Chapter 3

Web Services and Service-Oriented Architectures

Contents

Service-oriented architecture is a way to design, implement, and assemble services to support or automate business functions. Various Web services can be used to connect services. This chapter first explains Web services connections. It begins with

Web Services Service-Oriented Arch. DOI: http://dx.doi.org/10.1016/B978-0-12-398357-2.00003-8

More often than not, you can look to the past to find a pattern that will allow you to predict the future. I had an epiphany of this sort concerning the future of software systems architecture back in 2002 when I was writing the first edition of this book. At the time, I was upgrading my AV system. The past for this analogy is my old AV system and the future is the continued evolution of my AV system.

Since 2002, I have continued to evolve my AV system. The cable box was replaced with a digital video recorder (DVR) from my cable company. The VCR was removed, and I decided to resurrect an old turntable to play some of my vinyl albums. I have kept the same receiver and have resisted getting a flat-screen TV. All these components were connected using RCA connectors.

When we recently moved into a new home, my wife and I decided it was time to upgrade to a high-definition TV (HDTV). Of course, I now need to use high-definition multimedia interface (HDMI) connectors, yet I still have my old CD player and turntable. The DVR needed to be upgraded to HD and we purchased a new receiver that could handle HDMI as well as the older audio inputs that use RCA connectors. Figure 3.1 shows how I connected the various components.

an analogy to connections used in audio-video (AV) systems (specifically, services in a service-oriented architecture are to AV components as Web services are to the connections between AV components). The connection technology of Web services is explained along with the importance of standardized semantic vocabularies. Then service-oriented architectures are explained in more detail.

What does this have to do with software systems architecture? Well, it's all in the connections. Web services are connections not unlike those we have with AV systems. Moreover, just like AV systems, we will be able to assemble components in all sorts of ways because of those connections. In much the same way that that RCA and HDMI connectors are used to connect components to carry standardized audio and video signals, Web services connections increasingly use standardized

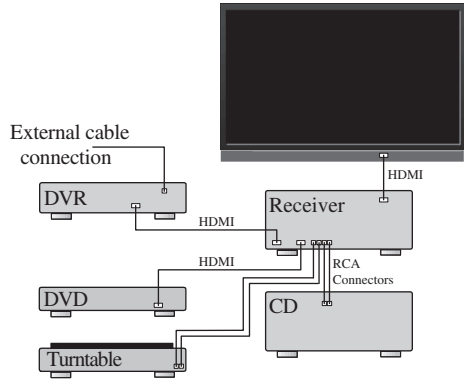

Figure 3.1 AV components.

semantic vocabularies to transport data (I'll explain more about vocabularies later in this chapter).

Service-Oriented Architecture Overview

The business trip that C. R. took in the introductory story in Chapter 1 involved using multiple services, both inside and outside his organization, such as travel, car rental, online calendar, and customer relationship management (CRM) services. From a software architectural point-of-view, this is a *service-oriented architecture* (SOA). An SOA is built using a collection of services that communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed. Those connections are Web services. The Application Program Interface (APIs) mentioned in Chapters 1–2 use Web services.

Services

A service is software and hardware. One or more services support or automate a business function. Most often, the intent is that a service can be used in multiple ways (often referred to as *reusability*). There are two types of services: atomic and composite. An *atomic* service is a well-defined, self-contained function that does not depend on the context or state of other services. A *composite* service is an assembly of atomic or other composite services. A service within a composite service may depend on the context or state of another service that is also within the same composite service.

The analogy to AV components fits well here. Manufacturers have decided on the basic functions of a DVD player, a DVR, and other components. Most of the AV components are analogous to composite services. For example, the turntable in our example also has a preamp. Audiophiles might prefer a separate preamp. In that case, both the turntable and the preamp would be analogous to atomic services.

Organizations will eventually evolve standard capabilities of CRM, enterprise resource planning (ERP), and other services. These will become standard services and could, in some ways, be seen as commodities. We may see these services come in various forms, just as AV components do today.[1]

What does this mean for software development? It means fewer people writing software and more organizations buying software or renting access to software rather

[1] The organizations working on the various standards can be found at *http://www.service-architecture.com/web-services/articles/organizations.html*.

than building it. Continuing with the AV analogy: I am old enough to have built my share of Heathkit electronic kits for audio and other systems. (This was much like building your own software.) The Heathkit era for electronics is over. I believe a lot of software development will go the same way.

Connections

Web services provide the means of connecting services. Just like there are multiple types of connections that can be used in an AV system (RCA, HDMI, etc.), there are multiple types of Web services for connection services (they will be discussed next in the "Web Services Explained" section).

Connections such as Web services are part of the inevitable evolution of interconnectedness. Consider how we can now exchange email among disparate products. Although we could not do that at one time, we now take it for granted. This e-mail exchange is possible because of standards. Connections like Web services (or the equivalent) will also be taken for granted some day because sets of standards will be developed.

Figure 3.2 illustrates a basic service-oriented architecture. It shows a service consumer at the right sending a service request message to a service provider at the left. The service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and the service provider.

A service provider can also be a service consumer. In the story of C. R.'s business trip, most of the service providers were also service consumers. For example, the virtual private assistant (VPA) service provided travel information, but to do that it needed to consume information from hotel services, car rental services, calendar services, and more.

The Architecture in SOA

There is more to the architecture of an SOA than described here. There are issues such as the granularity of services, loose coupling, composability, and more that need to be considered when designing a service-oriented architecture. Concepts related to these issues are described later in this chapter.

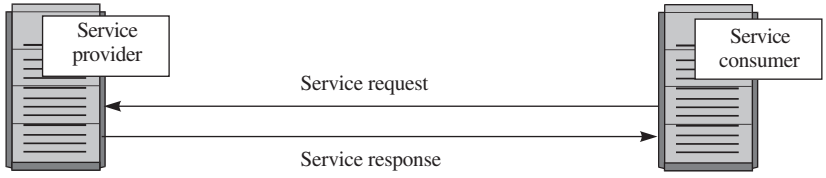

Figure 3.2 SOA basics.

Web Services Explained

Earlier, Web services were described as a connection technology. To get a full understanding of Web services, the history of the first Web services specification is discussed here.

History of Web Services Specification

Originally the only Web services specification included the Web Services Description Language (WSDL); Universal Description, Discovery, and Integration (UDDI): and SOAP. Over time, interest in UDDI has faded. Just to give you historical context, here is an overview of how the original specification was intended to work.

Web Services Description Language

WSDL forms the basis for the original Web services specification. Figure 3.3 illustrates the use of WSDL. At the left is a service provider and at the right is a

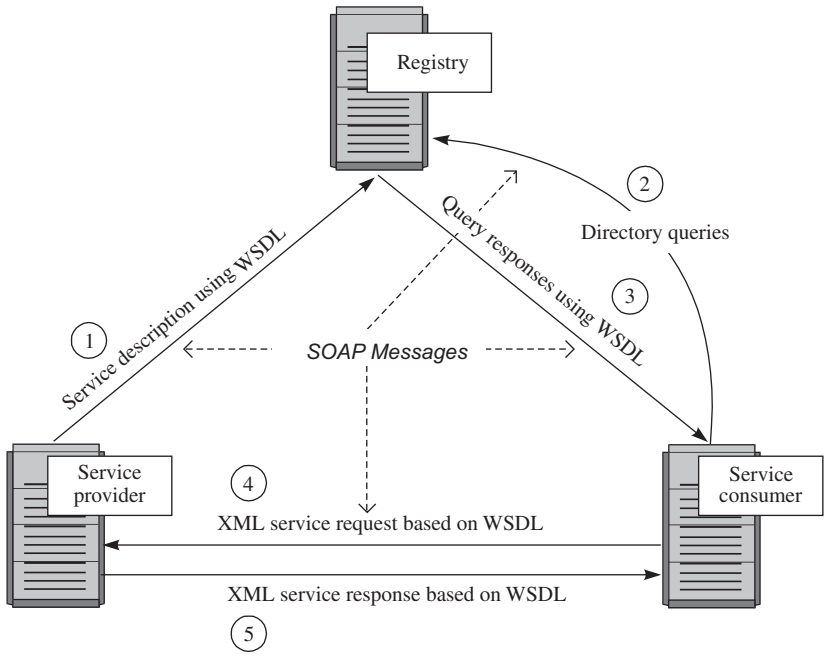

Figure 3.3 Web services basics.

service consumer. The steps involved in providing and consuming a service are as follows:

1. A service provider describes its service using WSDL. This definition is published to a registry of services. The registry uses UDDI.
2. A service consumer issues one or more queries to the registry to locate a service and determine how to communicate with that service.
3. Part of the WSDL provided by the service provider is passed to the service consumer. This tells the service consumer what the requests and responses are for the service provider.
4. The service consumer uses the WSDL to send a request to the service provider.
5. The service provider provides the expected response to the service consumer.

Universal Description, Discovery, and Integration

The UDDI registry was intended to serve as a means of "discovering" Web services described using WSDL. The idea was that the UDDI registry could be searched in various ways to obtain contact information and the services available from various organizations. UDDI registries have not been widely implemented.

The term *registry* is sometimes used interchangeably with the term *service repository*. Generally, repositories contain more information than a strict implementation of a UDDI registry. Today, instead of active discovery, repositories are used mainly at design time and to assist with governance.

SOAP

All the messages shown in Figure 3.3 are sent using SOAP. (SOAP at one time stood for Simple Object Access Protocol; now the letters in the acronym have no particular meaning.[2]) SOAP provides the envelope for sending Web services messages. SOAP generally uses HTTP, but other means of connection may be used. HTTP is the familiar connection we all use for the Internet.

Figure 3.4 provides more detail on the messages sent using Web services. At the left of the figure is a fragment of the WSDL sent to the registry. It shows a CustomerInfoRequest that requires the customer's account to object information. Also shown is the CustomerInfoResponse that provides a series of items on the customer including name, telephone, and address items. At the right of the figure is a

[2] Starting with SOAP version 1.2, SOAP is no longer an acronym.

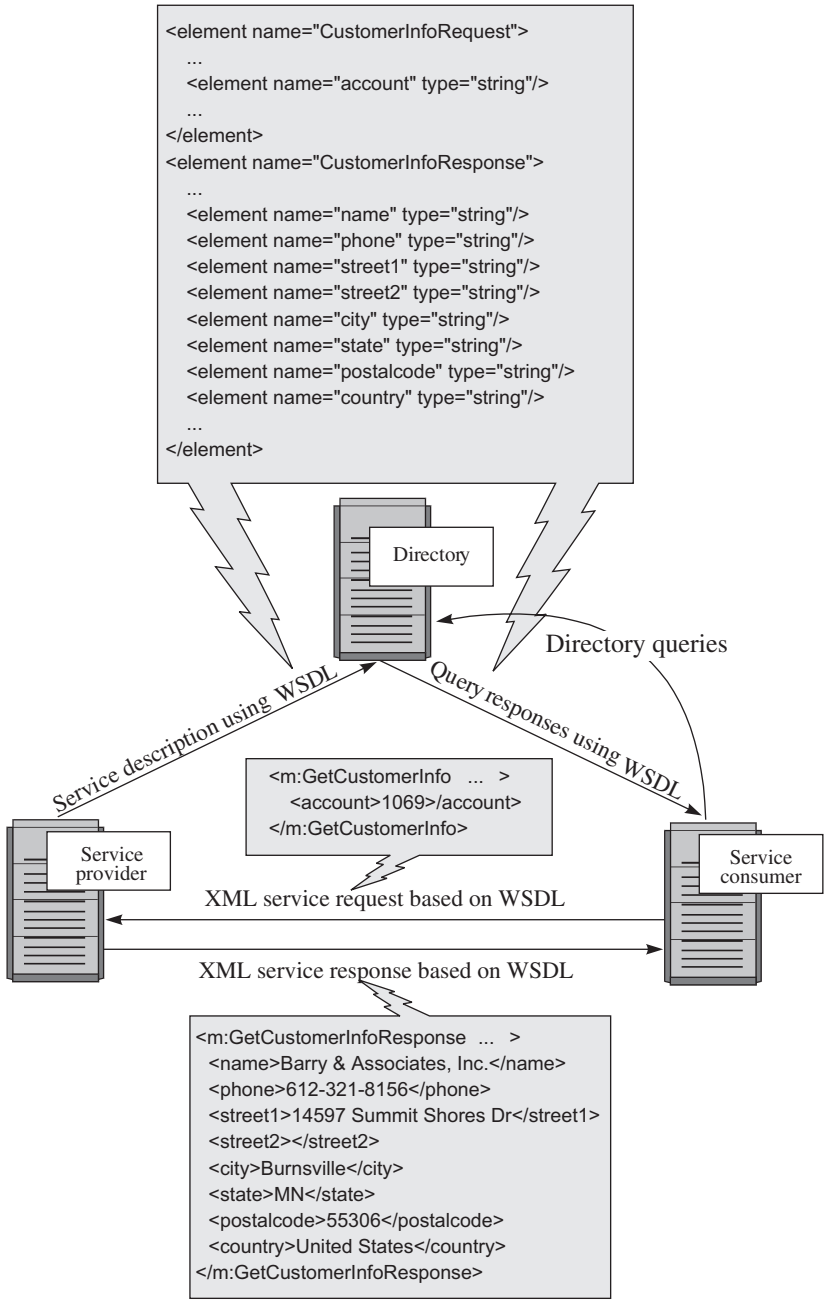

Figure 3.4 SOAP messaging with a directory.

fragment of the WSDL sent to the service consumer. This is the same fragment sent to the directory by the service provider. The service consumer uses this WSDL to create the service request shown above the arrow connecting the service consumer to the service provider. Upon receiving the request, the service provider returns a message using the format described in the original WSDL. That message appears at the bottom of Figure 3.4.

Web Services Specifications

There are multiple specifications that can be used for Web services. This section shows examples for SOAP/WSDL without UDDI, REST, XML, and JSON.

Using SOAP without UDDI

It is possible to use SOAP without UDDI. The connection is, instead, "hard-coded" if you will. The resulting interaction involves only the bottom part of Figure 3.4. The interaction between the service provider and the service consumer is shown in Figure 3.5. This is the nature of virtually all SOAP Web services today.

Using REST

The first alternative to SOAP that was developed is Representational State Transfer (REST). REST is a style of architecture based on a set of principles that describe how networked resources are defined and addressed. Roy Fielding first described these principles in 2000 as part of his doctoral dissertation.[3]

REST appeals to developers because it has a simpler style that makes it easier to use than SOAP, is a bit less verbose than SOAP (sends less down the "wire"), and is used in a way that other resources are used on the Internet.

Figure 3.6 illustrates a fragment of a REST message. It looks a lot like any other HTTP request that uses parameters. The return message in this example looks much like the return messages from SOAP.

[3] Chapter 5 of Roy Thomas Fielding's doctoral dissertation "Architectural Styles and the Design of Network-based Software Architectures" addresses REST. See *www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.*

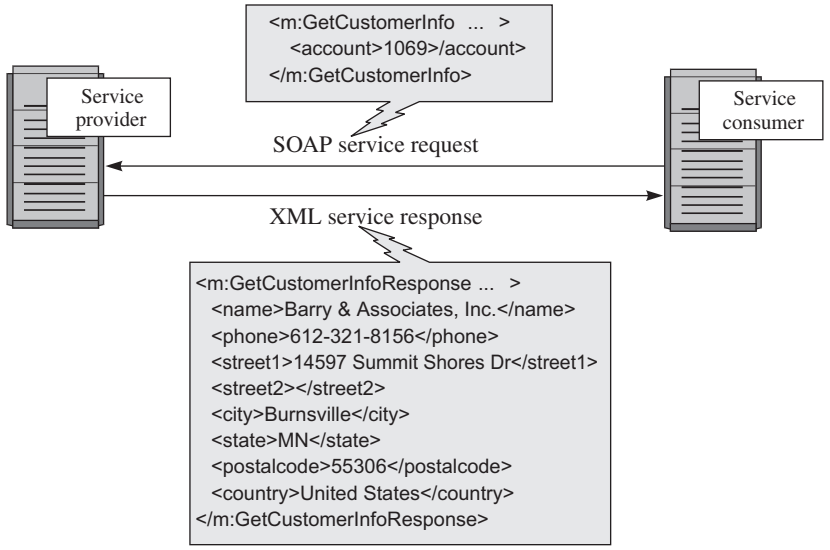

Figure 3.5 SOAP messaging.

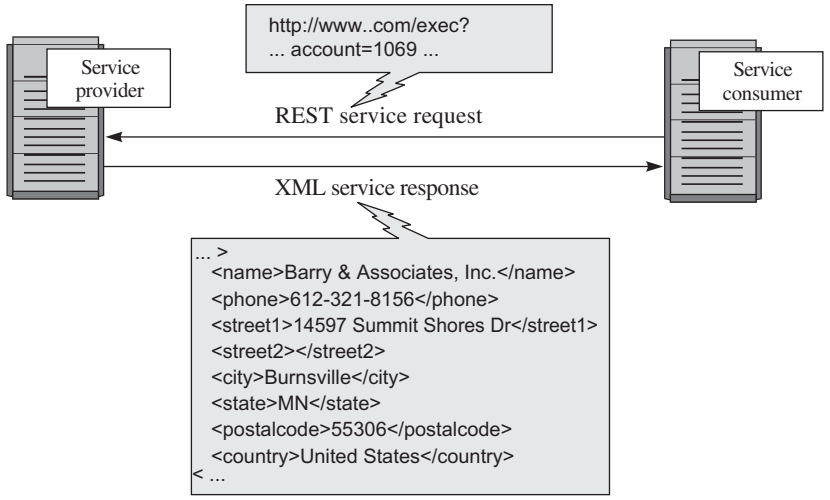

Figure 3.6 REST messaging.

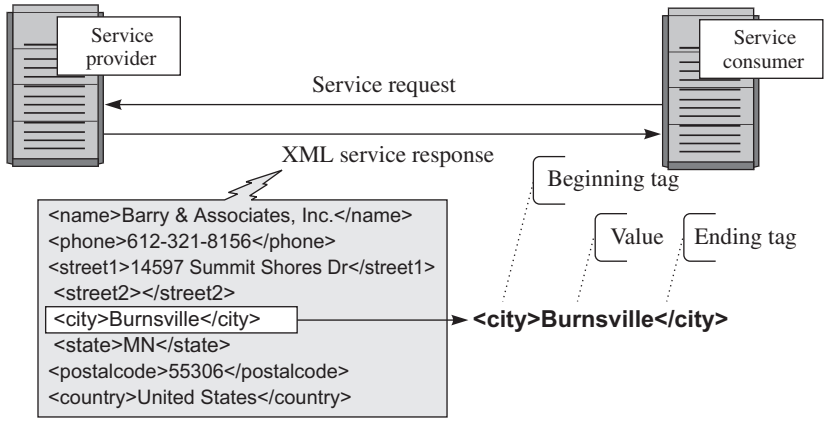

Figure 3.7 Tagged messages.

Using XML

The examples here show both SOAP and REST using XML for response messages. XML has a *tagged* message format. This is shown in Figure 3.7. The tag <city> is highlighted in this figure. The value of city is Burnsville. The tag </city> is the ending tag indicating the end of the value of city. Both the service provider and service consumer use these tags. In fact, the service provider could send the data shown at the bottom of Figure 3.7 in any order. The service consumer uses the tags and not the order of the data to get the data values.

The XML-tagged format provides a level of resilience not available with fixed record formats commonly used before the advent of XML. For example, if a service provider adds an additional element not expected by a service consumer, the XML-tagged format allows processing to continue without any problems occurring.

What if the data sent changes when using XML? Figure 3.8 shows that a service provider has added a new element, <extension> for a telephone extension. The service provider sends a response that includes the new element. As can happen, the service consumer did not know about the new element. Let's see what happens when XML-tagged messages are used.

The service consumer does not expect to receive the telephone extension. Nevertheless, because of the XML-tagged messages, essentially nothing bad happens when extra data (the value of the phone extension) is passed back by the service provider. This is shown at the bottom of Figure 3.9. The tags are used to identify each of the data items and the service consumer uses the proper values. The extra telephone extension data is simply ignored. Although it might be nice to have the extension data, the good news is that no other data is received incorrectly.

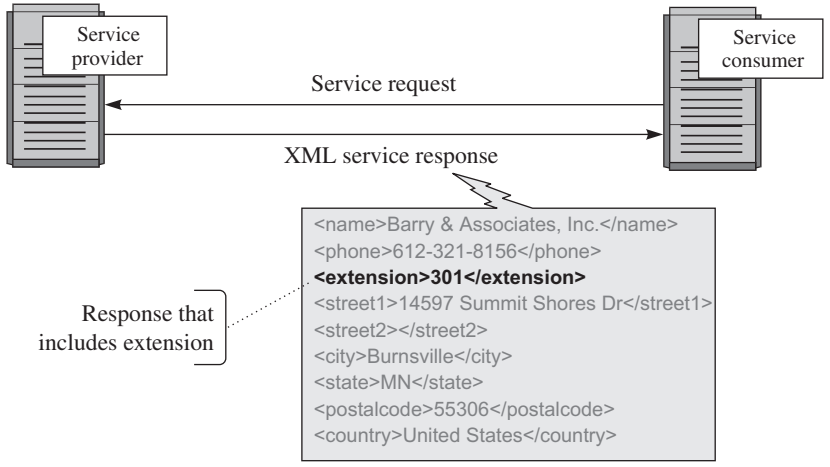

Figure 3.8 Adding a new element.

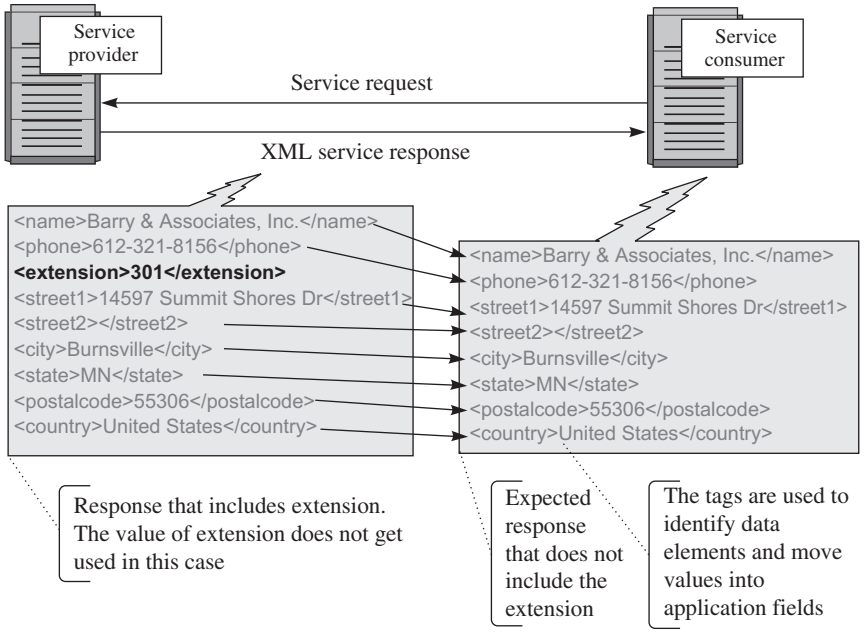

Figure 3.9 Example of the resilience provided by tagged messages.

If a fixed record format was used and the same error occurred, there could be harm. Let's look at this situation. Figure 3.10 shows a fixed record format that passes the same data related to customers. The length of this record is 129 characters. Now, assume the EXTENSION field is added after the PHONE field, but to keep the record length to 129 characters, the STREET2 field is shortened by three characters.

Figure 3.11 shows this change. Assume the same situation occurs as previously described. The service consumer does not know about the new element that contains a value for the telephone extension. Because the fixed record format assumes everything is based on position, whatever appears in a particular position is moved into a field in the service consumer. Figure 3.11 shows that both the EXTENSION and STREET1 fields are moved into the first street address in the service consumer.

Figure 3.12 provides another way to view how this happened. In fixed record messaging, everything is positional. Since the service consumer was unaware of the record change, it moved "30114597 Summit Shores Dr" into the STREET1 field shown at the bottom of Figure 3.12.

The effect of a change like this can vary. Obviously, if the service consumer sent postal mail to this address, it could not be delivered. Less obvious is the situation when a customer record does not have a phone extension. Then the first three spaces of the STREET1 field in the service consumer would be spaces. If the service consumer sent postal mail to this address, it could then be delivered as long as the address was no longer than 22 characters. If the address line exceeded 17 characters then the last part of the address line would appear on the first part of the second address line. That may or may not cause a delivery problem as well. Overall, only some addresses

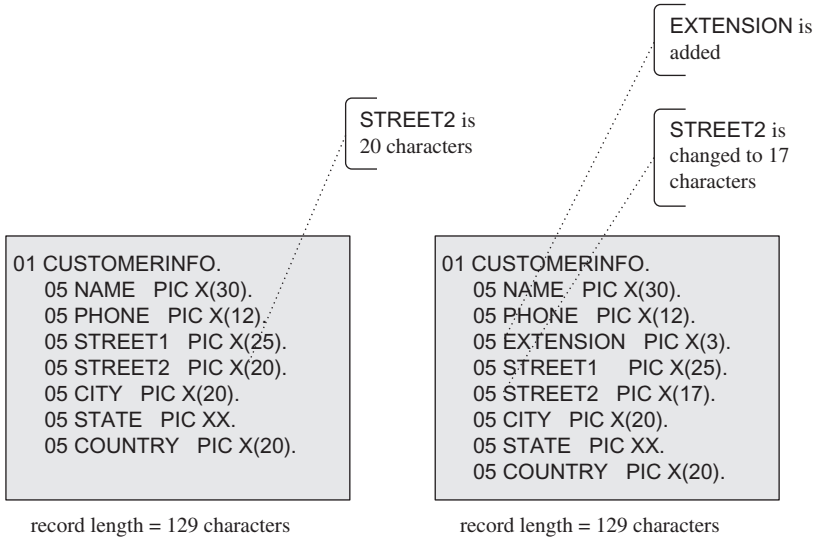

Figure 3.10 Record content changes without changing the length of the record.

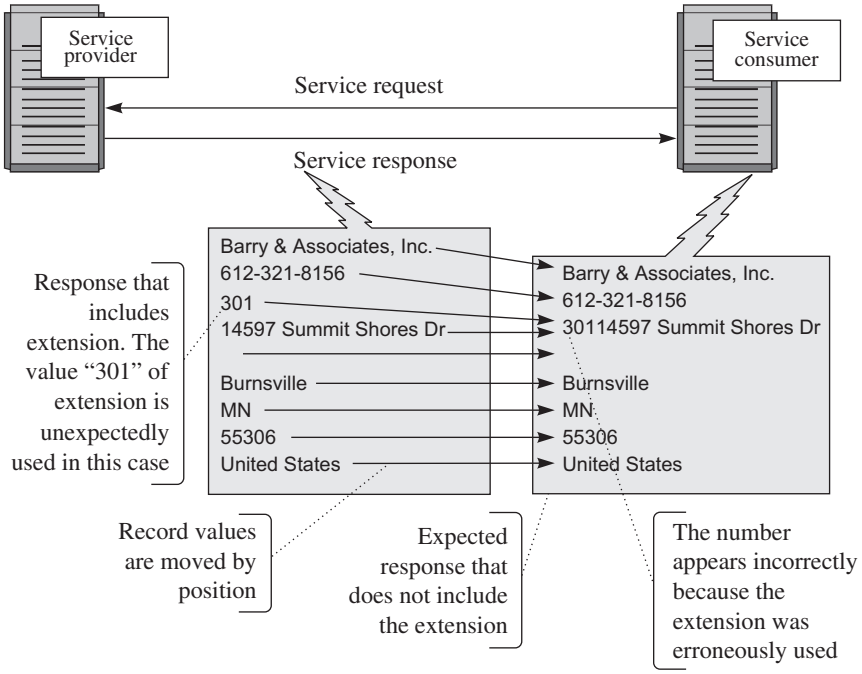

Figure 3.11 Example of the brittleness of fixed record messages.

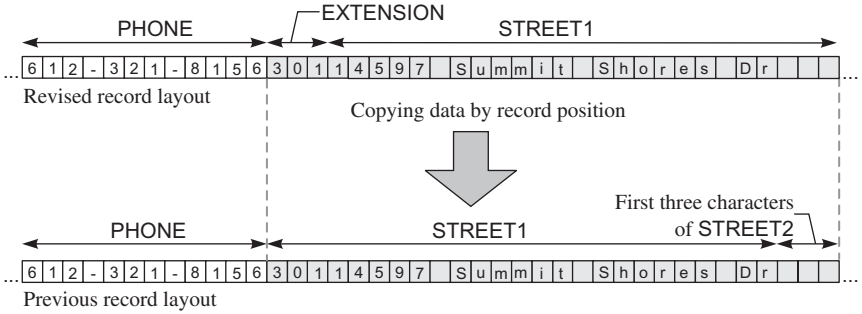

Figure 3.12 How the wrong data can be copied using fixed records.

would fail. Tracking down this type of error is often not easy. Certainly, more catastrophic errors can occur when changing the structure of fixed-length records. There could be situations where the service consumer could even fail because the record layout coming from the service provider is not the layout expected. This issue with fixed records is referred to as *brittleness*.

These types of data format changes occur all the time when exchanging data between systems, either internally or between an internal system and an external system. Using the XML tagged format makes systems more resilient in the face of such changes.

The downside of using XML is that the messages are much longer. XML messages are physically longer than fixed record messages because of the included tag information. So, there is a potential performance hit. With XML, you are trading some resilience in your systems for some reduction in performance. Nevertheless, as transmission speeds increase, this reduction in performance may not be noticed.

JSON, an XML Alternative

It is possible to use Web services without XML. JSON (JavaScript Object Notation) is one option. It uses name/value pairs instead of the tags used by XML. For example, the name "city" is paired with the value "Burnsville." This is illustrated on the right side of Figure 3.13. The name/value pairs in JSON provide the same type of resilience as the XML-tagged format for data exchanges described in the previous section. The name/value pairs do not have to be in any particular order to work.

Figure 3.13 also shows that XML and JSON can use the same vocabulary for the names of the data elements. This opportunity for standardizing on the names and the meaning of the names will be discussed later in this chapter.

When to Use SOAP, REST, JSON, or Other Options

By now, you might be wondering which option is "best" for Web services. If you are using external services, the service providers have chosen the Web service(s) they support. You will need to use whatever they have chosen. In all likelihood, your organization will use "all of the above": SOAP, REST, JSON, and whatever new Web service that is developed. Referring back to the AV analogy used earlier, the type of connections you can use between any two components is limited by the connections they can accept. The choice of Web services is no different.

XML

```
... >
  <name>Barry & Associates, Inc.</name>
  <phone>612-321-8156</phone>
  <street1>14597 Summit Shores Dr</street1>
  <street2></street2>
  <city>Burnsville</city>
  <state>MN</state>
  <postalcode>55306</postalcode>
  <country>United States</country>
< ...
```

JSON

```
{
  "name"       : "Barry & Associates, Inc.",
  "phone"      : "612-321-8156",
  "street1"    : "14597 Summit Shores Dr",
  "street2"    : "",
  "city"       : "Burnsville",
  "state"      : "MN",
  "postalcode" : "55306",
  "country"    : "United States"
}
```

Figure 3.13 Comparison of XML and JSON.

If you are developing your own service, you can choose the Web service that is best for you. The one that is best for you might be the Web service used by most in your industry or the Web service used by most services on the Internet that you are most likely to use. Be prepared, however, to use "all of the above" as mentioned before. However, there may be technical reasons that you should choose one Web service over another. The technical advantages and disadvantages of each type of Web service available are beyond the scope of this book.

The Opportunity and Importance of Standardized Semantic Vocabularies

Within an organization, it is not uncommon to find, for example, that the "account number" in one unit has the same meaning as the "customer ID" in another unit. This is often not documented and, if widespread enough, can lead to added development costs or even processing problems.

If you move to exchanging data among many organizations, the data element name and meanings can vary even more. So, the advent of Web services created an opportunity for industry groups and other organizations to establish standardized semantic vocabularies. This is because the most common means of exchanging data using Web services involves sending the name of a data element along with the value of that data element. This is the example shown earlier, where there is a data element named "city" with a value of "Burnsville." The data exchange includes both the name "city" and its value "Burnsville." XML does this using tags; JSON does it using name/value pairs.

The idea of standardizing on a semantic vocabulary also creates an opportunity for any organization to harmonize data elements among its units and with the larger world outside the organization. If, for example, the meaning of "account number" and many other names is universally understood in a given industry, it can easily minimize development costs and processing errors.

Harmonizing with industry semantic vocabularies is one way to position your organization for whatever might be coming in the future beyond Web services, service-oriented architecture, or cloud computing.

These semantic vocabularies are often referred to as *XML vocabularies*, since XML was used by the first Web services specification. A sampling of these vocabularies can be found on page 179.

Service-Oriented Architecture Explained

SOA is way to design, implement, and assemble services to support or automate business functions. SOA is not a new concept. The first SOA for many people was in the 1990s with the use of Microsoft's DCOM or Object Request Brokers (ORBs) based

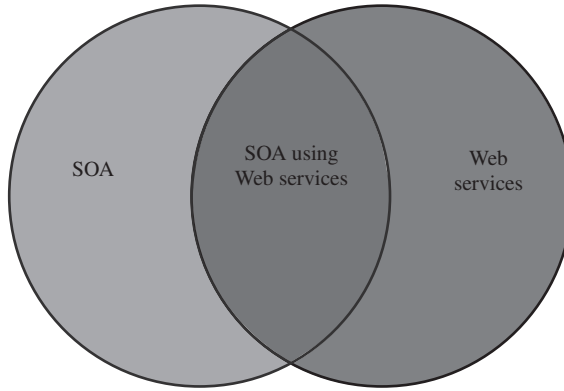

Figure 3.14 Relationship of Web services and SOA.

on the CORBA specification.[4] The basic idea goes back even further to the concept of *information hiding* that creates an interface layer above underlying systems.

Relationship of Web Services and SOA

Figure 3.14 uses a Venn diagram to illustrate the relationship between SOA and Web services. The overlapping area in the center represents SOA using Web services for connections. The nonoverlapping area of Web services represents that Web services can be used for connections, but connections alone do not make for an SOA. The non-overlapping area of SOA indicates that an SOA can use Web services as well as connections other than Web services (the original specifications of CORBA and DCOM are examples).

Identification and Design of Services

Key to SOA is the identification and design of services. The idea is that services should be designed in such a way that they become components that can be assembled in multiple ways to support or automate business functions. It is not necessarily easy to properly identify and design services. When done well, the services allow an organization to quickly assemble services—or modify the assembly of services—to

[4] See page 57 for more on CORBA and DCOM.

add or modify the support or automation of business functions. Here are basic concepts related to services:

- **Atomic service:** An atomic service is a well-defined, self-contained function that does not depend on the context or state of other services. Generally, an atomic service would be seen as *fine grained* or having a *finer granularity*.
- **Composite service:** A composite service is an assembly of atomic or other composite services. The ability to assemble services is referred to as *composability*. Composite services are also referred to as *compound services*. Generally, a composite service would be seen as *coarse grained* or having a *larger granularity*.
- **Loosely coupled:** This is a design concept where the internal workings of one service are not "known" to another service. All that needs to be known is the external behavior of the service. This way, the underlying programming of a service can be modified and, as long as external behavior has not changed, anything that uses that service continues to function as expected. This is similar to the concept of *information hiding* that has been used in computer science for a long time.

The design challenge is to find a balance between fine-grained and coarse-grained services to minimize communication overhead yet keep the services loosely coupled. Chapter 10 provides an approach for designing atomic and composite services.

Service-Oriented Architecture

So, what exactly does a service-oriented architecture look like? Let's start with a service provider. Any given service provider could provide multiple services. Multiple services are represented in Figure 3.15 by the small circles. Services are code—running on an underlying computer system—that provide computing as well as access and updates to stored data.

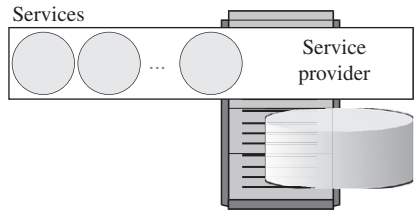

Figure 3.15 Services in a service provider.

Services are assembled to support or automate business functions. Figure 3.16 illustrates the assembly of services. This represents an SOA. Web services are used to connect the services in an SOA.

The services in an SOA can come from any service provider (which, as mentioned earlier, can also be a service consumer). So, in a given SOA, the services might be from internal systems along with any number of external systems accessible anywhere on the Internet. This is illustrated by Figure 3.17.

It is easy to imagine that you can reassemble the same services with other services to achieve a different functionality. This ability to change the assembly of services is one way that an SOA can quickly adapt to changing business needs.

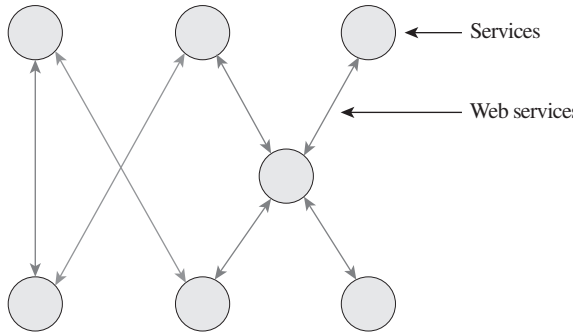

Figure 3.16 Assembly of services into an SOA.

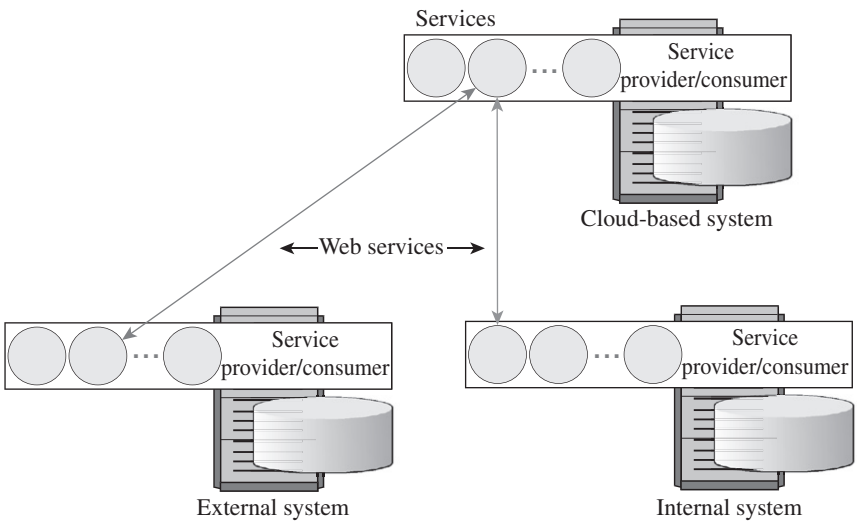

Figure 3.17 Example sources of services in an SOA.

It is also easy to imagine the number of available services quickly expanding to some unmanageable number. That is one reason why governance is important. Governance applies a structure and control over an organization's use of services. Chapter 12 will discuss governance in more detail.

Summary

This chapter outlined Web services and service-oriented architectures. It showed the importance of a robust messaging format in the use of Web services. The chapter also highlighted the importance of the ongoing standardization of the various semantic vocabularies. The chapter ended with an explanation of service-oriented architectures.

Chapter 4 will weave the concepts of service-oriented architecture into a discussion of cloud computing.