes very difficult to maintain the appearance of the "same location" for the service interface with network-based virtual IP addresses and hostnames. Administrative changes to service users, lookup services, or proxy agents are generally required.

These administrative changes significantly complicate the design, implementation, and test of service failover. A failure to make any of the required administrative changes during a failover will cause the service to become unavailable just as surely as a breakdown in bringing up the backup service provider. The more administrative changes that need to be made, the greater the chance an implementation mistake will cause the failover itself to fail. Since testing failover is an arduous and risky activity, there are advantages to keeping the administrative changes as simple as possible—or eliminating them altogether.

This situation gets even more complicated when asynchronous request-reply, subscription, and unsolicited notification patterns come into play. These patterns often require delivery interfaces on the service user—interfaces that the service provider, lookup service, or proxy need to know about. This makes the administrative problem even more complex, since location information now needs to be updated when service clients move.

## Message-Based Service Access

Messaging services provide an alternate means for service users and service providers to interact. We are all familiar with messaging services such as e-mail and instant messaging. At the systems level, messaging services such as the standards-based JMS (Java Messaging Service), IBM Websphere MQ, and TIBCO Rendezvous are in common use.

When using a messaging service, all communications between the parties is exchanged using an intermediary—the `Messaging Service Provider` (Figure 4–13). In contrast with the proxy approach, both sender and recipient are very much aware that they are employing a third party to send and receive messages.

Electronic messaging services differ from postal (physical mail) services in a subtle but significant way. Postal service mailboxes are physical entities that serve as the interfaces between the messaging service and its users. An address in physical mail systems designates the physical mailbox to which the letter or package is to be delivered. In other words, the address is location-specific.

In contrast, electronic messaging separates the concepts of destination and interface. The address now denotes a logical destination whose physical location is unknown to either the sender or recipient. The messaging service interface is no longer tied to a specific destination. Instead, the messaging service provides a generic interface for sending

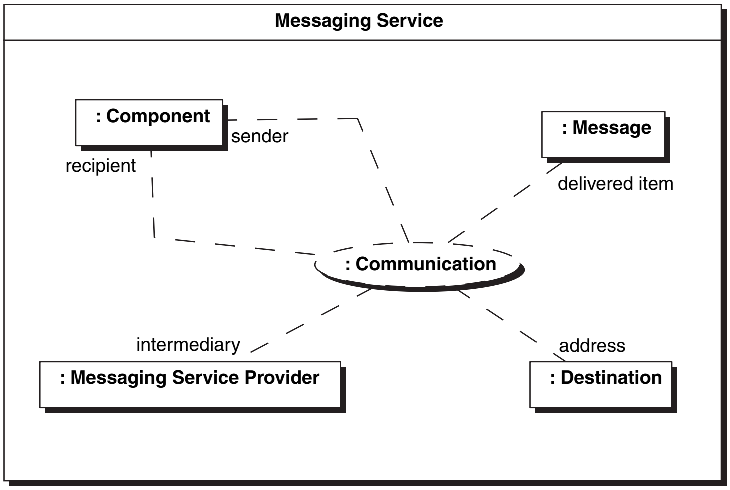

**Figure 4–13:** *Messaging Service*

and receiving messages regardless of the destination. This separation greatly simplifies the use of messaging services.

Abstract destinations provide location independence. The fact that the logical destination is no longer tied to the physical location of the recipient means that the recipient's location can change without impacting the sender—or the mail service. Thus, you can send an e-mail message to `jane.doe@messageservice.com` without having any idea where Jane Doe actually is. Furthermore, Jane Doe can retrieve her message from any machine that can access the messaging service, whether at home, at work, or at an Internet café in Istanbul!

The messaging service provides all the interfaces needed for communications (Figure 4–14). This simple shifting of interfaces to the messaging service greatly simplifies making deployment changes. None of the participants—sender, recipient, or messaging service—has any dependency at all on the location of the sender and recipient. In fact, the only location dependency that remains is that the sender and recipient must know the location of the messaging service interfaces. Consequently,

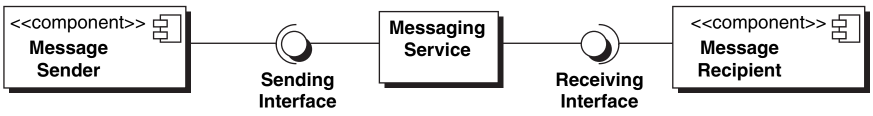

**Figure 4–14:** *Messaging Service Interfaces*

the sender and recipient can move freely without altering any configuration information—anywhere. As long as both can access the messaging service interfaces, they can communicate with each other.

There is another significant benefit that arises from the fact that the messaging service provides both the sending and receiving interfaces. Because the message recipient does not have to provide an interface to receive a message (it uses the messaging service's interface instead), it is easy for any component to receive a message. All that is required is an agreed-upon destination name. Thus it is easy for any component to be both a sender and a recipient. This communications flexibility enables the convenient implementation of the asynchronous request-reply, subscription, and unsolicited notification service interaction patterns. Therefore, the use of a messaging service provides flexibility in choosing whatever service interaction pattern seems most appropriate for the service being designed.

## Access Control

The primary reason for creating services is to make it easier to access their functionality, but at the same time you need to exercise some control over who is using the services. You may need to control who can perform certain operations. After all, you don't want some stranger making withdrawals from your bank account! You may need to limit the volume of usage. Services have finite capacity, and an unbounded demand may cause undesired performance degradation or outright failure. For a variety of reasons, you often need the ability to control access to your services.

Access control can be thought of as a set of *policies* applied at one or more *policy enforcement points*. An access control policy is a rule governing the access to the service. These rules may specify actions that individual parties are either allowed to take or must take under specific circumstances, or they may specify conditions under which access may be granted. One of the access control policies governing access to your bank account from an ATM is that you must provide a PIN (a required action on your part), and that PIN must match the one the bank has associated with your bank account (a condition that must be satisfied).

A policy enforcement point identifies the specific place in the architecture (usually an interface) at which the policy is enforced. In the ATM example, the enforcement point may be in the ATM (which would require the bank to provide the real PIN for the account to the ATM), or it may be at the bank when the transaction request is approved (a safer alternative).

Access control policies can cover many topics. Authentication (validating your credentials), authorization (establishing that the supplied credentials give you the right to access this particular account), and encryption (protecting your PIN from unauthorized viewing) are common topics for access control policies, but there are many others as well. Digital signing of message contents, the non-repudiation of requests (making an undeniable record that a particular message was received), and the creation of an audit trail of service utilization are frequently found to be requirements in business processes.

## Policy Enforcement Points

While it may be easy to state a policy, finding the appropriate place to enforce the policy is often not as straightforward as you might like. The enforcement of a policy requires the interception of the request, and possibly the reply as well. Setting up the system to intercept the request and reply requires a change to the architecture.

There are a number of places at which access control policies might be enforced. The choice of where to locate the policy enforcement point and which participant will initiate the policy enforcement depends very much upon the technique used to access the service. When services are being directly accessed, the enforcement point for these policies is generally the service interface, and the required functionality for enforcement is generally provided by the service provider (Figure 4–15). Of course, the service provider will likely employ other supporting services to aid in this enforcement, but it is the service provider that is responsible for intercepting the requests (and possibly the replies as well) and ensuring that the policies are checked and enforced.

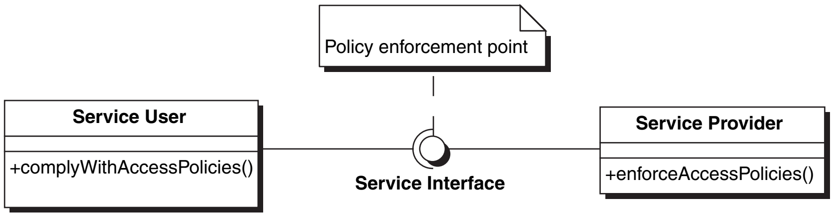

**Figure 4–15:** *Access Control Policy Enforcement in Direct Service Access*

While placing policy enforcement within the service provider may make logical sense, it does require a change to the service provider design. For some service providers, particularly purchased software applications, this may not be an option. Even when it is possible, changing the service provider design may be complex. From a software maintenance perspective, this can be an expensive option.

## Access Control with Proxies

Proxies provide an alternative location for policy enforcement—one that allows access control policies to be added without altering the service provider (Figure 4–16). This style of access control is commonly used when services are being provided via HTTP interfaces. In many cases such proxies are introduced into HTTP-based systems specifically to provide access control. Proxies used for this purpose are often referred to as *policy agents*.

Moving policy enforcement into the proxy provides a nice separation of concerns from a software engineering perspective. With this approach, policy enforcement can be added or modified without altering the underlying service provider. The policies and policy enforcement can be tested and managed independently of the service provider.

The proxy approach has an access control weakness: In the strictest sense, the original service interface remains unprotected. Any component that can gain access to this service interface is in a position to use it—or abuse it. Because of this, when the proxy pattern is employed, the actual service provider is generally located on a physically secure private network protected by a firewall. To guard against unauthorized access, the proxy is placed in a demilitarized zone (DMZ), which is separated from the public network by another firewall, resulting in the configuration shown in Figure 4–17.

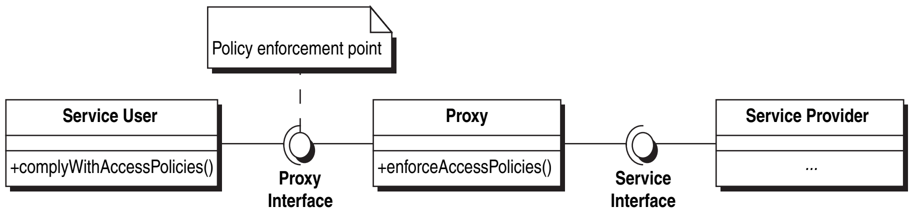

**Figure 4–16:** *Access Control Policy Enforcement with Proxy Access*

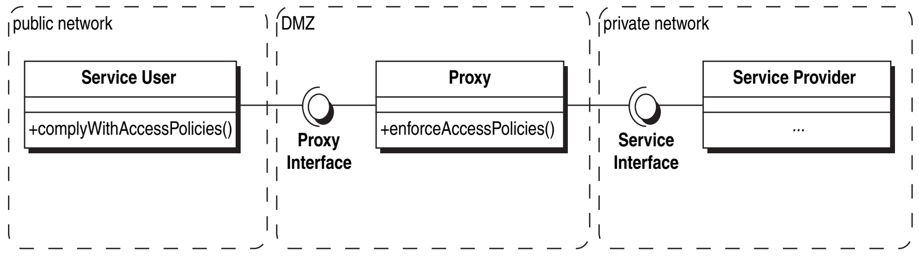

**Figure 4–17:** *Typical Proxy Deployment*

Beyond the access control weakness, there are other drawbacks to the proxy approach. The introduction of the proxy requires a configuration change on the part of the service users to redirect their requests to the proxy interface. The use of a proxy also introduces additional interprocess communications. In high-volume low-latency applications, the increase in latency caused by these additional communications can be an issue.

When asynchronous result delivery is required, there is additional work that must be done. To control access to the results delivery interface, either the existing proxy must be extended to also be proxy for the service user or a second proxy must be introduced. Since every proxy is dependent upon the location of the interfaces it is guarding, it becomes increasingly difficult to manage this approach as the number of services and service users increases. Every deployment change to every interface requires a proxy update.

## Access Control with a Mediation Service

The use of a messaging service for communications between the service user and the service provider presents yet another possible location for policy enforcement (Figure 4–18). Because of the extended functionality, this expanded service is more appropriately termed a *mediation service*. The mediation service contains within it both the messaging service and the policy enforcement.

As with the messaging service, the mediation service provides the interfaces for both the service user and service provider. Since the mediation service provides both interfaces, there are no dependencies on the location of either the service user or service provider. The symmetric nature of the sending and receiving interfaces makes it easy to support all of the basic service interaction patterns.

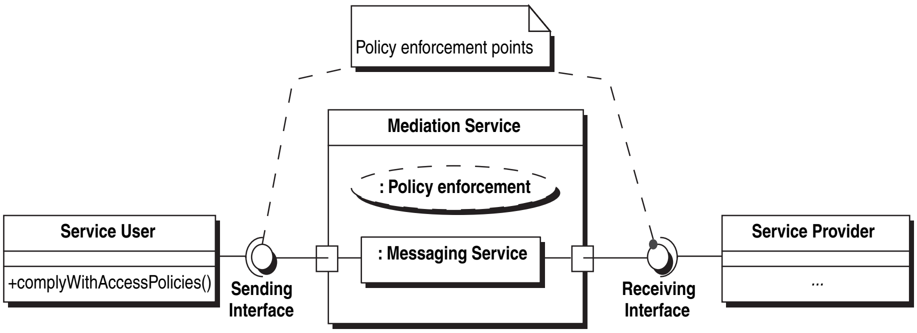

**Figure 4–18:** *Access Control Policy Enforcement with Message-Based Access*

The mediation service can enforce policies governing access to its own interfaces. It can authenticate the component trying to gain access and check that component's authorization to use the mediation service. Beyond this, it can check the component's authorization to send or receive from a particular messaging destination. It can require encrypted communications at both interfaces, perhaps using SSL. This ability to secure the connection to the service provider overcomes one of the shortcomings of the proxy approach, which cannot protect the service provider's interface from direct access.

## Service Request Routing

The discussion thus far has talked about both messaging and mediation services as if each were a single monolithic entity. But in reality any physical component has finite capacity limitations, so the implementation of the mediation service may require more than one component to spread the load. The service user may not be at the same location as the service provider, so the mediation service must be present at each location. There may also be more demand than can be satisfied by a single service provider, so the mediation service must be capable of distributing service requests among multiple service providers. For all of these reasons, the mediation service must be considered a logical service whose implementation can involve more than one component.

### Load Distribution

Perhaps the simplest routing task is simple load distribution (Figure 4–19). In this situation there are two or more service providers residing in the

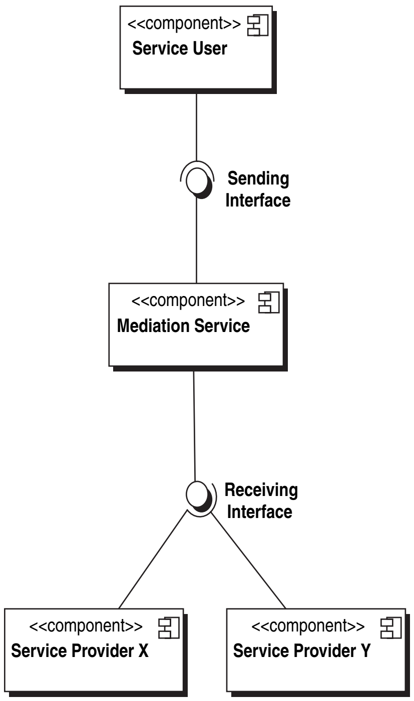

**Figure 4–19:** *Routing Requests for Load Distribution*

same location, and all of them are equivalent from a functional perspective. The routing problem is to determine which service provider should get each request. But even this can get complicated. What criteria do you use to select the service provider for the next request? Consider IP-redirectors, proxies that distribute incoming requests. Early redirectors used a round-robin approach, feeding each request to the next service provider in a sequence. Unfortunately, with this approach a "dead" service provider still gets its fair share of requests, which thus go unserved. More recent IP redirectors employ some form of liveness testing to avoid this kind of problem.

With the proxy-based approach, the proxy must make the decision as to which service provider should get each request. Making an assignment to an inoperable service provider runs the risk of the request going unserviced. Message-based mediation services avoid this problem. Message-based services hold the requests in a queue until a service provider retrieves them. Since the service provider takes the initiative in retrieving the request, the service provider must be operational to some extent. Dead service providers will not ask for more requests!

## Location-Based Routing

Another routing challenge arises when you have service users and service providers residing at different locations. A request originating at one location may need to be routed to a service provider at a different location (Figure 4–20). While this could be handled with a single mediation service component serving both locations, the result generally leads to some very inefficient communication patterns. Service users and providers at the remote location that need to intercommunicate must do so via the remote mediation service. From a communications perspective, it is far more efficient to have a local component of the mediation service and then let the mediation service take care of routing requests to other locations as required.

## Content-Based Routing

A third routing challenge arises when there are multiple service providers of the same service at different locations. Now the mediation service must determine which provider should get which request (Figure 4–21).

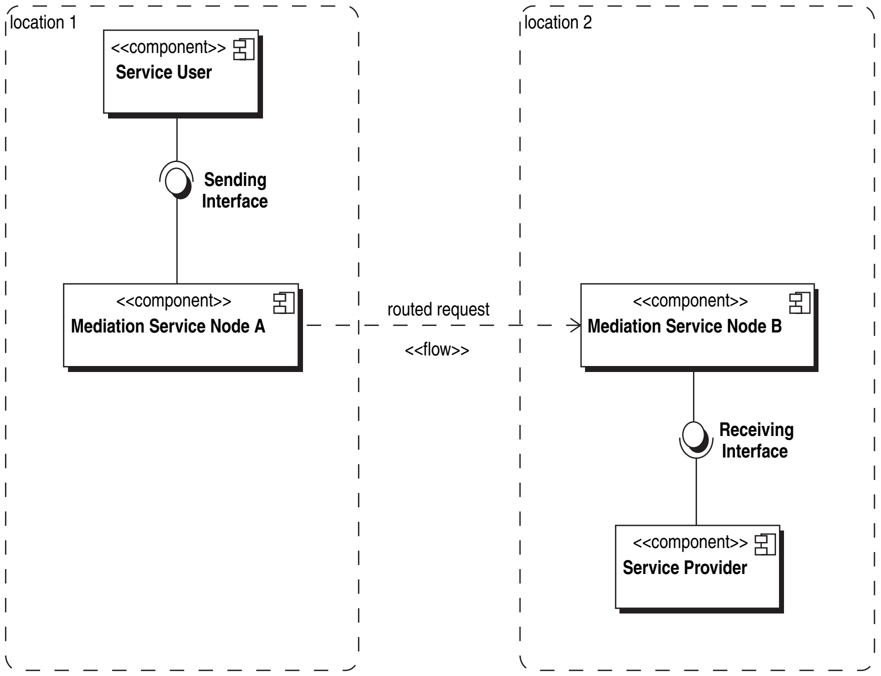

**Figure 4–20:** *Routed Service Requests*

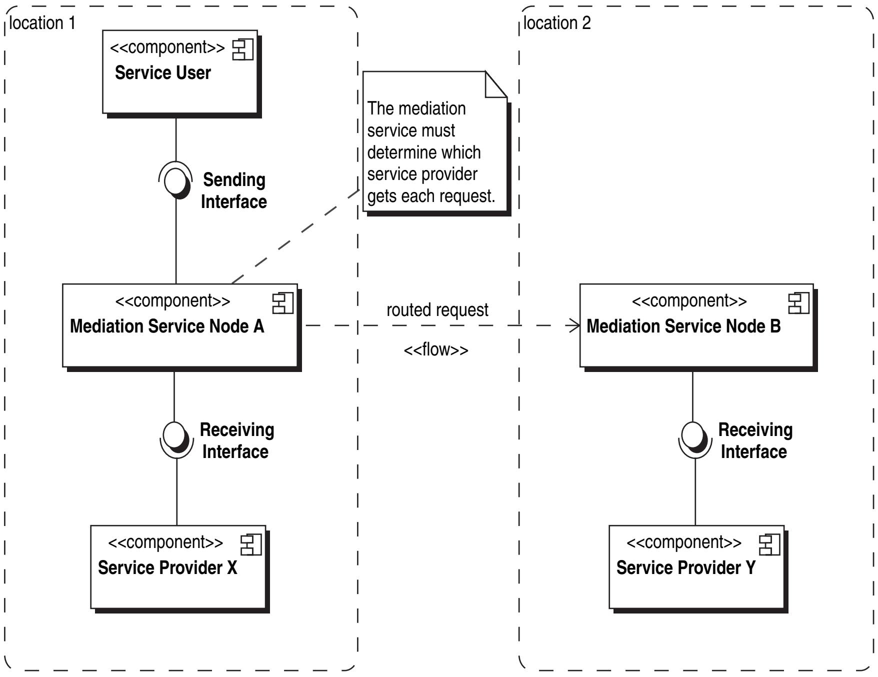

**Figure 4–21:** *Routed Requests Requiring Logic*

This routing can get complicated because different situations may call for different routing strategies.

The simplest situation is again the one in which the service providers are all identical. While you could leave the requests in a single queue and let both service providers pull from the same queue, this does not necessarily make optimum use of the bandwidth between the two sites. You may want to implement a strategy that weights requests for local servicing as long as the local service provider has the capacity. Only when the demand exceeds the local capacity and the other site has idle capacity would requests be routed to the remote service provider.

Life gets more complicated when the service providers are not all equivalent. You have your North American customer data in one database; your European, Middle Eastern, and African customers in a second database; and your Asian and Pacific-Rim customers in a third. Each database is located in the region it serves. For uniformity, you want to provide a single generic interface at each location through which all customer information can be accessed, regardless of where it is located. In this case, the mediation service must determine which service provider (database) can handle each request.

What makes this type of routing, commonly referred to as content-based routing, complex is that there is rarely enough information in the incoming request to do the routing. Determining the appropriate routing generally requires some form of reference data. When you place a phone call, the country code, area code, and telephone exchange are the keys to routing the call to the correct telephone exchange. However, to do the actual routing you need information that relates each combination of codes to the actual telephone exchange to which the call must be directed. Furthermore, the service needs to know how to extract the relevant data from either the message itself or its destination name.

Content-based routing makes the design of the mediation service dependent upon certain application-specific characteristics. The mediation service needs to know how to extract the key information from the request. It needs to know how to access the reference data and perhaps cache it for efficient access. It needs to know the correspondence between the reference data and the service providers. And it needs to know the rules for deciding which service provider gets which request.

The bottom line is that content-based routing is a service. As such, it needs to be treated with the same rigor as any other service. In particular, the stability of the interfaces to this service is a key concern. There is a tendency to think of content-based routing as being loosely driven by rules—rules that can easily be changed. But if changes to the rules require additional input data or reference data, interfaces change and there is a lot more work to be done. Thus any proposal for content-based routing must be evaluated in terms of its ability to support future as well as current needs.

## Service Composition

Services are only of value when they are employed in a business process. While some value is provided when a service is first employed as part of one business process, additional value—the added value that justifies making the functionality into a service—is provided when the service is employed in other business processes as well. The service makes it possible to create, modify, or extend other business processes faster and at a lower cost than would be possible without it. The ultimate goal of a service-oriented architecture is to enable the low-cost assembly of business processes from existing services, and the process of combining the services together is referred to as *composition*.

Composition is often discussed as a technique for combining services together to form a new higher-level service. Such a service is referred to as a *composite service.* However, the idea of composing services together is distinct from the idea of turning the resulting composite into a new service. The majority of composites you are likely to encounter are not services—they are business processes. The core value provided by SOA is the ability to quickly and inexpensively create these composites. Turning a composite into a service is gravy.

## Hard-Wired Composition

The simplest way to compose services is to hard wire them together so that the result of one service becomes the input to the next (Figure 4–22). This type of composition is termed hard-wired because there is no explicit embodiment of the overall process. To determine what this overall process actually is, you have to trace the sequence of interactions between the participants. While this type of composition may not strike you as being particularly service-oriented, it is the most commonly found composition technique in use today, although the participants in such compositions are often not designed as services.

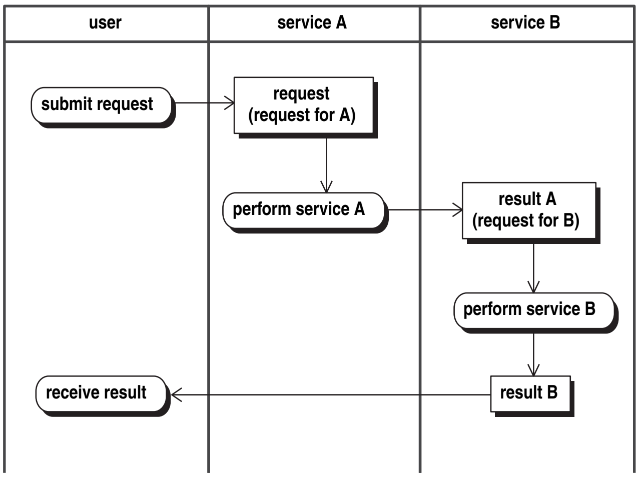

**Figure 4–22:** *Hard-Wired Composition*

In hard-wired composition, the result produced by one service may not be in the proper form to serve as the input to the next, or the component producing the result may not be configured to direct it to the next service. In such cases, intermediate components can be used to transform the results and deliver them to the appropriate destinations. Part IV of this book explores these components and their attendant design issues.

## Nested Composition

Service-oriented architectures lead to a very natural style, modularizing functionality into request-response service operations and then building composites that invoke these services (Figure 4–23). This compositional style is an excellent first thought in terms of organizing functionality, but it must be examined from a performance perspective

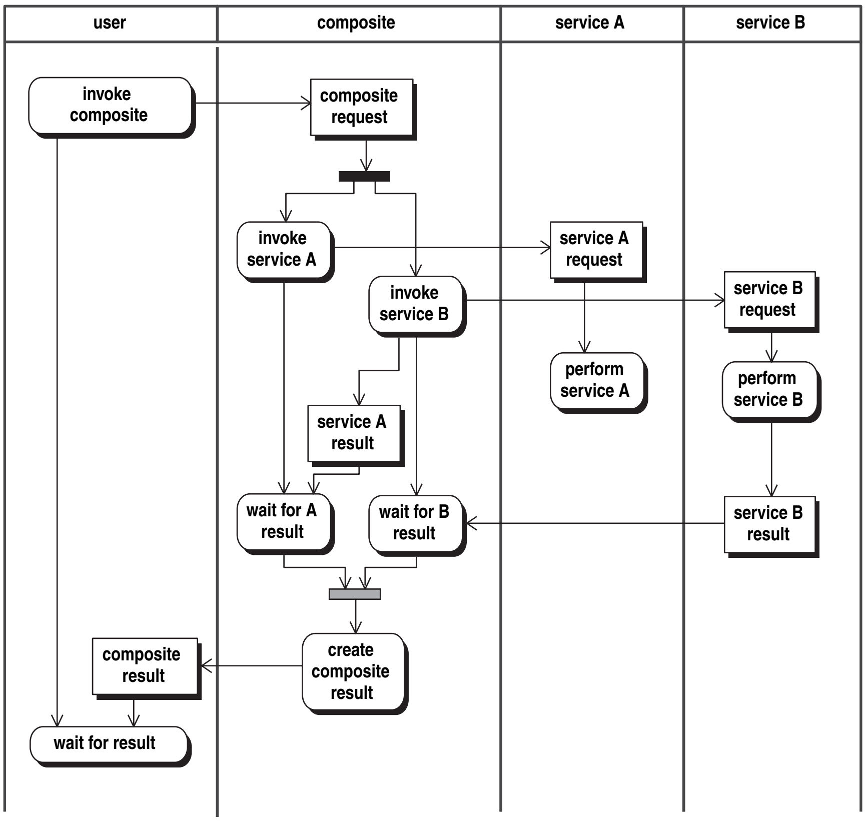

**Figure 4–23:** *Nested Composition*

to determine whether the modularization is adequate for the task. The performance examination must explore both response latency and the downstream service capacity limitations.

When requests are nested in this manner, the accumulated latency resulting from the nested request-response calls may result in unacceptable overall response time from the user's perspective. For each service call, there is a communication delay in each direction. On top of this, there is the time it takes the service provider to respond to the request, and the additional time it takes the service provider to perform the work. These delays add to the time required for the user to interact with the composite and the time it takes the composite to do its own work. When the underlying services are themselves composites, additional latencies are introduced. Depending upon the needs of the user, the accumulated composite latency may become unacceptable.

You also need to consider the load being placed on the lower-level service. The service may not have been designed to handle the volume of requests coming from the composite. One hotelier with multiple properties in the same location decided to change the service its customers use to locate hotel rooms in a city. Instead of having the customer first select from a list of hotels and then check room availability in that hotel, they implemented a new composite service that checked the availability of all hotels in the city for each customer request. The impact was that the volume of queries increased dramatically on all the hotels—by a factor of eight! During peak periods (the most important times for hotel bookings), the resulting demand exceeded the capacity of the individual hotel systems. As a result, the majority of the queries timed out and provided no response. Instead of delivering a new and improved service to customers, many room availability queries went unanswered.

Performance evaluation for any use of a service should be a routine part of the design process. Chapter 39 shows how to perform this type of analysis.

## Cached Composites

The nested composition performance lesson is one that was first learned when businesses first started adding customer-facing web self-service front-ends. These web sites allowed customers to check the status of their orders and shipments. However, the underlying back-office production systems that processed these orders were never designed to handle this kind of dynamic query load. Early web site implementations that simply queried the back-office production systems not only performed poorly, but their query load often had an adverse impact on the back-office system's ability to perform its work.

To cope with this type of situation, an architectural style evolved in which status information is extracted from the production system and cached for convenient access by the web application server. In service-oriented terms, the style of interaction with the underlying service changed from a request-reply interaction to a subscription interaction (Figure 4–24).

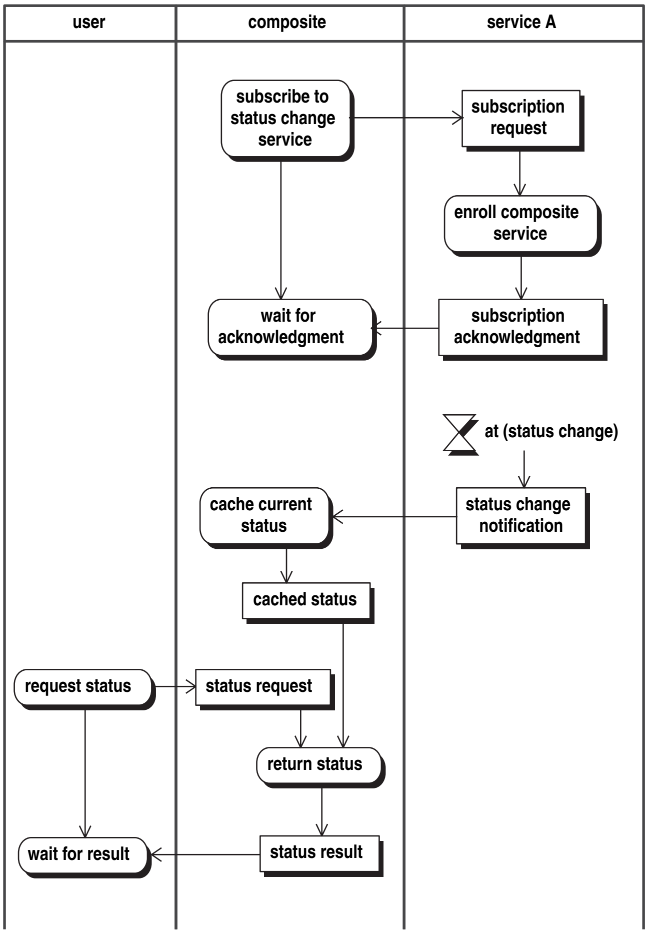

**Figure 4–24:** *Cached Composite*

Note that this architectural change impacted not only the composite, but the underlying service as well. Interaction with the underlying service changed from a request-reply to a subscription, and the composite changed from a nested composition to a cached composition. Once again, you can see the importance of understanding the intended utilization of a service—particularly the volume of activity and the required response times. This understanding will impact the architecture of both the composite and the service, and will deeply influence the operation interfaces.

## Locating Services

You can't use a service unless you are aware of its existence. A significant challenge in SOA is helping architects and designers locate the services that are appropriate for their needs. UDDI registries provide a mechanism for sharing some information about services (primarily interface details and information about service providers and users), but there is a wealth of additional information required to support the use of services, including (but not limited to):

- Service requirements—how the service specification is derived from its intended utilization
- Service specification—the interface (WSDL), including operations and data structures, the information managed by the service, the externally observable behavior, the characterization of intended utilization, and supported SLAs
- One-line description of the service—for quick weeding out of services
- Abstract (paragraph)—for a second-pass weeding
- Service user's guide—everything a user needs to know to use the service, from an introduction to its capabilities and intended utilization to the details of best practices in employing the service
- Service operation and maintenance guide
- Service operational support procedures

This information must be created, organized, archived, and made accessible to the community of service users. Services need to be categorized and indexed for easy access (some of which can be done in the UDDI registry), and the one-line descriptions and abstracts should be searchable as well.

## Enterprise Architecture for Services

Except in the most unusual of circumstances, you will not be constructing your SOA in one mammoth project. Instead, your enterprise will evolve its existing architecture in a series of projects. However, to reap the benefits of SOA, you must make sure that design decisions are made consistently from project to project. The role of maintaining this consistency is typically given to the enterprise architecture group. With respect to services, this group's responsibilities are:

- Selecting the service interface standards to be followed and selecting the supporting infrastructure to be used
- Defining service interaction patterns and their preferred technology implementations
- Defining the criteria to be used in determining when standardized data representations (common data models) should be employed in operation interfaces
- Defining the selection criteria for proposed services and the procedures for validating their appropriateness
- Defining the preferred architectural styles for implementing services and the criteria to be used in selecting a style
- Defining the service mediation architecture and selecting the supporting infrastructure to be used
- Establishing the preferred design patterns for content-based routing and the criteria to be used in selecting a pattern
- Establishing the capacity planning procedures for services and ensuring that these procedures are followed

As an enterprise architect, you should be aware that it is difficult to formulate an efficient and practical set of guidelines without a bit of trial and error. You must make every effort to observe the guidelines and procedures you have defined being put into practice. Are the guidelines easy to follow, or do they constantly require interpretation? Interpretation opens the door for variation and therefore inconsistency from project to project. Such variation may work against your SOA objectives. Observe the level of effort required for project teams to comply with the guidelines and weigh this effort against the benefit you expect from the guideline. Is the effort justified? Be particularly vigilant for signs of excessive administrative complexity, making records that are never used or whose accuracy is never validated. Such complexity not only increases the level of effort, but it can also be a significant source of error.

## Summary

When incorporating services into a business process, there are a number of ways in which a service user and service provider may interact. The synchronous request-reply style is perhaps the most common, but many business processes require asynchronous responses, subscription services, and unsolicited notifications as well. Determining the appropriate style of interaction requires an understanding of how the service will be employed in business processes.

Your choice of service access mechanisms directly impacts the ease with which you can access services, your ability to provide the different interaction patterns that may be required by the business processes, and the amount of work required to change the location of both service users and service providers. While direct or proxy-mediated access to services provides good support for synchronous request-reply, it makes implementation of other interaction patterns complex. Direct and proxy-mediated approaches are always dependent on the location of the service provider, and the use of interaction patterns other than synchronous request-reply makes them dependent upon the location of the service user as well.

Message-based service access provides more flexibility for interacting with services. It has no dependency on the location of the service users and providers, and it provides simple support for all of the service interaction patterns. It provides the most flexible service access.

Many services require some form of access control to manage who has access to which service. Access control can be implemented by the service provider, but this approach requires the modification of the service provider and can be expensive. Alternatively, access control can be implemented by either a proxy or a mediation service (an extended messaging service). Once again, the proxy approach is convenient for synchronous request-reply, but awkward for the other interaction patterns. The mediation service approach works well for all interaction patterns.

Accessing services may require the routing of service requests. The need for routing may arise from the need to distribute load across multiple service providers, the need to route specific requests to specific providers, or the placement of users and service providers at different geographic locations. Routing may also require some introspection into the message, the access of reference data, and the evaluation of routing rules. A mediation service provides a convenient architectural home for this functionality. Geographic routing generally requires the presence of mediation service components at each geographic location.

Services provide value when they are actually utilized in business processes. Combining services to form business processes is referred to as composition. Optionally, the resulting composite may, itself, be offered as a higher-level service.

The architecture of composites must be carefully evaluated for performance. Nested synchronous request-reply interactions can lead to excessive latency and unacceptable loads on back-end services. In such cases, architectural alternatives such as caching information in the composite may be more appropriate. Choosing such alternatives may impact the architecture of the back-end service as well as that of the composite, transforming a synchronous request-reply interaction between them into a subscription interaction. Thus, to achieve the service interface stability that is required to justify the service development cost, potential service usages must be thoroughly explored before the service is specified and implemented.

Achieving the benefits of a service-oriented architecture requires consistent decision making across many projects. The enterprise architecture group plays a key role not only in establishing standards and best practices for building services but also in ensuring that these guidelines are practical for routine day-to-day usage by project teams.

## Key Service Utilization Questions

1. Does your SOA infrastructure support the common service interaction patterns: synchronous request-reply, asynchronous request-reply, subscription, and unsolicited notification?
2. How does your SOA infrastructure control access to services? Does the access control work for the four common service interaction patterns?
3. How does your SOA infrastructure route requests to the service provider(s)? Does it support the routing of requests between geographic locations? Does it support content-based routing?

4. Who reviews the proposed architecture of new services from a performance perspective?
5. Has your enterprise architecture group established standards and best practices for the development of services and for the supporting infrastructure? Has the group followed up to determine whether these standards and best practices are practical and being used?
6. Do you have an archive for service-related information? Does it contain one-line descriptions, abstracts, and user documentation? Is it searchable?

## Suggested Reading

W3C (World Wide Web Consortium). March 2004. "Web Services Description Language (WSDL) Version 2.0, Part 2: Message Exchange Patterns," W3C Working Draft 26. www.w3.org/TR/2004/WD-wsdl20-patterns-20040326.

# Index