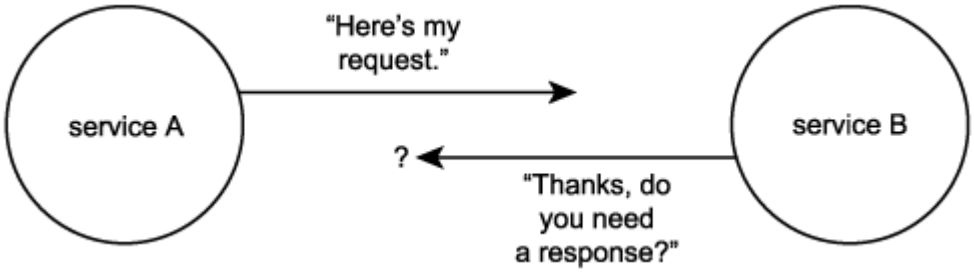
## 6.1. Message exchange patterns

Every task automated by a Web service can differ in both the nature of the application logic being executed and the role played by the service in the overall execution of the business task. Regardless of how complex a task is, almost all require the transmission of multiple messages. The challenge lies in coordinating these messages in a particular sequence so that the individual actions performed by the message are executed properly and in alignment with the overall business task (Figure 6.2).

**Figure 6.2. Not all message exchanges require both requests and responses.**




*Message exchange patterns* (MEPs) represent a set of templates that provide a group of already mapped out sequences for the exchange of messages. The most common example is a request and response pattern. Here the MEP states that upon successful delivery of a message from one service to another, the receiving service responds with a message back to the initial requestor.

Many MEPs have been developed, each addressing a common message exchange requirement. It is useful to have a basic understanding of some of the more important MEPs, as you will no doubt be finding yourself applying MEPs to specific communication requirements when designing service-oriented solutions.

### In Plain English

The Reconnect agency we discussed in the last chapter finally responds with good news. They have located and contacted Chuck. We subsequently arrange a meeting to catch up on old times.

During this reunion, I observe different types of conversations. For example, when Chuck asks Bob something, Bob typically responds back to Chuck with an answer. However, Chuck also says something to Bob after which Bob chooses not to say anything back. Then I notice Bob tell Chuck to only talk to him if Chuck has something to say that might actually interest Bob. (I begin to realize why Chuck lost touch with Bob in the first place.)

An MEP is like a type of conversation. It's not a long conversation; it actually only covers one exchange between two parties. Incidentally, each of these example scenarios represents actual message exchange patterns.

### 6.1.1. Primitive MEPs

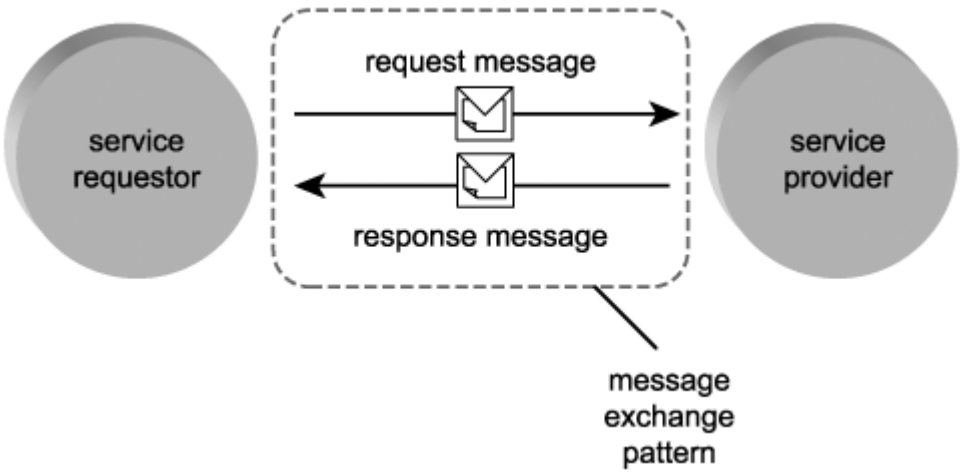
Before the arrival of contemporary SOA, messaging frameworks were already well used by various messaging-oriented middleware products. As a result, a common set of *primitive MEPs* has been in existence for some time.

#### *Request-response*

This is the most popular MEP in use among distributed application environments and the one pattern that defines synchronous communication (although this pattern also can be applied asynchronously).

The *request-response* MEP (Figure 6.3) establishes a simple exchange in which a message is first transmitted from a source (service requestor) to a destination (service provider). Upon receiving the message, the destination (service provider) then responds with a message back to the source (service requestor).

**Figure 6.3. The request-response MEP.**



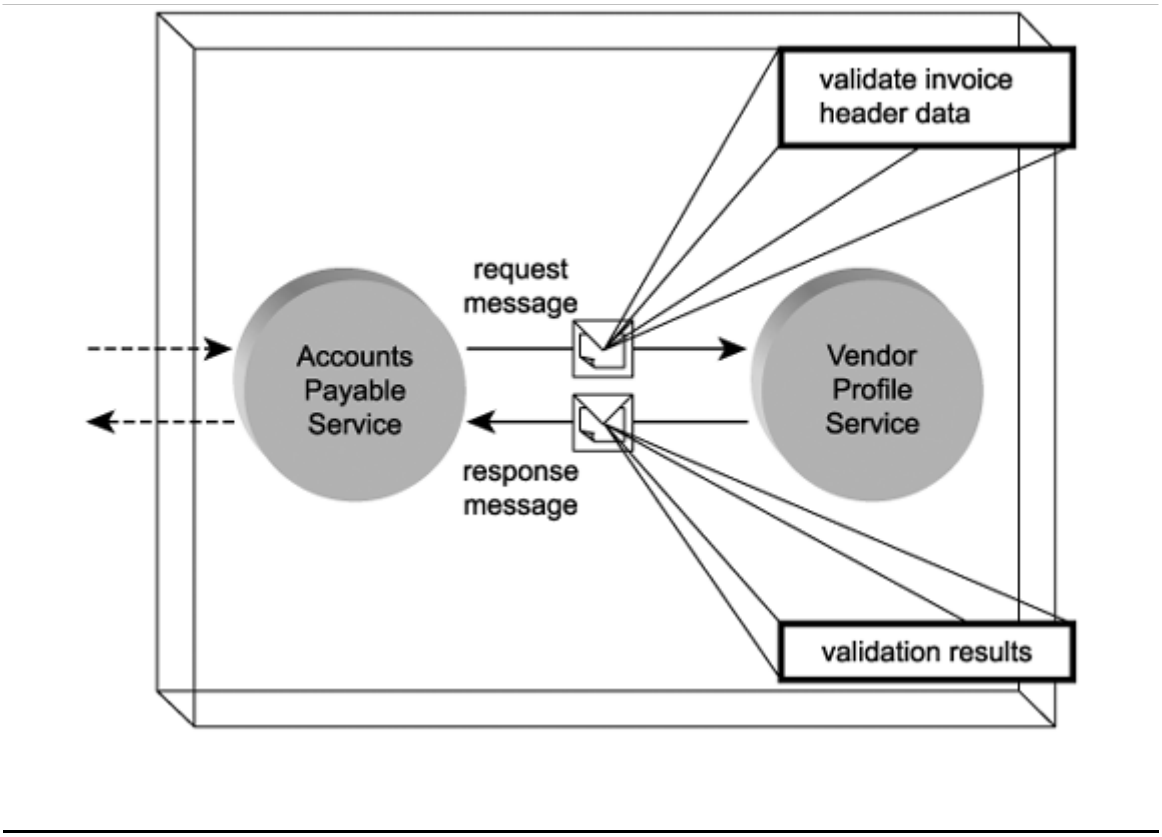

Note that within this MEP, services typically require a means of associating the response message with the corresponding request message. This can be achieved through correlation, a concept explained in Chapter 7.

## Case Study

In the *Service compositions* section of Chapter 5, we provided an example where the TLS Accounts Payable Service, upon receiving an invoice submission from a vendor, enlists the TLS Vendor Profile Service to validate the invoice header information.

The MEP used in this situation is the standard request-response pattern, where a response from the Vendor Profile Service is expected once it receives and processes the original request. The Accounts Payable Service requires a response to ensure that the header details provided in the invoice submission are valid and current (Figure 6.4). Failure to validate this information terminates the Invoice Submission Process and results in the Accounts Payable Service responding to the vendor with a rejection message.

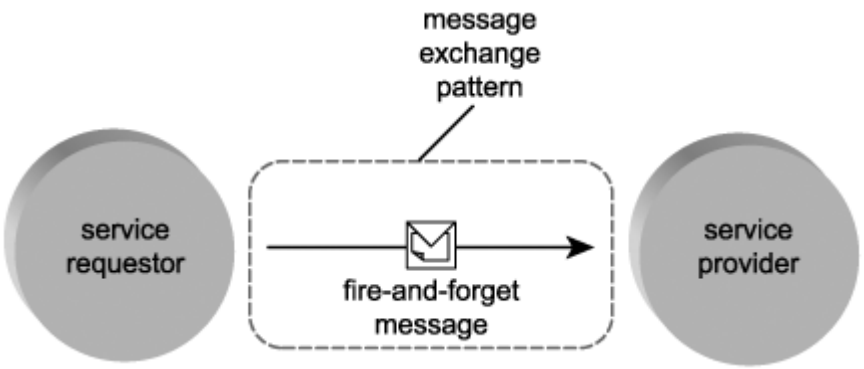
**Figure 6.4. A sample request-response exchange between the TLS Accounts Payable and Vendor Profile Services.**




### *Fire-and-forget*

This simple asynchronous pattern is based on the unidirectional transmission of messages from a source to one or more destinations (Figure 6.5).

**Figure 6.5. The fire-and-forget MEP.**




A number of variations of the *fire-and-forget* MEP exist, including:

- The *single-destination* pattern, where a source sends a message to one destination only.
- The *multi-cast* pattern, where a source sends messages to a predefined set of destinations.
- The *broadcast* pattern, which is similar to the multi-cast pattern, except that the message is sent out to a broader range of recipient destinations.

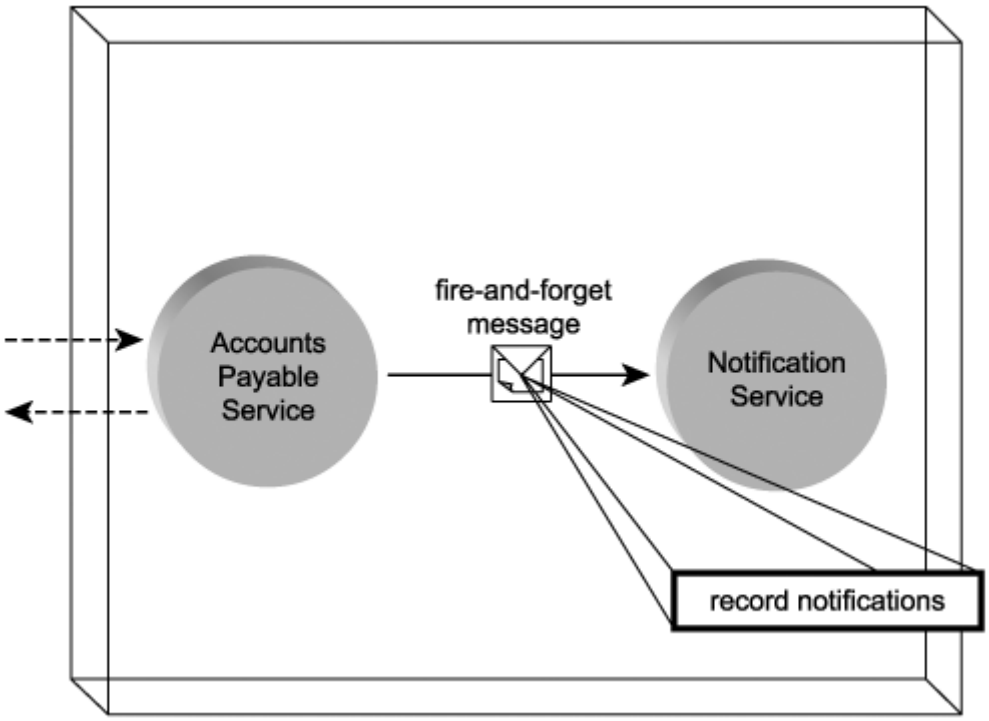
The fundamental characteristic of the fire-and-forget pattern is that a response to a transmitted message is

not expected.

## Case Study

The TLS Accounts Payable Service contains a rule that when an invoice header fails validation, an e-mail notification is generated. To execute this step, the Accounts Payable Service sends a message to the Notification Service. This utility service records the message details in a notification log database. (These records are used as the basis for e-mail notifications, as explained in the next example.) Because the message sent from the Accounts Payable Service to the Notification Service requires no response, it uses a single-destination fire-and-forget MEP (Figure 6.6).

**Figure 6.6. The TLS Accounts Payable Service sending off a one-way notification message.**




### *Complex MEPs*

Even though a message exchange pattern can facilitate the execution of a simple task, it is really more of a building block intended for composition into larger patterns. Primitive MEPs can be assembled in various configurations to create different types of messaging models, sometimes called *complex MEPs*.

A classic example is the *publish-and-subscribe* model. Although we explain publish-and-subscribe approaches in more detail in Chapter 7, let's briefly discuss it here as an example of a complex MEP.

The publish-and-subscribe pattern introduces new roles for the services involved with the message exchange. They now become *publishers* and *subscribers*, and each may be involved in the transmission and receipt of messages. This asynchronous MEP accommodates a requirement for a publisher to make its messages available to a number of subscribers interested in receiving them.

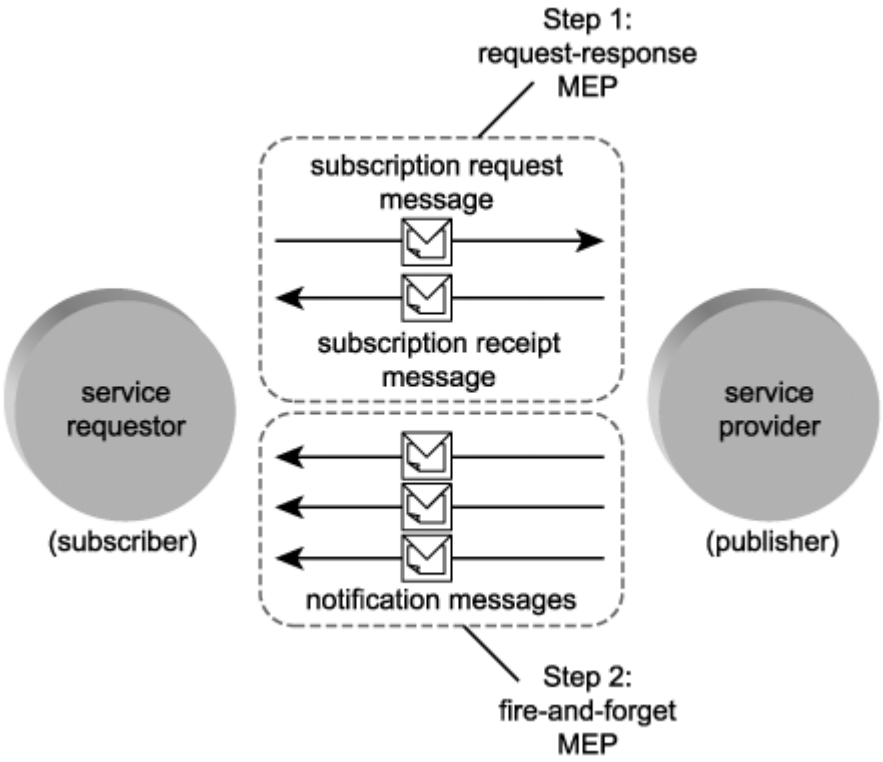
The steps involved are generally similar to the following:

| | |
|---|---|
| **Step 1.** | The subscriber sends a message to notify the publisher that it wants to receive messages on a particular topic. |
| **Step 2.** | Upon the availability of the requested information, the publisher broadcasts messages on the particular topic to all of that topic's subscribers. |

This pattern is a great example of how to aggregate primitive MEPs, as shown in Figure 6.7 and explained here:

- Step 1 in the publish-and-subscribe MEP could be implemented by a request-response MEP, where the subscriber's request message, indicating that it wants to subscribe to a topic, is responded to by a message from the publisher, confirming that the subscription succeeded or failed.
- Step 2 then could be supported by one of the fire-and-forget patterns, allowing the publisher to broadcast a series of unidirectional messages to subscribers.

**Figure 6.7. The publish-and-subscribe messaging model is a composite of two primitive MEPs.**




WS-* specifications that incorporate this messaging model include:

- WS-BaseNotification
- WS-BrokeredNotification
- WS-Topics
- WS-Eventing

Concepts relating to these specifications are discussed in the next chapter as part of the *Notification and eventing* section.

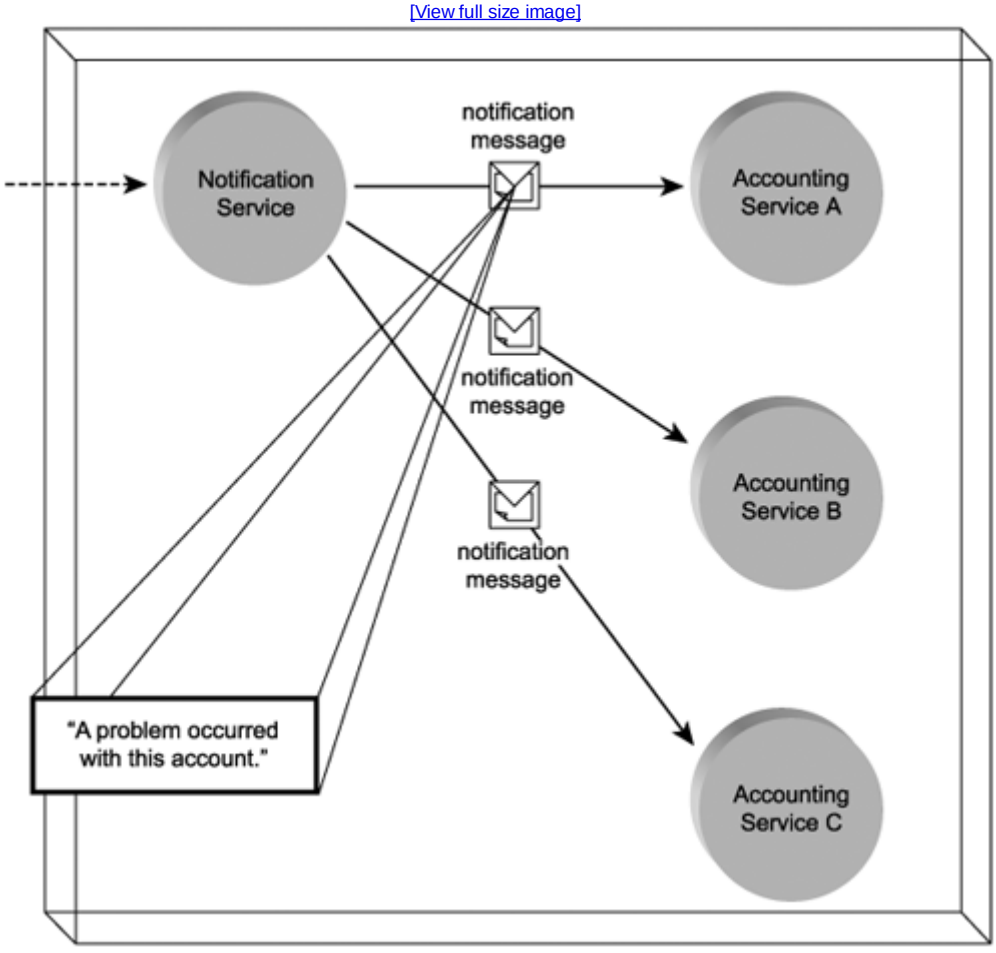
## Case Study

The utility Notification Service periodically generates and distributes notification messages for a number of different topics. Messages from outside vendors that fail validation, for example, are first logged in a dedicated notification repository. At the end of every working day, the Notification Service queries this repository to retrieve all failed submissions.

It then summarizes specific pieces of information from the query results and uses this data to populate a broadcast notification message. This message is subsequently sent to a list of subscribers consisting primarily of specialized accounting services (Figure 6.8). These services record the notification data into various profile and account records. Some pass the notification on as an e-mail to select accounting personnel.

**Figure 6.8. The TLS Notification Service notifying subscribers about a problem condition via a notification broadcast.**

[View full size image]




Several other applications that integrate with the TLS accounting system provide the means for subscribers (humans and services) to add or remove themselves from topic notification distribution lists.

### 6.1.2. MEPs and SOAP

On its own, the SOAP standard provides a messaging framework designed to support single-direction

message transfer. The extensible nature of SOAP allows countless messaging characteristics and behaviors (MEP-related and otherwise) to be implemented via SOAP header blocks. The SOAP language also provides an optional parameter that can be set to identify the MEP associated with a message. (Note that SOAP MEPs also take SOAP message compliance into account.)

### 6.1.3. MEPs and WSDL

Operations defined within service descriptions are comprised, in part, of message definitions. The exchange of these messages constitutes the execution of a task represented by an operation. MEPs play a larger role in WSDL service descriptions as they can coordinate the input and output messages associated with an operation. The association of MEPs to WSDL operations thereby embeds expected conversational behavior into the interface definition.

WSDL operations support different configurations of incoming, outgoing, and fault messages. These configurations are equivalent to message exchange patterns, but within the WSDL specification, they often are referred to simply as *patterns*. It is important to note that WSDL definitions do not restrict an interface to these patterns; they are considered minimal conversational characteristics that can be extended.

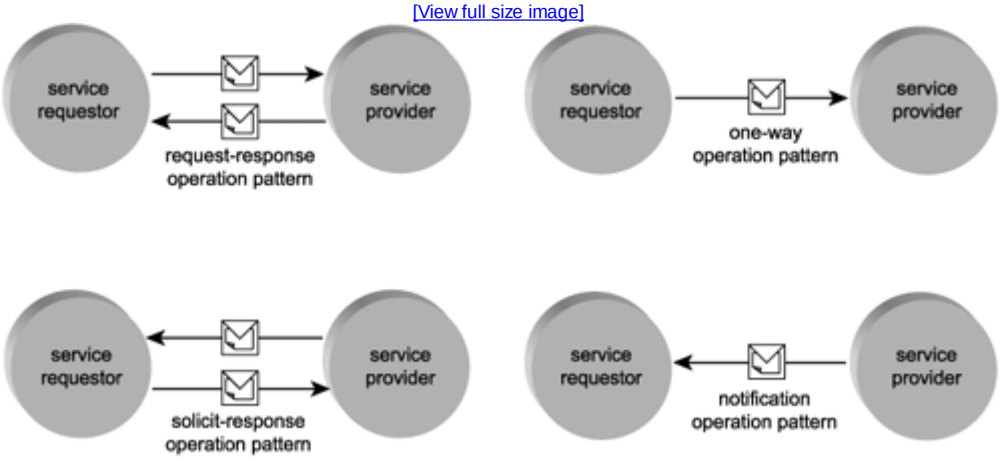
Release 1.1 of the WSDL specification provides support for four message exchange patterns that roughly correspond to the MEPs we described in the previous section. These patterns are applied to service operations from the perspective of a service provider or endpoint. In WSDL 1.1 terms, they are represented as follows:

- *Request-response operation*— Upon receiving a message, the service must respond with a standard message or a fault message.
- *Solicit-response operation*— Upon submitting a message to a service requestor, the service expects a standard response message or a fault message.
- *One-way operation*— The service expects a single message and is not obligated to respond.
- *Notification operation*— The service sends a message and expects no response.

Of these four patterns (also illustrated in Figure 6.9), only the request-response operation and one-way operation MEPs are recommended by the WS-I Basic Profile.

**Figure 6.9. The four basic patterns supported by WSDL 1.1.**

[View full size image]




Not only does WSDL support most traditional MEPs, recent revisions of the specification have extended this support to include additional variations. Specifically, release 2.0 of the WSDL specification extends MEP support to eight patterns (and also changes the terminology) as follows.

- The *in-out pattern*, comparable to the request-response MEP (and equivalent to the WSDL 1.1 request-response operation).
- The *out-in pattern*, which is the reverse of the previous pattern—where the service provider initiates the exchange by transmitting the request. (Equivalent to the WSDL 1.1 solicit-response operation.)
- The *in-only pattern*, which essentially supports the standard fire-and-forget MEP. (Equivalent to the WSDL 1.1 one-way operation.)
- The *out-only pattern*, which is the reverse of the in-only pattern. It is used primarily in support of event notification. (Equivalent to the WSDL 1.1 notification operation.)
- The *robust in-only pattern*, a variation of the in-only pattern that provides the option of launching a fault response message as a result of a transmission or processing error.
- The *robust out-only pattern*, which, like the out-only pattern, has an outbound message initiating the transmission. The difference here is that a fault message can be issued in response to the receipt of this message.
- The *in-optional-out pattern*, which is similar to the in-out pattern—with one exception. This variation introduces a rule stating that the delivery of a response message is optional and should therefore not be expected by the service requestor that originated the communication. This pattern also supports the generation of a fault message.
- The *out-optional-in pattern* is the reverse of the in-optional-out pattern, where the incoming message is optional. Fault message generation is again supported.

Until version 2.0 of WSDL becomes commonplace, these new patterns will be of limited importance to SOA. Still, it is useful to know in what direction this core standard is heading.

**Note**

Version 2.0 of the WSDL specification was originally labeled 1.2. However, the working group responsible for the new specification decided that the revised feature set constituted a full new version number. Therefore, 1.2 was changed to 2.0. However, you still may find references to version 1.2 in some places. WSDL 2.0 is not yet widely used, and details regarding this version of the specification are provided here as they demonstrate the broadening applicability of MEPs.

### 6.1.4. MEPs and SOA

MEPs are highly generic and abstract in nature. Individually, they simply relate to an interaction between two services. Their relevance to SOA is equal to their relevance to the abstract Web services framework. They are therefore a fundamental and essential part of any Web services-based environment, including SOA.

**SUMMARY OF KEY POINTS**

- An MEP is a generic interaction pattern that defines the message exchange between two services.
- MEPs have been around for as long as messaging-based middleware products have been used. As a result, some common patterns have emerged.
- MEPs can be composed to support the creation of larger, more complex patterns.
- The WSDL and SOAP specifications support specific variations of common MEPs.