erl_ch03.fm Page 47 Wednesday, March 17, 2004 4:55 PM

CHAPTER

3

Introduction to Web services technologies

Before we delve into the concepts and technology behind Web services, let's complete the timeline we began at the beginning of the previous chapter. In 2000, the W3C accepted a submission for the Simple Object Access Protocol (SOAP). This XML-based messaging format established a transmission framework for inter-application (or inter-service) communication via HTTP. As a vendor-neutral technology, SOAP provided an attractive alternative to traditional proprietary protocols, such as CORBA and DCOM.

During the following year, the W3C published the WSDL specification. Another implementation of XML, this standard supplied a language for describing the interface of Web services. Further supplemented by the Universal Description, Discovery, and Integration (UDDI) specification that provided a standard mechanism for the dynamic discovery of service descriptions, the first generation of the Web services platform had been established. Figure 3.1 illustrates, on a high level, the relationship between these standards.

Since then, Web services have been adopted by vendors and manufacturers at a remarkable pace. Industry-wide support furthered the popularity and importance of this platform and of service-oriented design principles. This led to the creation of a second generation of Web services specifications (discussed in Chapter 4).

3.1 Web services and the service-oriented architecture (SOA)

3.1.1 Understanding services

The concept of services within an application has been around for a while. Services, much like components, are intended to be independent building blocks that collectively represent an application environment. Unlike traditional components, though, services have a number of unique characteristics that allow them to participate as part of a service-oriented architecture.

One of these qualities is complete autonomy from other services. This means that each service is responsible for its own domain, which typically translates into limiting its scope to a specific business function (or a group of related functions).

This design approach results in the creation of isolated units of business functionality loosely bound together by a common compliance to a standard communications

erl_ch03.fm Page 49 Wednesday, March 17, 2004 4:55 PM

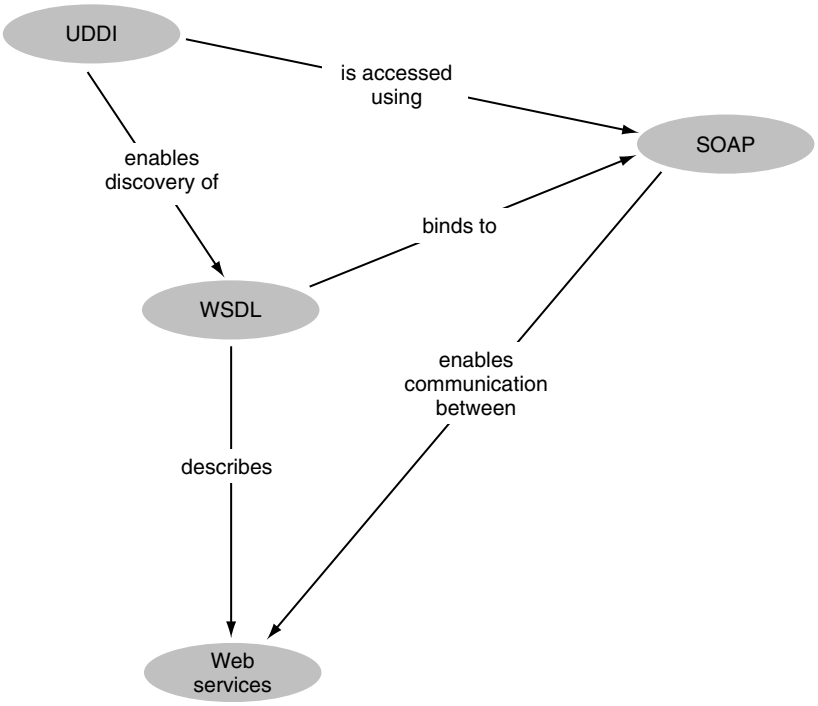

Figure 3.1
The relationship between first-generation specifications.

framework. Due to the independence that services enjoy within this framework, the programming logic they encapsulate does not need to comply to any one platform or technology set.

3.1.2 XML Web services

The most widely accepted and successful type of service is the *XML Web service* (from hereon referred to as *Web service* or, simply, *service*).

This type of service has two fundamental requirements:

- it communicates via Internet protocols (most commonly HTTP)
- it sends and receives data formatted as XML documents

That's pretty much it. You can write a simple ASP or JSP script that resides on a Web server sending and receiving XML formatted messages via HTTP, and you can go out and get yourself an "I'm Service-Oriented" T-shirt.

erl_ch03.fm Page 50 Wednesday, March 17, 2004 4:55 PM

Broad acceptance of the Web service design model has resulted in the emergence of a set of supplementary technologies that have become de facto standards. An industry standard Web service, therefore, generally is expected to:

- provide a service description that, at minimum, consists of a WSDL document
- be capable of transporting XML documents using SOAP over HTTP

These technologies do not alter the core functionality of a Web service as much as they do its ability to represent itself and communicate in a standard way. Many of the architectural conventions expressed in this chapter assume that SOAP and WSDL are part of the described Web services framework.

Additionally, it is common for Web services to:

- be able to act as both the requestor and provider of a service
- be registered with a discovery agent through which they can be located

In a typical conversation with a Web service, the client initiating the request is a Web service as well. As shown in Figure 3.2, any interface exposed by this "client service" also qualifies it as a service from which other services can request information. Therefore, Web services do not fit into the classic client-server model. Instead, they tend to establish a peer-to-peer system, where each service can play the role of client or server.

NOTE

The tutorials in this part of the book cover technology only. SOA design principles, as they apply to business and architecture modeling, are covered in detail in Chapter 14. If you are interested in supplementing the technical knowledge provided here with service-oriented design theory, I highly recommend you read through the "SOA modeling basics" section.

3.1.3 Service-oriented architecture (SOA)

As previously mentioned, it is not too much trouble to append an application with a few Web services. This limited integration may be appropriate as a learning experience, or to supplement an existing application architecture with a service-based piece of functionality that meets a specific project requirement. It does not, however, establish a service-oriented architecture.

There is a distinct difference between:

erl_ch03.fm Page 51 Wednesday, March 17, 2004 4:55 PM

- an application that uses Web services
- an application based on a service-oriented architecture

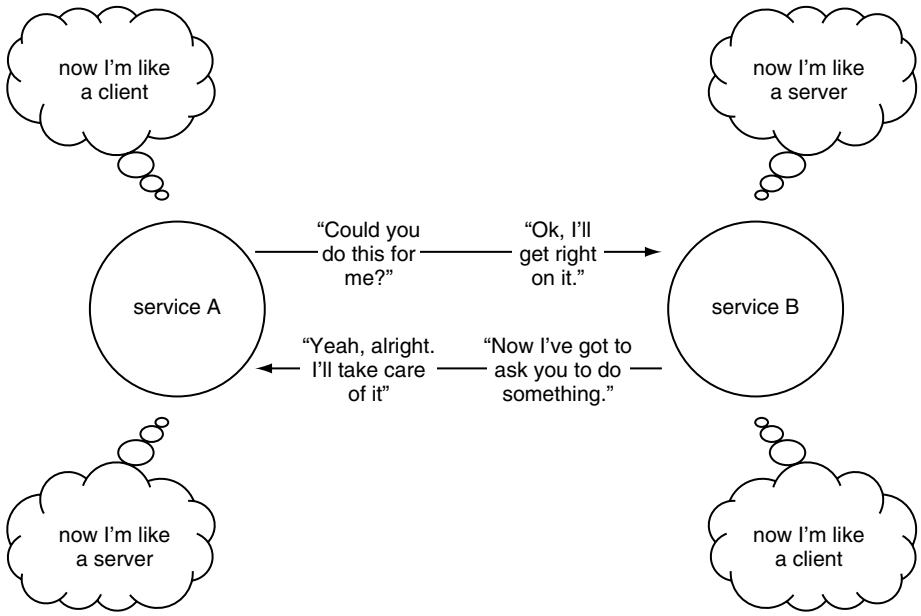

Figure 3.2
Web services swapping roles during a conversation.

An SOA is a design model with a deeply rooted concept of encapsulating application logic within services that interact via a common communications protocol. When Web services are used to establish this communications framework, they basically represent a Web-based implementation of an SOA.

The resulting architecture essentially establishes a design paradigm within which Web services are a key building block. This means that when migrating your application architecture to an SOA, you are committing yourself to Web services design principles and the accompanying technologies as core parts of your technical environment.

An SOA based on XML Web services builds upon established XML technology layers, with a focus on exposing existing application logic as loosely coupled services. In support of this model, an SOA promotes the use of a discovery mechanism for services via a service broker or discovery agent.

Figure 3.3 shows how an SOA alters the existing multi-tier architecture by introducing a logical layer that, through the use of standard programmatic interfaces (provided by Web services), establishes a common point of integration.

erl_ch03.fm Page 52 Wednesday, March 17, 2004 4:55 PM

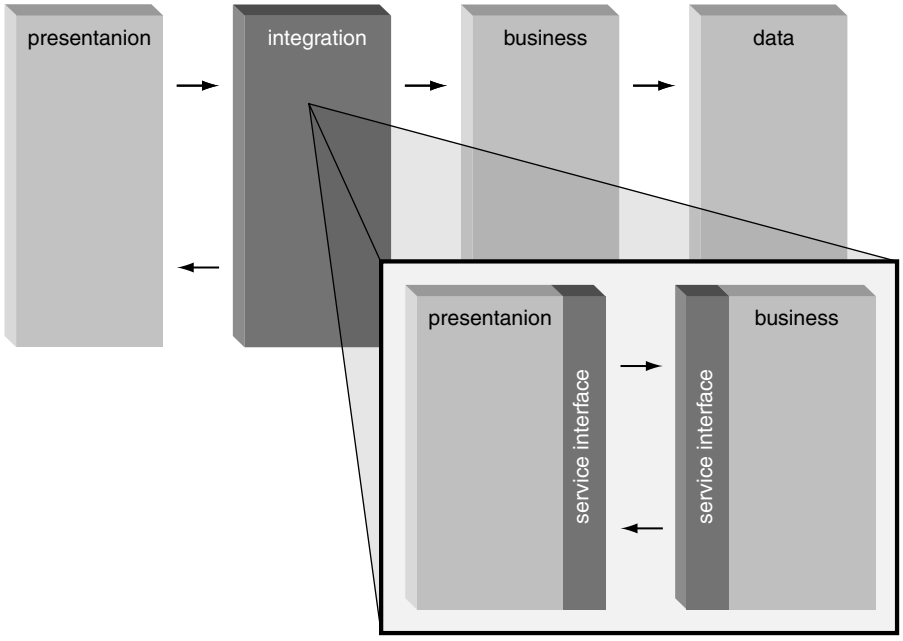

Figure 3.3
A logical representation of a service-oriented architecture (for a single multi-tier application).

This *service integration layer* forms the basis of a new model that can extend beyond the scope of a single application to unify disparate legacy platforms into an openly interoperable environment. When Web services are used for cross-application integration (as in Figure 3.4's logical service-oriented integration architecture), they establish themselves as part of the enterprise infrastructure.

It is important to understand the increased design complexities introduced by enterprise SOAs. Even more so than in n-tier environments, application designers need to appreciate fully how the introduction of services will affect existing data and business models, especially if current or future EAI initiatives need to comply to a service-oriented model.

As the utilization of services diversifies, the significance of security and scalability requirements are amplified. Well-designed service-oriented environments will attempt to address these challenges with adequate infrastructure, rather than custom, application-specific solutions.

For detailed information about service-oriented architecture design principles, and how SOAs can lead to the evolution of a service-oriented enterprise, see Chapter 14.

erl_ch03.fm Page 53 Wednesday, March 17, 2004 4:55 PM

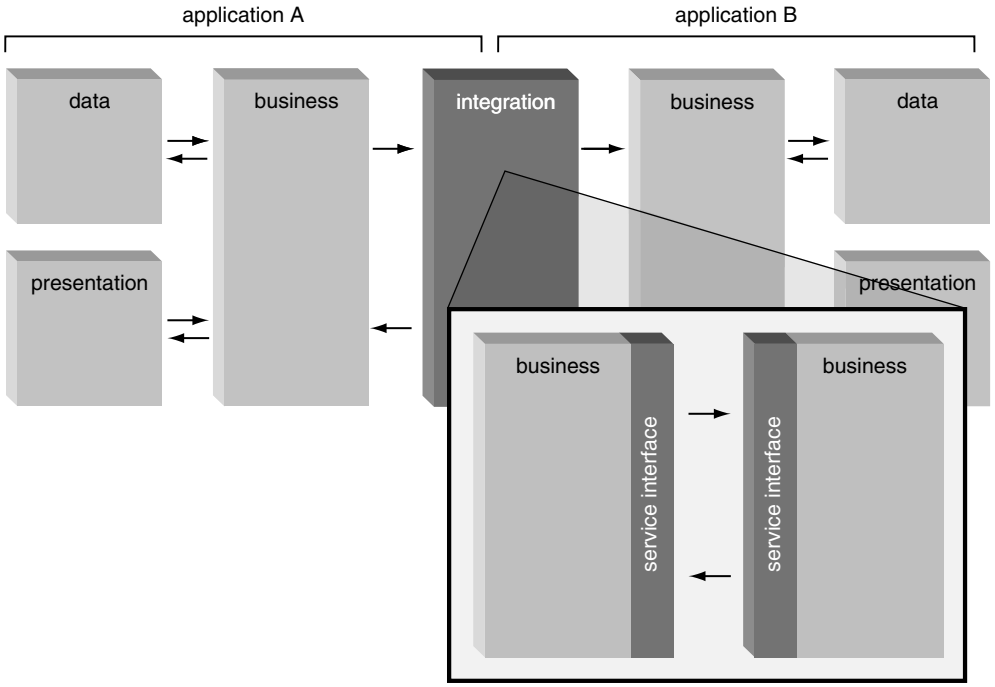

Figure 3.4
A logical representation of a service-oriented integration architecture.

SUMMARY OF KEY POINTS

- XML Web services are a Web-based implementation of service-oriented principles.
- The service-oriented architecture introduces a new logical layer within the distributed computing platform.
- The service integration layer establishes a common point of integration within application tiers and across application boundaries.

NOTE

The roles and scenarios illustrated in the next two sections are limited to the involvement of Web services only. The underlying SOAP messaging framework is explained separately, later in this chapter.

3.1.4 Web service roles

Services can assume different roles when involved in various interaction scenarios. Depending on the context with which it's viewed, as well as the state of the currently

erl_ch03.fm Page 54 Wednesday, March 17, 2004 4:55 PM

running task, the same Web service can change roles or be assigned multiple roles at the same time.

Service provider

When acting as a *service provider*, a Web service exposes a public interface through which it can be invoked by requestors of the service. A service provider promotes this interface by publishing a service description. In a client-server model, the service provider is comparable to the server (Figure 3.5).

NOTE

The term "service provider" can also be used to describe the organization or environment hosting (providing) the Web service.

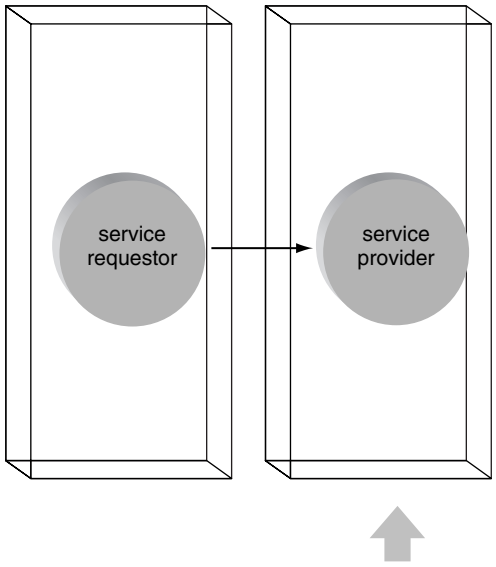

Figure 3.5
A service provider receiving a request from a service requestor.

A service provider can also act as a service requestor. For example, a Web service can play the role of service provider when a service requestor asks it to perform a function. It can then act as a service requestor when it later contacts the original service requestor (now acting as a service provider) to ask for status information.

erl_ch03.fm Page 55 Wednesday, March 17, 2004 4:55 PM

Service requestor

A *service requestor* is the sender of a Web service message or the software program requesting a specific Web service. As shown in Figure 3.6, the service requestor is comparable to the client within the standard client-server model.

NOTE

Service requestors are sometimes referred to as *service consumers.*

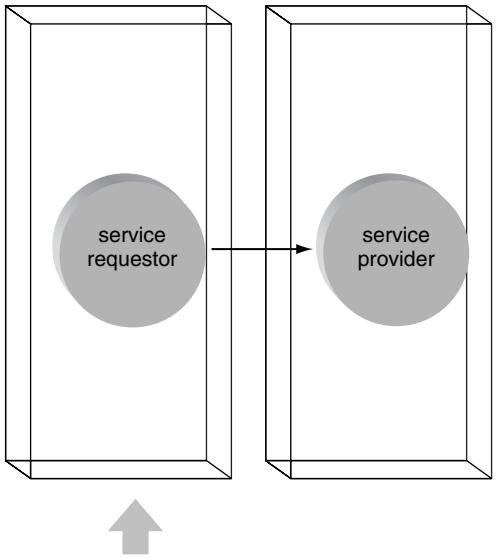

Figure 3.6
A service requestor initiating a request to a service provider.

A service requestor can also be a service provider. For instance, in a request and response pattern, the initiating Web service first acts as a service requestor when initially requesting information from the service provider. The same Web service then plays the role of a service provider when responding to the original request.

Intermediary

The role of *intermediary* is assumed by the Web service when it receives a message from a service requestor, and then forwards the message to a service provider. Figure 3.7 explains how, during the time that an intermediary processes a message, it too can act as a service provider (receiving the message) and as a service requestor (sending the message forward).

erl_ch03.fm Page 56 Wednesday, March 17, 2004 4:55 PM

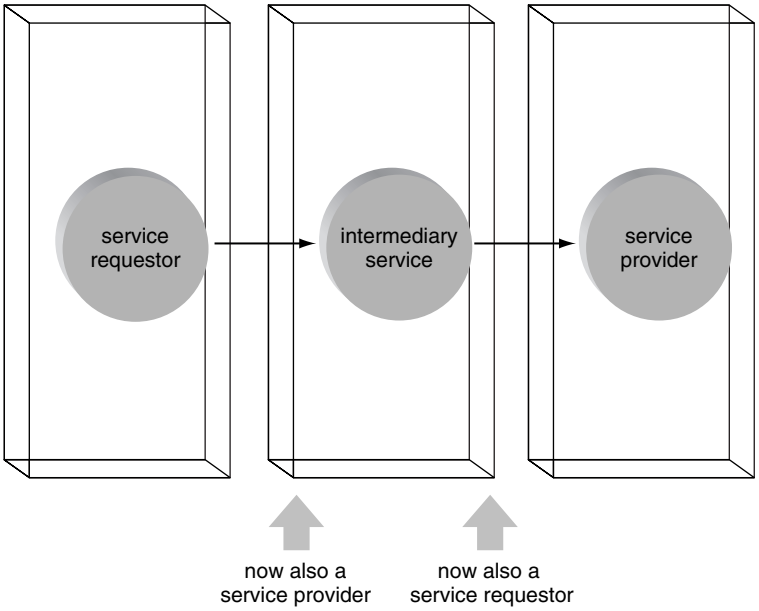

Figure 3.7
An intermediary service transitioning through two roles while relaying a message.

Intermediaries can exist in many different forms. Some are passive, and simply relay or route messages, whereas others actively process a message before passing it on. Typically, intermediaries are allowed only to process and modify the message header. To preserve the integrity of a message, its data should not be altered. (Intermediaries are discussed in more detail in Chapter 9.)

Initial sender
As the Web services responsible for initiating the transmission of a message, *initial senders* also can be considered service requestors (Figure 3.8). This term exists to help differentiate the first Web service to send a message, from intermediaries also classified as service requestors.

Ultimate receiver
The last Web service to receive a message is the *ultimate receiver*. As shown in Figure 3.9, these services represent the final destination of a message, and also can be considered service providers.

3.1.5 Web service interaction

When messages are passed between two or more Web services, a variety of interaction scenarios can be played out. Following are common terms used to identify and label these scenarios.

erl_ch03.fm Page 57 Wednesday, March 17, 2004 4:55 PM

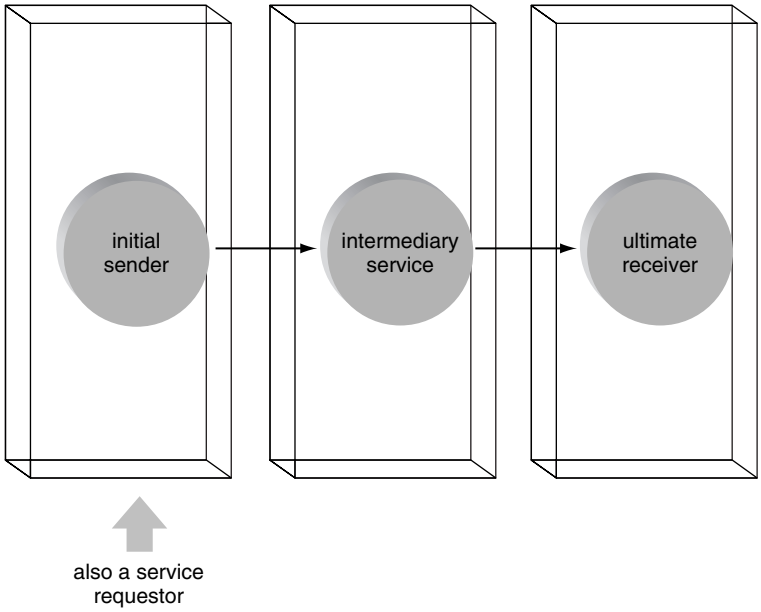

Figure 3.8
The first Web service is identified as the initial sender.

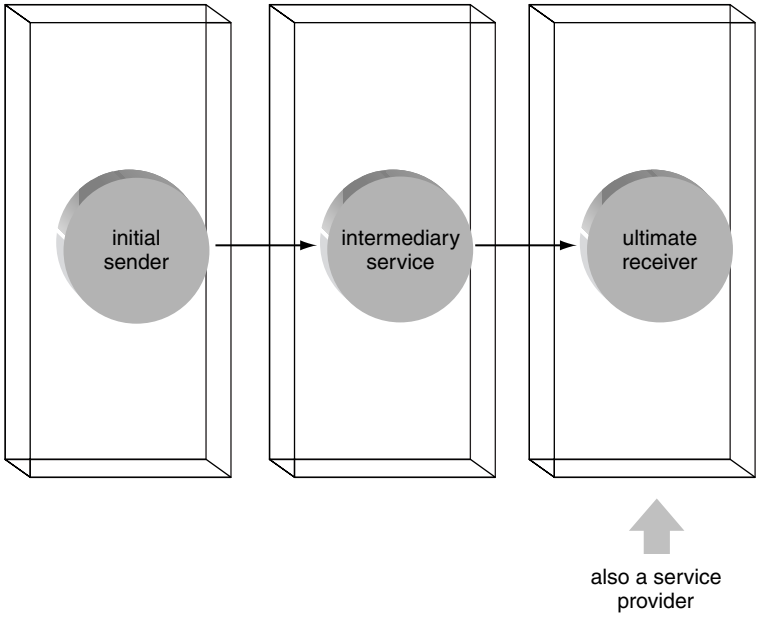

Figure 3.9
A service acting as the ultimate receiver.

erl_ch03.fm Page 58 Wednesday, March 17, 2004 4:55 PM

Message path

The route along which a message travels is the *message path*. It must consist of one initial sender, one ultimate receiver, and can contain zero or more intermediaries. Figure 3.10 illustrates a simple message path.

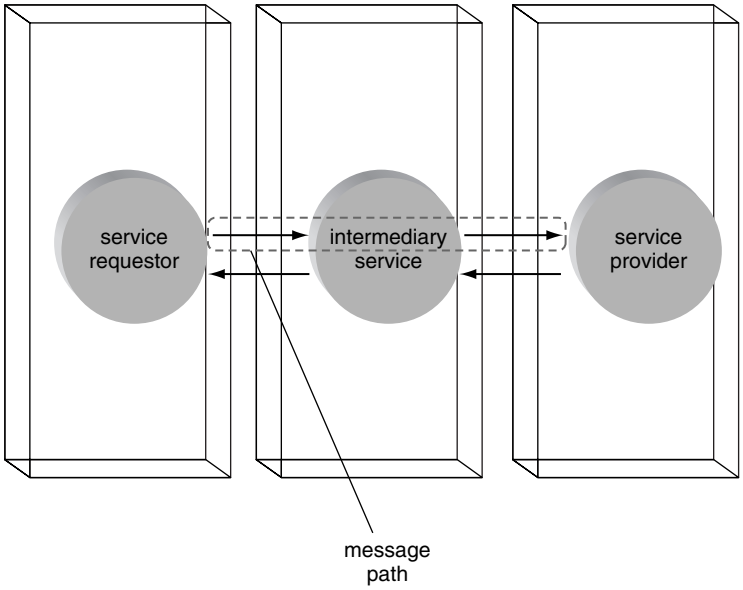

Figure 3.10
A message path consisting of three Web services.

The actual transmission path traveled by a message can be dynamically determined by routing intermediaries. Routing logic may be invoked in response to load balancing requirements, or it can be based on message characteristics and other variables read and processed by the intermediary at runtime. Figure 3.11 explains how a message is sent via one of two possible message paths, as determined by a routing intermediary.

Message exchange pattern

Services that interact within a service-oriented environment typically participate in one of a number of predefined *message exchange patterns*.

Typical patterns include:

- request and response (see Figure 3.12)
- publish and subscribe
- fire and forget (one-to-one)
- fire and forget (one-to-many or broadcast)

erl_ch03.fm Page 59 Wednesday, March 17, 2004 4:55 PM

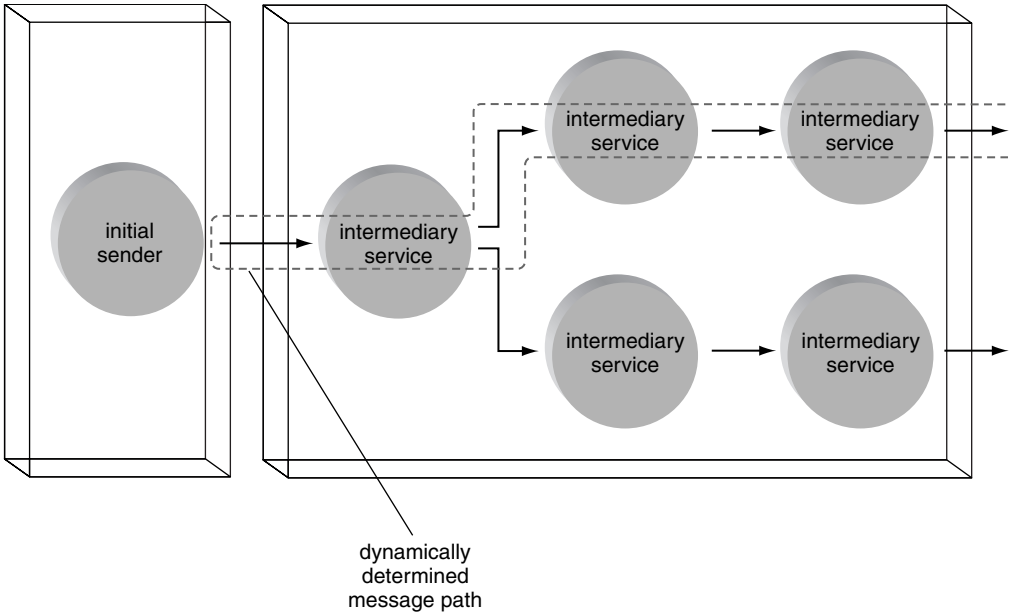

Figure 3.11
A message determined dynamically by a routing intermediary.

The request and response pattern is more common when simulating synchronous data exchange. The remaining patterns are used primarily to facilitate asynchronous data transfer.

Correlation

Correlation is a technique used to match messages sent along different message paths. As illustrated in Figure 3.13, it is commonly employed in a request and response message exchange pattern, where the response message needs to be associated to the original message initiating the request.

Embedding synchronized ID values within the related messages is a frequently used technique to achieve correlation.

Choreography

Rules that govern behavioral characteristics relating to how a group of Web services interact can be applied as a *choreography*. These rules include the sequence in which Web services can be invoked, conditions that apply to this sequence being carried out, and a usage pattern that further defines allowed interaction scenarios. The scope of a choreography is typically tied to that of an activity or task (see Figure 3.14 for an example).

erl_ch03.fm Page 60 Wednesday, March 17, 2004 4:55 PM

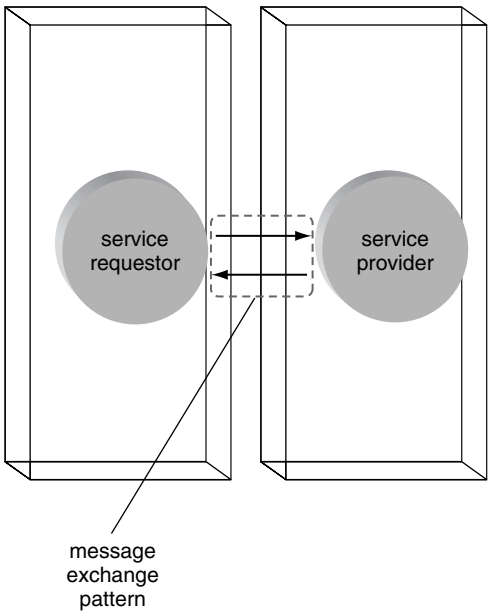

Figure 3.12
Example of a request and response message exchange pattern.

Activity

Message exchange patterns form the basis for service *activities* (also known as *tasks*). An activity consists of a group of Web services that interact and collaborate to perform a function or a logical group of functions. Figure 3.15 shows a simple service activity.

The difference between a choreography and an activity is that the activity is generally associated with a specific application function, such as the execution of a business task.

3.1.6 Service models

Depending on the extent to which they are utilized, Web services can introduce a great deal of standardization on a number of levels, including:

- application architecture
- enterprise infrastructure
- global data exchange

Despite their contribution to establishing a consistent framework for information interchange, Web services themselves do not come in standard shapes or sizes. A

erl_ch03.fm Page 61 Wednesday, March 17, 2004 4:55 PM

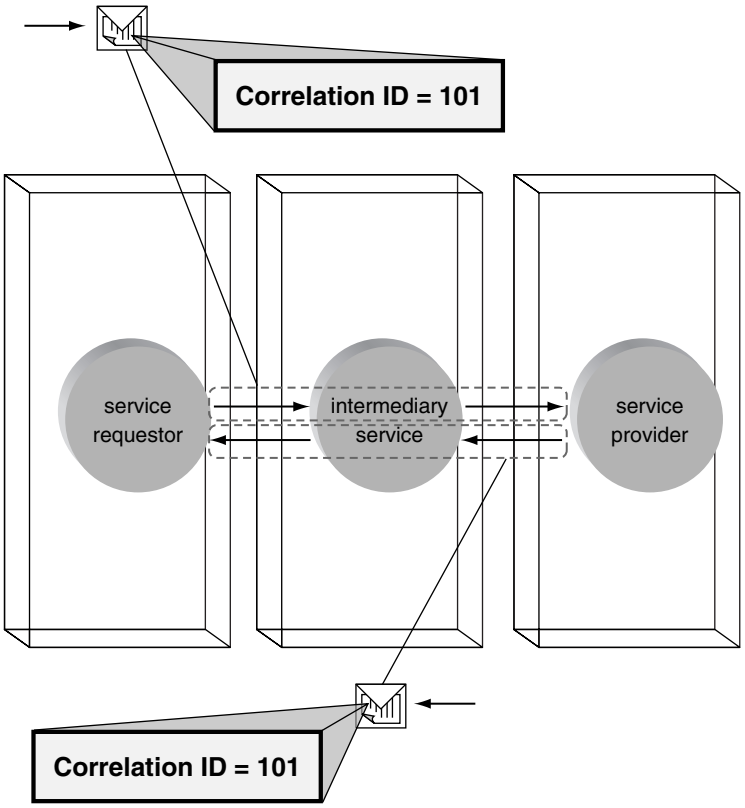

Figure 3.13
Correlation used in a request and response message exchange pattern.

NOTE

A choreography is similar in concept, but still different from orchestration. Orchestration is an implementation of a choreography within the context of a workflow or business process. To learn more about orchestration, read through Chapter 10.

service can be based on one of a number of design models, each with its own role and function.

This book documents a number of the common XWIF design models for Web services (listed in Table 3.1). Consider these a starting point, and feel free to customize them to whatever extent they assist in meeting your requirements. As the overall acceptance of service-oriented designs increases, and as Web services and their associated technology set become more integrated into the IT mainstream, this list is sure to evolve.

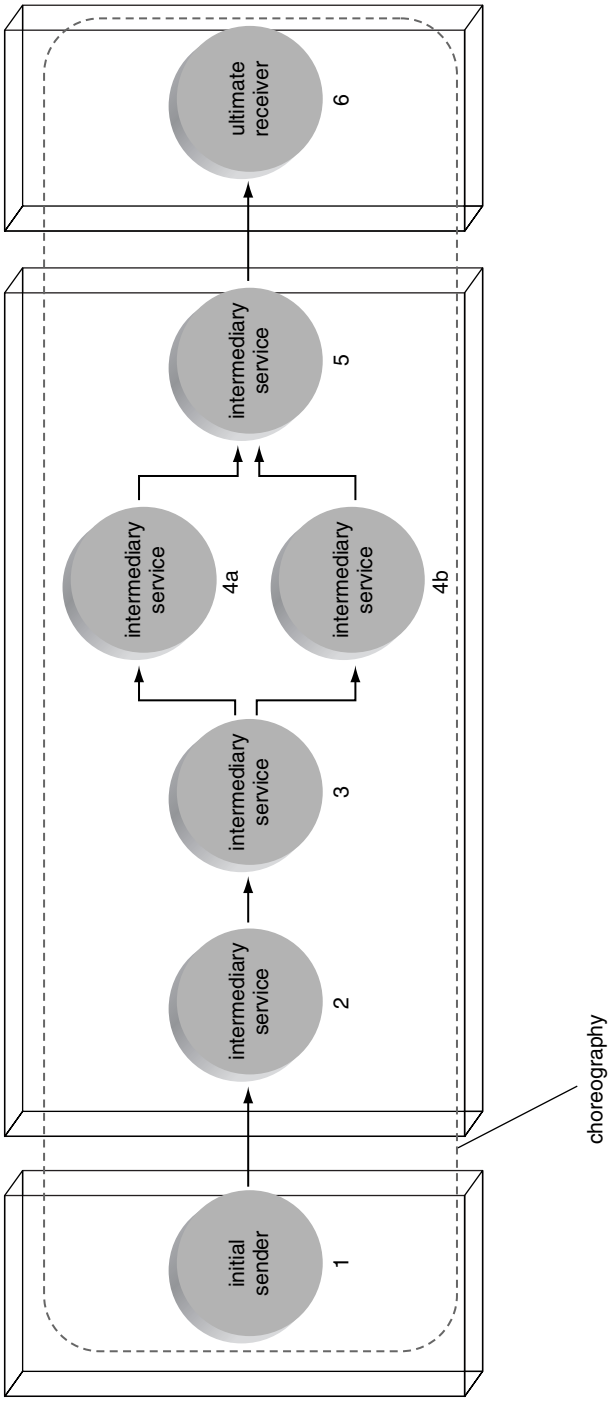

Figure 3.14
The interaction sequence of a group of services being governed by a choreography.

erl_ch03.fm Page 63 Wednesday, March 17, 2004 4:55 PM

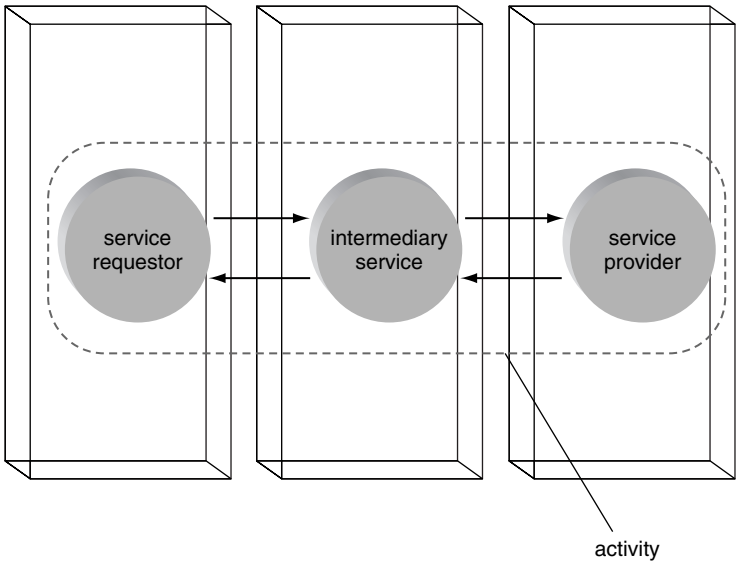

Figure 3.15
A service activity involving three services.

Table 3.1 Service models and the locations of their respective descriptions within this book

XWIF Service Model	Location
utility service	Chapter 6
business service	Chapter 6
controller service	Chapter 6
proxy service	Chapter 9
wrapper service	Chapter 9
coordination service (for atomic transactions)	Chapter 9
process service	Chapter 10
coordination service (for business activities)	Chapter 10

3.1.7 Web service description structure

An XML Web service is described through a stack of definition documents that constitute a service description. Figure 3.16 illustrates the relationship between these documents, each of which is described individually, thereafter.

erl_ch03.fm Page 64 Wednesday, March 17, 2004 4:55 PM

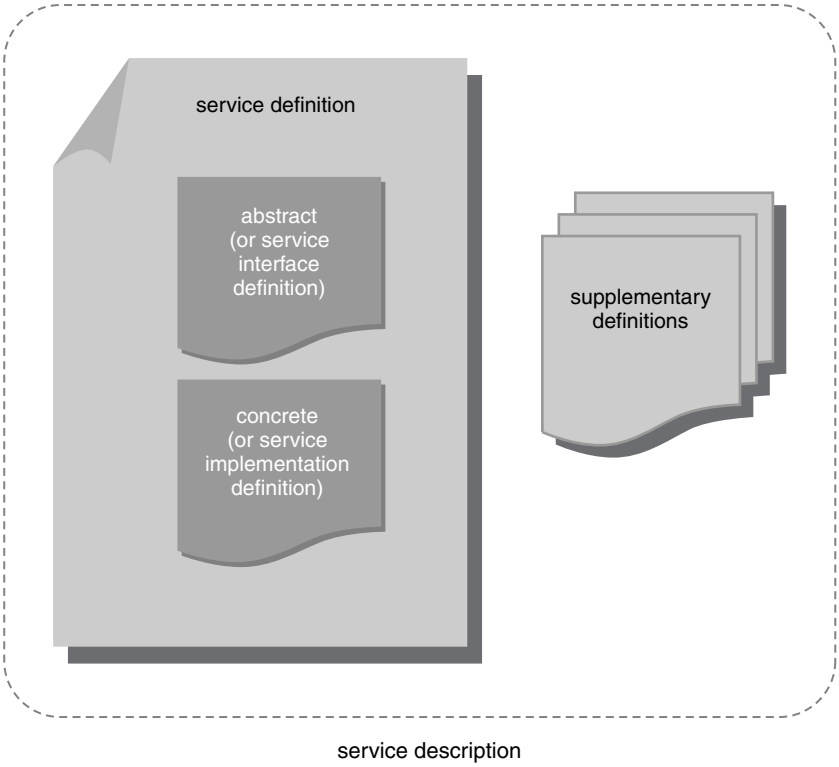

Figure 3.16
Contents of a service description.

Definition documents acts as building blocks for a service description:

Abstract + Concrete = Service Definition

Service Definition + Supplementary Definitions = Service Description

Abstract

The description of a Web service interface, independent of implementation details, is referred to as the *abstract*. Within a WSDL document, this abstract interface definition is primarily made up of the `interface` and `message` constructs. It is further supported by the `types` construct, which is often classified separately. (Descriptions of these elements are provided later in this chapter, as part of the WSDL tutorial.) In a component-based architecture, the service interface is often compared to the Interface Definition Language (IDL) file used to describe a component interface.

erl_ch03.fm Page 65 Wednesday, March 17, 2004 4:55 PM

NOTE

The term "abstract" superseded the term "service interface definition" as of the May 14, 2003 working draft release of the W3C Web services architecture specification.

Concrete

Specific location and implementation information about a Web service are the *concrete* parts of a WSDL document, as represented by the `binding`, `service`, and `endpoint` (or `port`) elements.

NOTE

The term "service implementation definition" was replaced with the term "concrete" in the May 14, 2003 working draft release of the W3C Web services architecture specification.

Service definition

Generally, the contents of a WSDL document constitute a *service definition*, which includes the interface (abstract) and implementation (concrete) definitions.

Service description

Often a *service description* consists of only a WSDL document providing a service definition; however, it can include a number of additional definition documents that can provide supplemental information (such as how this service relates to others).

3.1.8 Introduction to first-generation Web services

The W3C framework for Web services consists of a foundation built on top of three core XML specifications:

- Web Services Definition Language (WSDL)
- Simple Object Access Protocol (SOAP)
- Universal Description, Discovery, and Integration (UDDI)

These technology standards, coupled with service-oriented design principles, form a basic XML-driven SOA. This *first-generation Web services architecture* allows for the creation of independent Web services capable of encapsulating isolated units of business functionality. It also has a number of limitations, which have been addressed in a second generation of specifications. (Key second-generation Web services specifications are introduced through a series of tutorials in Chapter 4.)

erl_ch03.fm Page 66 Wednesday, March 17, 2004 4:55 PM

The next three sections provide introductory tutorials to each of the first-generation Web services technologies.

3.2 Web Services Definition Language (WSDL)

Web services need to be defined in a consistent manner so that they can be discovered by and interfaced with other services and applications. The Web Services Definition Language is a W3C specification providing the foremost language for the description of Web service definitions.

The integration layer introduced by the Web services framework establishes a standard, universally recognized and supported programmatic interface. As shown in Figure 3.17, WSDL enables communication between these layers by providing standardized endpoint descriptions.

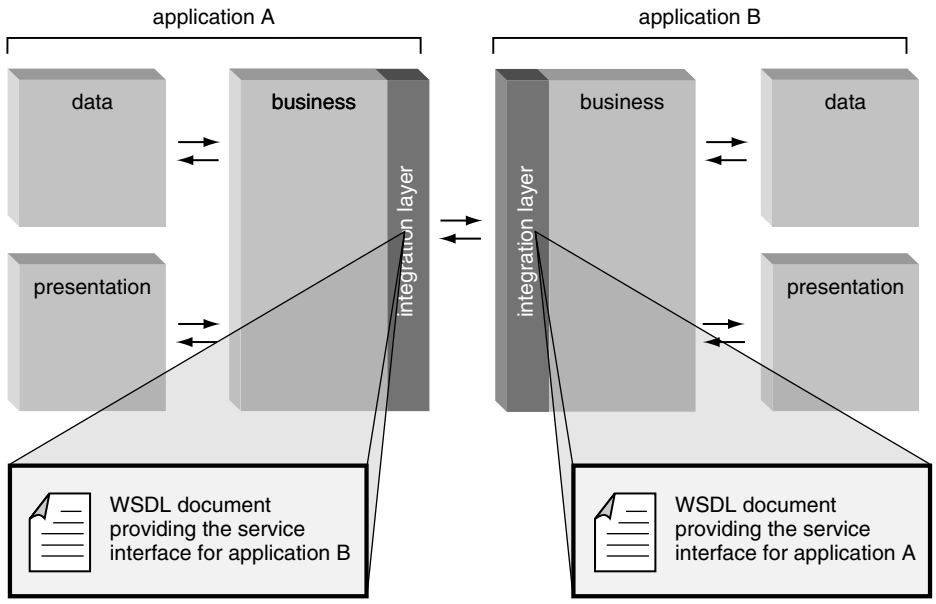

Figure 3.17
WSDL documents representing Web services to applications.

The best way to understand how a Web service is defined and expressed by a WSDL document is to step through each of the constructs that collectively represent this definition. Let's begin with the root `definitions` element, which acts as the container for the entire service definition.

erl_ch03.fm Page 67 Wednesday, March 17, 2004 4:55 PM


```
<definitions>
  <interface name="Catalog">
    ...
  </interface>
  <message name="BookInfo">
    ...
  </message>
  <service>
    ...
  </service>
  <binding name="Binding1">
    ...
  </binding>
</definitions>
```

Example 3.1 A service definition, as expressed by the definitions construct

A WSDL definition can host collections of the following primary constructs:

- `interface` (previously known as `portType`)[1]
- `message`
- `service`
- `binding`

Figure 3.18 illustrates how the first two constructs represent the service interface definition, and the latter two provide the service implementation details.

3.2.1 Abstract interface definition

Individual Web service interfaces are represented by WSDL `interface` elements. These constructs contain a group of logically related operations. In a component-based architecture, a WSDL `interface` is comparable to a component interface. An operation is therefore the equivalent of a component method, as it represents a single action or function.

```
<definitions>
  <interface name="Catalog">
    <operation name="GetBook">
      ...
    </operation>
  </interface >
</definitions>
```

Example 3.2 A Web service interface represented by the interface element

1. As of June 11, 2003, the WSDL specification changed the name of this element from `portType` to `interface`.

erl_ch03.fm Page 68 Wednesday, March 17, 2004 4:55 PM

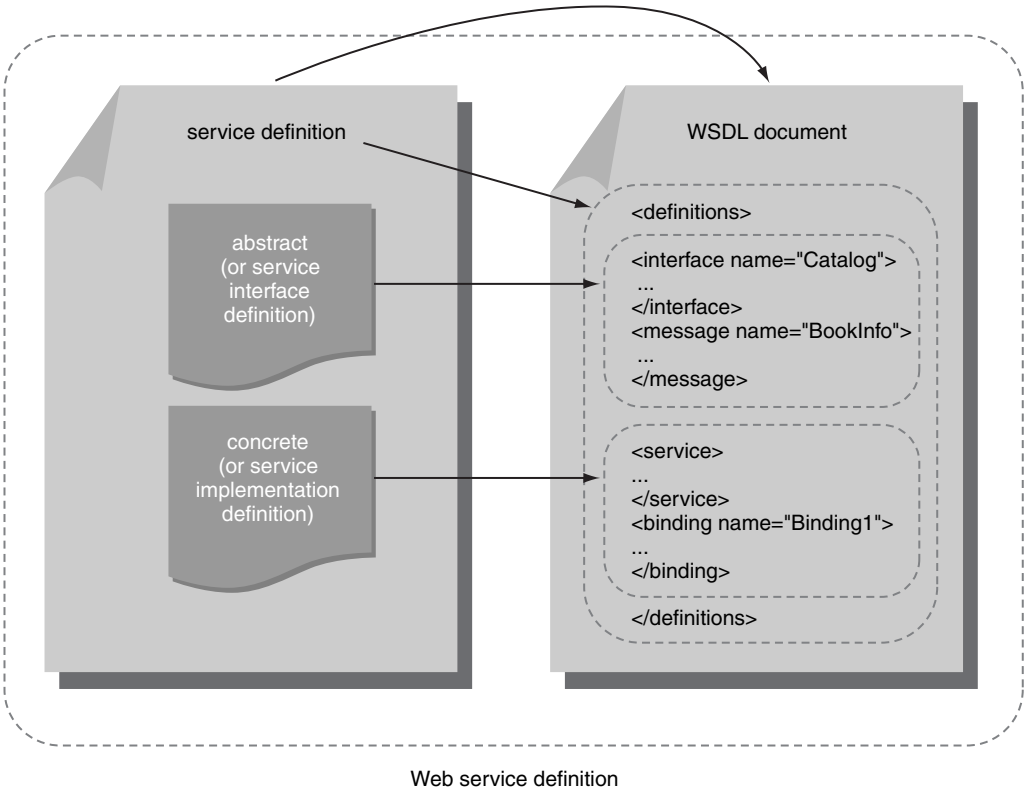

Figure 3.18
The contents of a WSDL document, as they relate to a service definition.

A typical `operation` element consists of a group of related input and output messages. The execution of an operation requires the transmission or exchange of these messages between the service requestor and the service provider.

Operation messages are represented by `message` constructs that are declared separately under the `definitions` element. The message names then are referenced in the operation's `input` or `output` child elements.

```
<definitions>
  <message name="BookInfo">
    ...
  </message>
  <interface name="Catalog">
    <operation name="GetBook">
      <input name="Msg1" message="BookInfo" />
    </operation>
```

erl_ch03.fm Page 69 Wednesday, March 17, 2004 4:55 PM


```
  </interface>
</definitions>
```

Example 3.3 The input element within an operation construct referencing a message block

A `message` element can contain one or more input or output parameters that belong to an operation. Each `part` element defines one such parameter. It provides a name/value set, along with an associated data type. In a component-based architecture, a WSDL `part` is the equivalent of an input or output parameter (or a return value) of a component method.

```
<definitions>
  <message name="BookInfo">
    <part name="title" type="xs:string">
      Field Guide
    </part>
    <part name="author" type="xs:string">
      Mr. T
    </part>
  </message>
</definitions>
```

Example 3.4 A message block with part constructs representing operation parameters

Here's a brief summary of the fundamental constructs that can be assembled to establish an abstract interface definition:

- `interfaces` represent service interfaces, and can contain multiple `operations`
- `operations` represent a Web service function, and can reference multiple `messages`
- `messages` represent collections of input or output parameters, and can contain multiple `parts`
- `parts` represent either incoming or outgoing `operation` parameter data

3.2.2 Concrete (implementation) definition

On to the implementation details. Using the elements described in this section, a WSDL document can establish concrete binding details for protocols, such as SOAP and HTTP.

Within a WSDL document, the `service` element represents one or more endpoints at which the Web service can be accessed. These endpoints consist of location and protocol information, and are stored in a collection of `endpoint` (previously known as `port`)[2] elements.

2. As of June 11, 2003, the WSDL specification changed the name of this element from `port` to `endpoint`.

erl_ch03.fm Page 70 Wednesday, March 17, 2004 4:55 PM


```
<definitions>
  <service name="Service1">
    <endpoint name="Endpoint1" binding="Binding1">
      ...concrete implementation details...
    </endpoint>
  </service>
</definitions>
```

Example 3.5 The endpoint element

Now that we've described how a Web service can be accessed, we need to define the invocation requirements of each of its operations. The `binding` element associates protocol and message format information to operations. The `operation` construct that resides within the `binding` block resembles its counterpart in the `interface` section.

```
<definitions>
  <service>
    <binding name="Binding1">
      <operation>
        <input name="Msg1" message="book" />
      </operation>
    </binding>
  </service>
</definitions>
```

Example 3.6 The binding element representing an existing operation

The description of concrete information within a WSDL document can be summarized as follows:

- `service` elements host collections of endpoints represented individually by `endpoint` elements
- `endpoint` elements contain endpoint data, including physical address and protocol information
- `binding` elements associate themselves to `operation` constructs
- each `endpoint` can reference a `binding` element, and therefore relates the endpoint information to underlying operation

3.2.3 Supplementary constructs

An additional feature used to provide data type support for Web service definitions is the `types` element. Its construct allows XSD schemas to be embedded or imported into the definition document.

erl_ch03.fm Page 71 Wednesday, March 17, 2004 4:55 PM


```
<definitions>
  <types>
    <xsd:schema
      targetNamespace="http://www.examples.ws/"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      ...
    </xsd:schema>
  </types>
</definitions>
```

Example 3.7 The types element establishing XSD data types

Finally, the optional `documentation` element allows supplementary annotations to be added.

```
<definitions>
  <documentation>
    I wrote this service definition some time ago,
    when I was younger and times were simpler for us all...
  </documentation>
</definitions>
```

Example 3.8 The documentation element allows you to supplement service definitions with annotations

SUMMARY OF KEY POINTS

- A service definition can be represented by the contents of a WSDL document, as defined within the `definitions` construct.
- The abstract interface definition is described by the `interface`, `message`, and `types` constructs. This part of the WSDL document provides a mobile, platform-independent description of the Web service interface.
- Concrete implementation information is contained within the `binding`, `service`, and `endpoint/port` elements. This part of the WSDL document is used to bind an abstract interface to specific protocols, such as SOAP and HTTP.

For more information regarding WSDL features and resources, visit `www.specifications.ws`.

3.3 Simple Object Access Protocol (SOAP)

Although originally conceived as a technology to bridge the gap between disparate RPC-based communication platforms, SOAP has evolved into the most widely supported messaging format and protocol for use with XML Web services. Hence the

erl_ch03.fm Page 72 Wednesday, March 17, 2004 4:55 PM

SOAP acronym is frequently referred to as the Service-Oriented Architecture (or Application) Protocol, instead of the Simple Object Access Protocol.

The SOAP specification establishes a standard message format that consists of an XML document capable of hosting RPC and document-centric data (see Figure 3.19). This facilitates synchronous (request and response) as well as asynchronous (process-driven) data exchange models. With WSDL establishing a standard endpoint description format for applications, the document-centric message format is much more common.

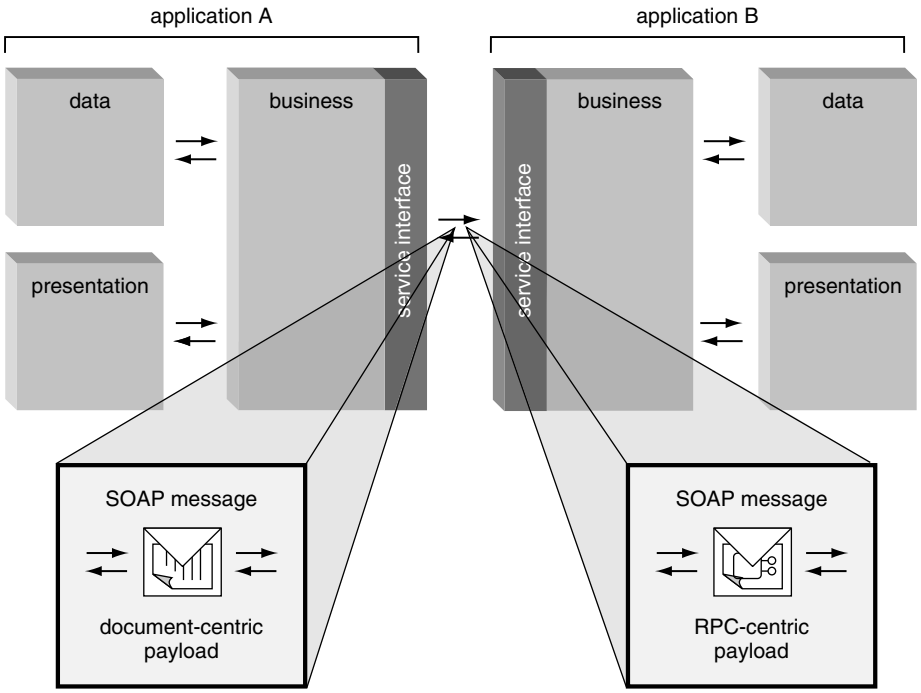

Figure 3.19
SOAP establishes two primary standard message formats.

The architectures we explore throughout this book make reference to both types of message formats using the standard diagram symbols provided in Figures 3.20 and 3.21.

Additionally, we discuss the use of SOAP message attachments, which are described in separate second-generation Web services specifications. (Specifically, the WS-Attachments and SOAP Messages with Attachments (SwA) standards, are covered in Chapter 4.) SOAP messages containing attachments are represented with the symbol shown in Figure 3.22.

erl_ch03.fm Page 73 Wednesday, March 17, 2004 4:55 PM

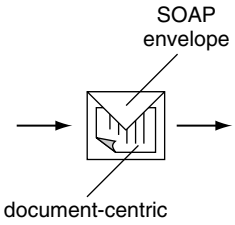

Figure 3.20
The symbol used to represent a SOAP message with a document-centric payload.

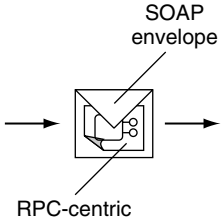

Figure 3.21
The symbol used to represent a SOAP message with an RPC-centric payload.

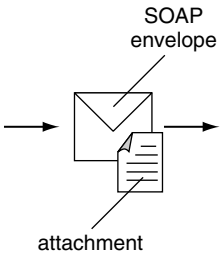

Figure 3.22
The symbol used to represent a SOAP message delivering its data as an attachment.

3.3.1 SOAP messaging framework

SOAP institutes a method of communication based on a processing model that is in relative alignment with the overall Web services framework described at the beginning of this chapter. It differs only in that it introduces terms and concepts that relate specifically to the manner in which SOAP messages need to be handled (within a technical implementation of this framework).

erl_ch03.fm Page 74 Wednesday, March 17, 2004 4:55 PM

Note that the diagram symbols used to identify SOAP nodes in the following sections are not displayed in other chapters. Their existence is implied in subsequent architectural diagrams that include Web services.

SOAP nodes

A *SOAP node* represents the processing logic responsible for transmitting, receiving, and performing a variety of processing tasks on SOAP messages. An implementation of a SOAP node is typically platform specific, and is commonly labeled as a *SOAP server* or a *SOAP listener*. Specialized variations also exist, such as *SOAP routers*. Conceptually, all are considered SOAP nodes. Figure 3.23 establishes the SOAP node as the underlying transport mechanism for a Web service.

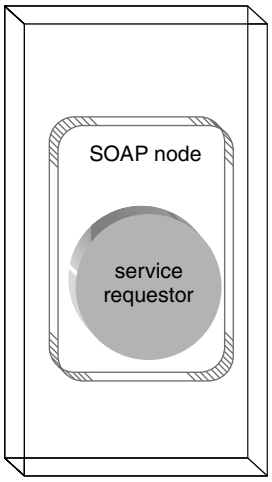

Figure 3.23
A SOAP node.

SOAP nodes are best viewed as the technical infrastructure that powers all of the communication scenarios explored in the earlier section, "Web service interaction." When discussing the involvement of SOAP nodes, however, a slight departure from the terms and concepts established by the Web services framework needs to be incorporated.

SOAP node types

Like Web services, SOAP nodes can exist as initial senders, intermediaries, and ultimate receivers. Whereas Web services are also classified as requestors and providers, SOAP nodes performing the equivalent tasks (sending, receiving) are referred to as *SOAP senders* and *SOAP receivers* (see Figure 3.24). The SOAP specification, however, does not classify these as roles. For the purpose of this book, we'll refer to them as "types" of SOAP nodes.

erl_ch03.fm Page 75 Wednesday, March 17, 2004 4:55 PM

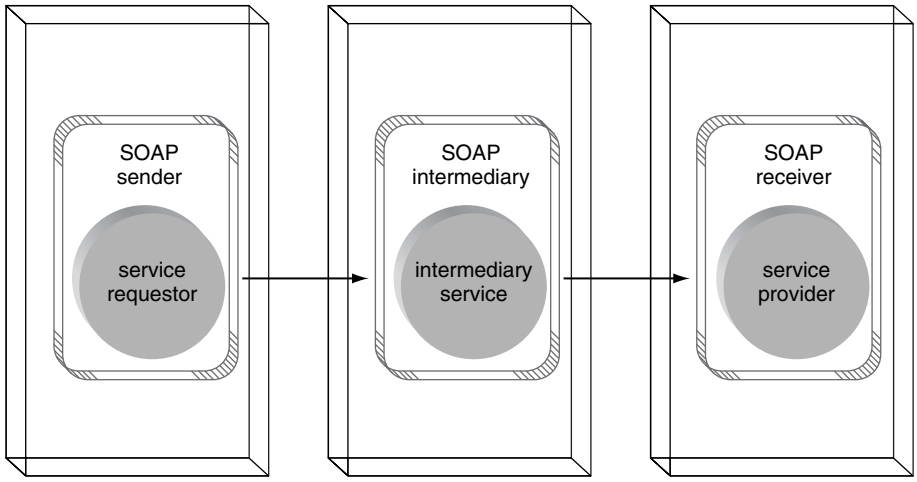

Figure 3.24
Fundamental SOAP node types along a message path.

As illustrated in Figure 3.25, a node type of SOAP initial sender is also a SOAP sender, and the SOAP ultimate receiver is also a SOAP receiver.

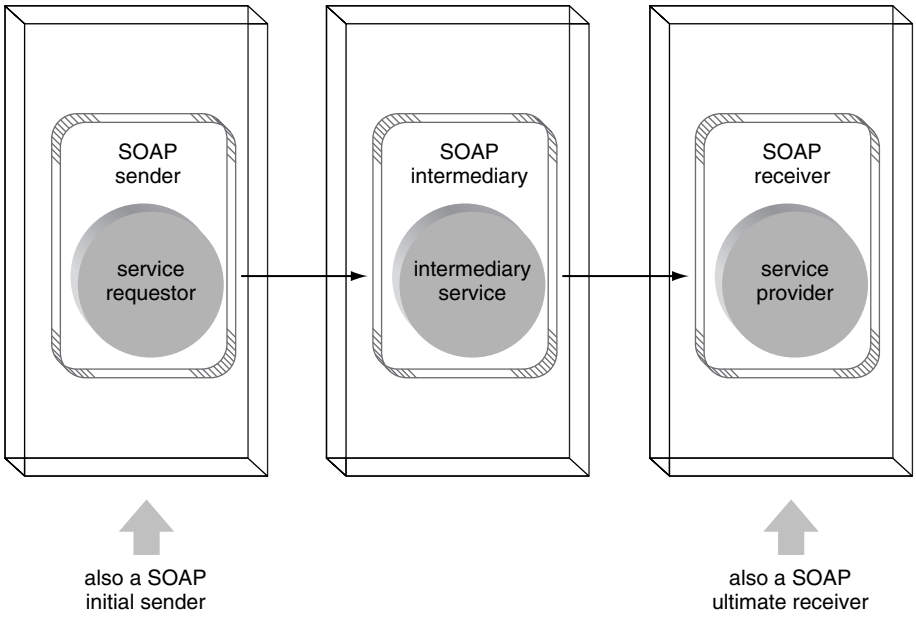

Figure 3.25
SOAP nodes with multiple types.

erl_ch03.fm Page 76 Wednesday, March 17, 2004 4:55 PM

Figure 3.26 shows how a SOAP node, acting as an intermediary, transitions through both SOAP sender and receiver types during the processing of a SOAP message.

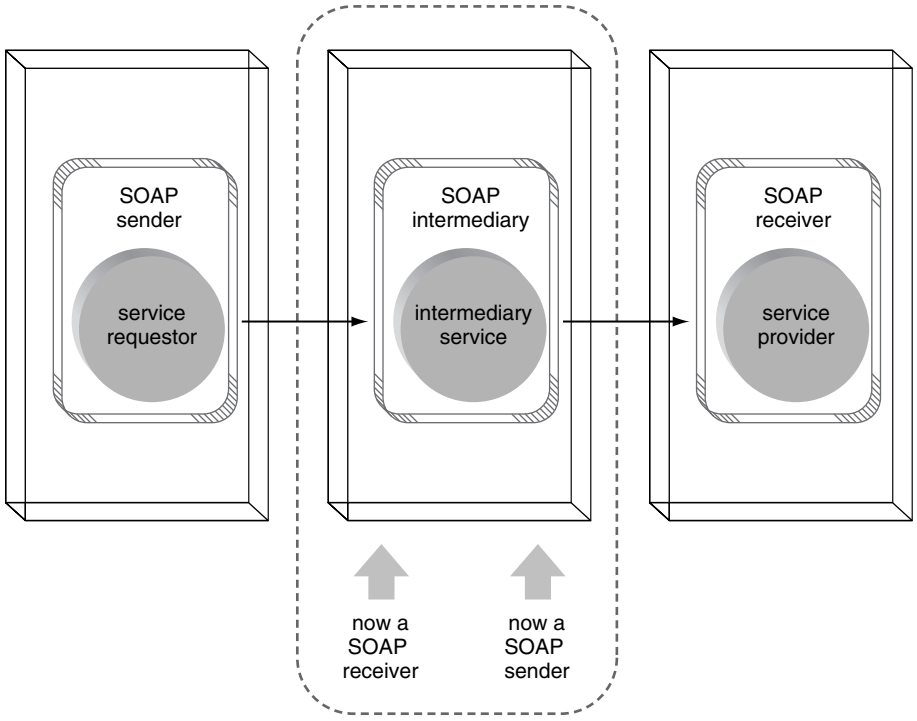

Figure 3.26
A SOAP node going through the transition of being a SOAP receiver and sender during the processing of a SOAP message.

As with Web service roles, the same SOAP node can act as different types depending on its position within the message path and the state of the current business activity. For instance, a SOAP node transmitting a message as the initial sender can later receive a response as the ultimate receiver.

NOTE
Roles for SOAP nodes also exist, but are described separately at the end of this section.

3.3.2 SOAP message structure

The container of SOAP message information is referred to as a *SOAP envelope*. Let's open it and take a brief look at the underlying structure of a typical SOAP message.

erl_ch03.fm Page 77 Wednesday, March 17, 2004 4:55 PM

The root `Envelope` element that frames the message document consists of a mandatory body section and an optional header area.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

Example 3.9 A skeleton envelope construct

The SOAP header is expressed using the `Header` construct, which can contain one or more sections or blocks.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:shipping >
      UPS
    </n:shipping>
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

Example 3.10 The Header construct with a header block

Common uses of header blocks include:

- implementation of (predefined or application-specific) SOAP extensions, such as those introduced by second-generation specifications
- identification of target SOAP intermediaries
- providing supplementary meta information about the SOAP message

While a SOAP message progresses along a message path, intermediaries may add, remove, or process information in SOAP header blocks. Although an optional part of a SOAP message, the use of the header section to carry header blocks is commonplace when working with second-generation Web services specifications.

The one part of a SOAP message that is not optional is the body. As represented by the `Body` construct, this section acts as a container for the data being delivered by the

erl_ch03.fm Page 78 Wednesday, March 17, 2004 4:55 PM

SOAP message. Data within the SOAP body is often referred to as the *payload* or *payload data*.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    <x:Book xmlns:x="http://www.examples.ws/">
      <x:Title>
        Service-Oriented Architecture
        A Field Guide to Integrating XML
        and Web services
      </x:Title>
    </x:Book>
  </env:Body>
</env:Envelope>
```

Example 3.11 The Body construct

The `Body` construct can also be used to host exception information within nested `Fault` elements. Although fault sections can reside alongside standard data payloads, this type of information is often sent separately in response messages that communicate error conditions.

The `Fault` construct consists of a series of system elements used to identify characteristics of the exception.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>
          env:VersionMismatch
        </env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en">
          versions do not match
        </env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Example 3.12 A sample fault construct providing error information

erl_ch03.fm Page 79 Wednesday, March 17, 2004 4:55 PM

SOAP node roles

Now that you've had a look at the internal structure and syntax of a SOAP message, let's finish by briefly introducing *SOAP node roles*. When discussing SOAP nodes, roles relate to an optional `env:role`[3] attribute that a SOAP message can use to identify header blocks intended for specific types of SOAP receivers. Therefore, SOAP roles are associated only to types of SOAP nodes that perform a receiving function. In other words, intermediaries and ultimate receivers (see Figure 3.27).

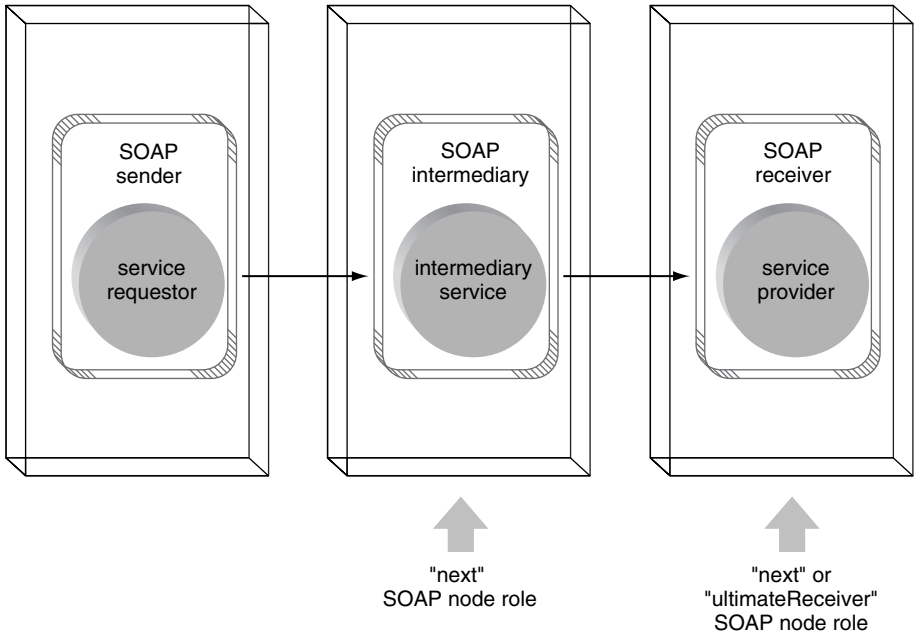

Figure 3.27
Roles that can be assumed by receiving SOAP nodes

The two most common `env:role` attribute values are `next` and `ultimateReceiver`. An intermediary node will process only header blocks identified with the `next` role, whereas a node acting as the ultimate receiver will process both.

To learn more about SOAP, header blocks, and how they relate to second-generation Web services, visit `www.specifications.ws`.

3. The `env:role` attribute was introduced in version 1.2 of the SOAP specification. It was previously named `env:actor`.

erl_ch03.fm Page 80 Wednesday, March 17, 2004 4:55 PM

SUMMARY OF KEY POINTS

- Implementations of the SOAP messaging framework can be collectively conceptualized as an end-to-end messaging engine that drives communication throughout contemporary service-oriented architectures.
- A SOAP message consists of a simple XML document structure. The parent `envelope` construct houses an optional `header` and a required `body` construct. Exception information can be placed in a special `Fault` element that is nested within the message body.
- The utilization of SOAP header blocks by second-generation Web services specifications is an important aspect of this framework that vastly increases its power and complexity.

3.4 Universal Description, Discovery, and Integration (UDDI)

One of the fundamental components of a service-oriented architecture is a mechanism for Web service descriptions to be discovered by potential requestors. To establish this part of a Web services framework, a central directory to host service descriptions is required. Such a directory can become an integral part of an organization or an Internet community, so much so, it is considered an extension to infrastructure.

This is why the Universal Description, Discovery, and Integration specification has become increasingly important. A key part of UDDI is the standardization of profile records stored within such a directory, also known as a *registry*. Depending on who the registry is intended for, different implementations can be created.

A *public business registry* is a global directory of international business service descriptions. Instances of this registry are hosted by large corporations (also referred to as *node operators*) on a series of dedicated UDDI servers. UDDI records are replicated automatically between repository instances. Some companies also act as UDDI registrars, allowing others to add and edit their Web service description profiles. The public business registry is complemented by a number of *service marketplaces* offering generic Web services for sale or lease.

Private registries are service description repositories hosted within an organization (see Figure 3.28). Those authorized to access this directory may include select external business partners. A registry restricted to internal users only can be referred to as an *internal registry*.

erl_ch03.fm Page 81 Wednesday, March 17, 2004 4:55 PM

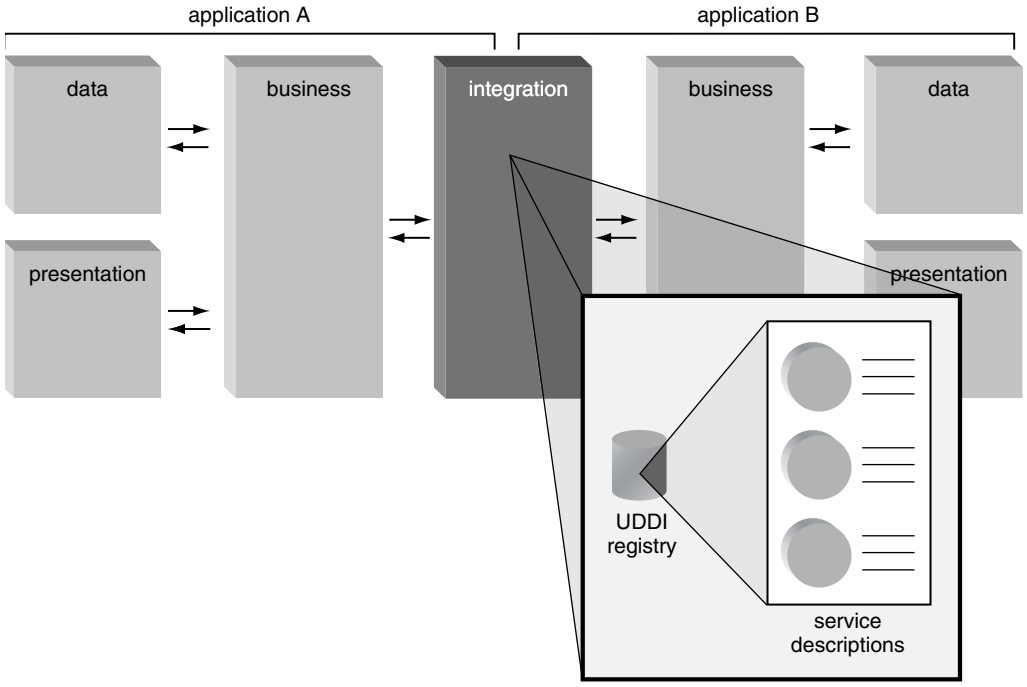

Figure 3.28
Service descriptions centralized in a private UDDI registry

The discovery process can occur in various situations depending on why service information is required. For instance:

- An organization seeking to establish new business relationships for online transactions can search for (and compare) suitable business partners using a public business registry.
- When building an inter-enterprise integration channel, the architect working for an organization's business partner will need to learn about the organization's external contact points. Service interfaces and the logic they express will form the basis for the Web services designed by the business partner. Access to the organization's private registry allows the architect to efficiently gather this information.
- An architect designing a new e-Business application may first want to research the availability of generic programming logic within an organization. By reading through existing service descriptions, opportunities for reuse may be discovered. Centralizing service descriptions in an internal registry provides a convenient resource repository for public endpoint descriptions within an enterprise.

erl_ch03.fm Page 82 Wednesday, March 17, 2004 4:55 PM

- That same architect may also want to shop for a third-party Web service providing pre-built application logic that could be incorporated (locally or remotely) within the e-Business application. Service marketplaces offer venues to purchase or lease third-party Web services.
- A developer building new services will need to access interface definitions for existing services. The internal registry spares the developer from having to worry about whether the service interfaces being incorporated are current.

UDDI registries organize registry entries using six primary types of data:

- business entities
- business services
- specification pointers
- service types
- business relationships
- subscriptions

Business entity data, as represented by the `businessEntity` element, provides profile information about the registered business, including its name, a description, and a unique identifier.

Here is a sample XML document containing a `businessEntity` construct.

```
<businessEntity xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7"
  operator="Microsoft Corporation"
  authorizedName="Thomas Erl"
  xmlns="urn:uddi-org:api_v2">
  <discoveryURLs>
    <discoveryURL useType=
      "businessEntity">http://test.uddi.microsoft.com/discovery
      ?businesskey=e9355d51-32ca-49cf-8eb4-1ce59afbf4a7
    </discoveryURL>
  </discoveryURLs>
  <name xml:lang="en">
    XMLTC Consulting Inc.
  </name>
  <description xml:lang="en">
    XMLTC has been building end-to-end enterprise
```

erl_ch03.fm Page 83 Wednesday, March 17, 2004 4:55 PM


```
    eBusiness solutions for corporations and
    government agencies since 1996. We offer a
    wide range of design, development and
    integration services.
  </description>
  <businessServices>
    <businessService
      serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a"
      businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7">
      <name xml:lang="en-us">
        Corporate Home Page
      </name>
      <bindingTemplates>
        <bindingTemplate
          bindingKey="48b02d40-0312-4293-a7f5-4449ca190984"
          serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a">
          <description xml:lang="en">
            Entry point into the XMLTC Web site
            through which a number of resource
            sites can be accessed.
          </description>
          <accessPoint URLType="http">
            http://www.xmltc.com/
          </accessPoint>
          <tModelInstanceDetails />
        </bindingTemplate>
      </bindingTemplates>
      <categoryBag>
        <keyedReference
          tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
          keyName="Namespace" keyValue="namespace" />
      </categoryBag>
    </businessService>
  </businessServices>
</businessEntity>
```

Example 3.13 An actual business entity document retrieved from a public service registry

NOTE
This document can be retrieved manually or programmatically using the URL `http://test.uddi.microsoft.com/discovery?businesskey =e9355d51-32ca-49cf-8eb4-1ce59afbf4a7`

Let's take this document apart to study the individual constructs.

erl_ch03.fm Page 84 Wednesday, March 17, 2004 4:55 PM


```
<businessEntity xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7"
  operator="Microsoft Corporation"
  authorizedName="Thomas Erl"
  xmlns="urn:uddi-org:api_v2">
```

Example 3.14 The parent businessEntity element with a number of attributes

When I registered XMLTC Consulting Inc. it was given a unique identifier of `e9355d51-32ca-49cf-8eb4-1ce59afbf4a7`, which was then assigned to the `businessKey` attribute of the `businessEntity` parent element. Since Microsoft acted as the node operator providing an instance of the UDDI registry, its name is displayed in the `businessEntity` element's operator attribute.

The `discoveryURL` element identifies the address used to locate this XML document.

```
<discoveryURLs>
  <discoveryURL useType="businessEntity">
    http://test.uddi.microsoft.com/discovery
    ?businesskey=e9355d51-32ca-49cf8eb4-1ce59afbf4a7
  </discoveryURL>
</discoveryURLs>
```

Example 3.15 The discoveryURLs construct containing the original URL

The `name` element simply contains the official business name.

```
<name xml:lang="en">
  XMLTC Consulting Inc.
</name>
```

Example 3.16 The name element providing the business name

Business service records representing the actual services offered by the registered business are nested within the `businessEntity` construct.

```
<businessServices>
  <businessService
    serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a"
    businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7">
    <name xml:lang="en-us">
      Corporate Home Page
    </name>
    <bindingTemplates>
      <bindingTemplate
        bindingKey="48b02d40-0312-4293-a7f5-4449ca190984"
        serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a">
```

erl_ch03.fm Page 85 Wednesday, March 17, 2004 4:55 PM


```
        <description xml:lang="en">
          Entry point into the XMLTC Web site
          through which a number of resource
          sites can be accessed.
        </description>
        <accessPoint URLType="http">
          http://www.xmltc.com/
        </accessPoint>
        <tModelInstanceDetails />
      </bindingTemplate>
    </bindingTemplates>
    <categoryBag>
      <keyedReference
        tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
        keyName="Namespace" keyValue="namespace" />
    </categoryBag>
  </businessService>
</businessServices>
```

Example 3.17 The businessServices construct

A business service is identified with a unique value assigned to the `serviceKey` attribute. Its parent `businessEntity` element is referenced by the `businessKey` attribute.

```
<businessService
  serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a"
  businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7">
  ...
</businessService>
```

Example 3.18 The businessService element's serviceKey and businessKey attributes

The only business service associated with this business entity is the business's Web site home page, as identified by the `name` element.

```
<name xml:lang="en-us">
  Corporate Home Page
</name>
```

Example 3.19 The name element with the service name

Each business service provides *specification pointers*. Also known as *binding templates*, these records consist of addresses linking the business service to implementation information. Using service pointers, a developer can learn how and where to physically bind to a Web service.

erl_ch03.fm Page 86 Wednesday, March 17, 2004 4:55 PM


```
<bindingTemplates>
  <bindingTemplate
    bindingKey="48b02d40-0312-4293-a7f5-4449ca190984"
    serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a">
    <description xml:lang="en">
      Entry point into the XMLTC Web site
      through which a number of resource
      sites can be accessed.
    </description>
    <accessPoint URLType="http">
      http://www.xmltc.com/
    </accessPoint>
    <tModelInstanceDetails />
  </bindingTemplate>
</bindingTemplates>
```

Example 3.20 The bindingTemplates construct housing concrete location information

The `bindingTemplate` construct displayed in the preceding example establishes the location and description of the service using the `accessPoint` and `description` elements.

Various categories can be assigned to business services. In our example, the URL we identified has been classified as a namespace using the `keyedReference` child element of the `categoryBag` construct.

```
<categoryBag>
  <keyedReference
    tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
    keyName="Namespace" keyValue="namespace" />
</categoryBag>
```

Example 3.21 The categoryBag element providing a categorization using the nested keyedReference element

There is no formal relationship between UDDI and WSDL. A UDDI registry provides a means of pointing to service interface definitions through the use of a *tModel*. Though it would most likely be a WSDL document, it does not have to be. The tModel represents the definition of the UDDI *service type*, and also can provide information relating to message formats, as well as message and security protocols.

Finally, business relationship and subscription data is represented by `publisherAssertion` and `subscription` elements, respectively. `publisherAssertion` constructs provide a means of establishing the relationship of the current `businessEntity` with another. `Subscription` allows subscribers to be notified when business entity profile information is updated.

erl_ch03.fm Page 87 Wednesday, March 17, 2004 4:55 PM

You can interface programmatically with a UDDI registry. The UDDI specification provides a number of APIs that can be grouped into two general categories: *inquiry* and *publishing*. For instance, you could issue a SOAP message to search for a company by name with the following payload:

```
<find_business xmlns="urn:uddi-org:api_v3">
  <findQualifiers>
    <findQualifier>
       uddi:uddi.org:findQualifier:exactMatch
    </findQualifier>
  </findQualifiers>
  <name>
    XMLTC Consulting Inc.
  </name>
</find_business>
```

Example 3.22 The find_business construct encasing a command for the UDDI inquiry API

Although this brief overview has discussed the fundamentals of UDDI (with a focus on the structure of business entities), it has not delved into the heart of a UDDI registry: the tModel. This important construct provides access to the technical details required for requestors to interface and interact with available Web services.

SUMMARY OF KEY POINTS

- UDDI directories can be implemented as public business registries, private registries, and internal registries.
- Registry entries consist of the following profile information: business entities, business services, specification pointers, service types, business relationships, and subscriptions.
- UDDI provides inquiry and publishing APIs, allowing applications to interface programmatically with a registry.

To learn more about the tModel and UDDI in general, visit `www.specifications.ws`.

erl_ch03.fm Page 88 Wednesday, March 17, 2004 4:55 PM