Service-oriented architecture approach

The aim of this chapter is to describe an approach to developing service-oriented architectures. This approach combines the IBM Patterns for e-business with service-oriented computing concepts to solve key business problems encountered in industry today.

We use this service-oriented architecture approach to develop a solution overview for a sample business scenario. We explore the lower-level design and implementation considerations for our SOA solution in the scenario chapters, Chapter 6 through Chapter 9.
4.1 The SOA approach and Patterns for e-business

A service-oriented architecture, as described earlier, is a set of business-aligned services that are combined (composed and choreographed) to fulfill business goals. IT systems provide interfaces to these services and combine them into applications that support rapidly changing business needs. The services are manifested as a set of interfaces without any dependencies on their implementation mechanism or location. This architectural style allows better alignment of required business capabilities with IT functions.

4.1.1 Service identification

SOA has implications not just at the technology level but at the junction point between business and technology. It helps bridge that gap by bringing a more business-driven focus to how we discover and expose services. These services will often, but not necessarily always, be at the business use-case level. At this level, we are dealing with the large-grained activities that the business wants to expose, trigger, or support in a business process.

Identifying the set of services that a business requires their IT to support is not a trivial task. Analysts agree that an approach or method is needed to uncover the business-aligned services, their dependencies, and their supporting large-grained components. Service identification is a determining factor in creating and migrating to a successful service-oriented architecture.

The approach outlined in this chapter can be used with your existing application development method. It provides extensions to help you create an SOA. It augments current object-oriented methods with a set of seven steps and their corresponding outputs (artifacts or work products) that support the design of component-based, service-oriented architectures.

4.1.2 Patterns for e-business and SOA

The role of Patterns for e-business in the SOA approach is shown in Figure 4-1 on page 81. Using the Patterns for e-business, we can leverage IBM best-practices in designing and building robust, industrial strength applications.
As highlighted in Figure 4-2 on page 82, most (if not all) of the Business and Integration patterns are applicable in the context of service-oriented architectures.

The Self-Service business pattern allows users to interact with business services. The Collaboration pattern enables collaborations between business partners that partly involve service integration and partly involve workflow and people. The Extended Enterprise business pattern allows one business to interact with another businesses’ services. These Business patterns are combined on the front-end using the Access Integration pattern and on the back-end using the Application Integration pattern.
We focus on Self-Service, Extended Enterprise and Application Integration in this publication, exploring how the Business patterns, Integration patterns, Application patterns, Runtime patterns and Product mappings can be applied at the appropriate stages of the SOA approach.

### 4.2 Business scenario: Supply chain management

The Web Services Interoperability Organization (WS-I) has developed a supply chain management business scenario to demonstrate features of the WS-I Basic Profile 1.0. The WS-I sample business scenario and the technical solution overview are described in the following documents:

- WS-I Supply Chain Management Use Cases 1.0
- WS-I Usage Scenarios 1.0
- WS-I Supply Chain Management Technical Architecture 1.0

For full details, see the Web Services Interoperability Organization Web site:

http://www.ws-i.org

We use this business scenario to show how the Patterns for e-business and SOA approach can be used to develop solutions to real-world business requirements, that are based on interoperability principles defined in the WS-I Basic Profile.
As shown in Figure 4-3, this business scenario is a simplified supply chain for a consumer electronics retailer. In a typical B2C model, customers may access the retailer’s Web site, review the catalog, and place orders for products such as TVs, DVD players and video cameras.

The retailer system requests fulfilment of a consumer’s order from the internal company warehouse, which responds as to whether line items from the order can be filled. If stock for any line item falls below a minimum threshold in the warehouse, a replenishment order is sent to an external manufacturer using the B2B model.

The manufacturer does not immediately fulfil replenishment orders. It will complete the order at some later time (possibly after completing a manufacturing run).

Now, rather than taking the WS-I Supply Chain Management Technical Architecture as a given, we will apply the SOA approach that includes the Patterns for e-business to this business scenario.

4.3 Steps of the SOA approach

This section introduces a method for discovering services (top-down) and leveraging services (bottom-up, from legacy and packaged applications). It includes a combination of techniques and steps that describe how to employ
top-down and bottom-up techniques to successfully create and/or migrate to an SOA.

This method consists of the seven main steps shown in rough sequence in Figure 4-4. These activities or steps are not necessarily linear and sequential, but rather exploratory and iterative, incrementing functionality and understanding of the team as the project proceeds.

The process can be optimized by determining which steps can be carried out in parallel and which are dependent on other steps. Our experience shows that you can interleave and spin off multiple threads of activities. But there are times when you have to wait for the output of a process before you begin the next step. You will discover what works best for you and your team, depending on the project and organizational culture.

Here, you start at the top of the figure and work your way down towards implementation, test and deployment (not shown here for simplicity). Steps that are vertically parallel can be done pretty much concurrently.

The usage of the Patterns for e-business within the steps of building an SOA are also shown in Figure 4-4. The timing for applying the Patterns may vary, but should roughly be as shown.
We start the process when the business problem suggests the use of a service-oriented approach. Business problems involving exposure of back-end business services at the edge of the enterprise are typical candidates.

The top-down aspect of the SOA approach comes from taking business perspectives and models into consideration: Business functions, processes, sub-processes, and use cases are elaborated to form the outlines of component boundaries.

Components provide boundaries and containers for services (often discovered through use case analysis and goal-service model creation). Services are the units of functionality that are exposed for business process orchestration and choreography and the creation of composite applications. Components provide these services on their interfaces and will often contain finer grained object hierarchies and/or legacy systems that implement them.

In the following sections, we outline the seven key steps necessary to support component-services architectures. We apply each of the steps to the WS-I supply chain management sample scenario. The supply chain scenario is reasonably simple, so our application of the SOA approach will be at a comparable level of detail.

As already stated, the SOA approach does support legacy integration (starting at Step 1b Existing Assets Analysis in Figure 4-4 on page 84), but we do not cover this aspect in this publication.

### 4.3.1 Domain decomposition

In this step we decompose the domain into its constituent business architecture consisting of value-chain, business processes, sub-processes, and use cases.

From a business perspective, the domain consists of a set of functional areas. We decompose the domain into functional areas across the value-net; these are often good candidates for implementation as technology subsystems. Figure 4-5 on page 86 shows our domain decomposition for the supply chain management scenario.
In defining the functional areas across a value-chain or value-net we define the scope of the effort:

- Is it within the enterprise; is it across one, two or more business lines?
- Is it across a value-chain within business partners in a supply chain?

After decomposing domain into a value-chain of functional areas, we then decompose each functional area into processes, sub-processes, and business use cases.

**Use case model**

The decomposition of the functional areas described above leads to the following set of business use-cases:

- UC1: Purchase Goods
- UC2: Source Goods
- UC3: Replenish Stock
- UC4: Supply Finished Goods
- UC5: Manufacture Finished Goods
- UC6: Configure and Run Demo (Not implemented in our sample.)
- UC7: Log Events
- UC8: View Events

For a full description of the use cases, please see WS-I document *Supply Chain Management Use Case Model*, available at:

http://www.ws-i.org/

The use case model is shown in Figure 4-6 on page 87.
We can now use the business use cases (high-level, coarse-grained use cases like Purchase Goods) to further decompose the domain, as shown in Figure 4-7 on page 88. The business use cases identified are good candidates for services that will ultimately be exposed as Web services on an enterprise component. The business use case definitions are business driven and aligned, and offer a common, reusable “chunk” of business functionality.
With the business use cases, it is important to define the data to be input and output from each service or external invocation at least at a high level. These definitions are then refined during component and service specification.

As we move into design, each functional area is mapped to one or more subsystems in the architecture. Functional areas are a business notion, while the subsystems are a technology notion. There is a straightforward mapping between them, so we often find that at least one subsystem will then be identified and elaborated for each functional area. Similarly, business level use cases can be mapped to system level use cases in this stage.

In this way, each functional area or business process can be thought of as an IT subsystem that creates a natural business-driven boundary for the large-grained enterprise components that provide services.

In contrast, object-oriented analysis and design tends to produce object graphs of relatively tightly coupled finer-grained objects, which tends to impede component reuse. With the SOA approach this is avoided by identifying the larger encompassing (perhaps virtual) structure first. The constituent elements are then
refined through a more top-down approach, or allocated (if leveraging legacy, for example).

**Applying Business and Application Integration patterns**

At each stage of the SOA approach, we can leverage corresponding assets from the Patterns for e-business. We start with Business patterns to define high-level business participants and their relationships, using the functional areas and business use cases identified in domain decomposition.

Let us look at how we applied Business and Integration patterns to the WS-I SCM sample business problem. We know there are two business models associated with it:

- A B2C interaction between the consumer (end user) and the retailer’s business system; and
- A B2B interaction between the retailer’s warehouse and the external manufacturers

This implies the use of the Self-Service and Extended Enterprise business patterns. We can also use the Application Integration pattern to integrate the internal service consumers and service providers in the value-chain, and the Extended Enterprise business pattern to service consumers and service providers across enterprise boundaries. We applied these Patterns to the SCM business scenario, as shown in Figure 4-8.

*Figure 4-8  Supply chain management Business and Application Integration patterns*
Table 4-1 summarizes the linkage between the supply chain management use cases and functional areas from domain decomposition, to the Business or Integration pattern we have applied to each.

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Description</th>
<th>Invoker</th>
<th>Implemented by</th>
<th>Business or Integration pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC1: Purchase goods</td>
<td>Consumer selects and purchases goods from retailer catalog</td>
<td>Consumer (or customer)</td>
<td>Retailer</td>
<td>Self-Service</td>
</tr>
<tr>
<td>UC2: Source goods</td>
<td>Retailer system sources goods to fulfil a consumer order from warehouses</td>
<td>Retailer</td>
<td>Warehouse</td>
<td>Application Integration</td>
</tr>
<tr>
<td>UC3: Replenish stock</td>
<td>When stock in a warehouse falls below a threshold, a reorder is issued</td>
<td>Warehouse</td>
<td>Warehouse</td>
<td>Application Integration</td>
</tr>
<tr>
<td>UC4: Supply finished goods</td>
<td>Response to a replenish stock use case</td>
<td>Warehouse</td>
<td>Manufacturer</td>
<td>Extended Enterprise</td>
</tr>
<tr>
<td>UC5: Manufacture finished goods</td>
<td>Replenishment of manufacturer stock when levels fall below a specified threshold</td>
<td>Manufacturer</td>
<td>Manufacturer</td>
<td>Application Integration</td>
</tr>
<tr>
<td>UC6: Configure and run demonstration</td>
<td>Allows sample application to be run for different technical scenarios</td>
<td>Demo user</td>
<td>None</td>
<td>Self Service (not implemented)</td>
</tr>
<tr>
<td>UC7: Log events</td>
<td>Track activities performed by different system actors</td>
<td>Retailer, Warehouse, Manufacturer</td>
<td>Logging facility</td>
<td>Application Integration and Extended Enterprise</td>
</tr>
<tr>
<td>UC8: View events</td>
<td>View activities recorded in the log</td>
<td>Demo user</td>
<td>Logging facility</td>
<td>Self Service</td>
</tr>
</tbody>
</table>

### 4.3.2 Goal-service model creation

Service identification implies the discovery of business aligned services for the entire organization. As discussed, the business level use cases identified in domain decomposition are good candidates for services. In this step we create a goal-service model to test the completeness of the candidate services identified.
By interviewing business owners, querying them on the goals within the scope of work, we can create a tree of related sub-goals that are prerequisites to achieving the initial high-level, often intangible, and lofty goal. Each level of sub-goal is broken down to a set of further sub-goals until the services required to fulfill them are clear. This is called a goal-service model.

To create a goal-service model, identify the goals and sub-goals that must be realized in order support higher level goals. Then associate the sub-goals with the services required to realize the sub-goals. This will make services traceable back to the goals that the business indicates it needs to achieve. This traceability of services back to business goals is essential to ensure that the complete set of business services is anticipated as early as possible.

Various notations are possible for a goal-service model. We provide a simple example using a nested list of goals, sub-goals, and services in Figure 4-9. This example covers the retailer functional area.

---

**Goal-service model for retailer business function:**

1. Increase revenue
2. Increase sales
3. Provide self-service shopping capability
   3.1. *Purchase Goods*
   3.2. *Source Goods*
   3.3. Provide user-friendly interaction experience
      3.3.1. *Get Catalog*
      3.3.2. *Manage Shopping Cart*
4. Maintain accounting records
   4.1. *Log Events*
   4.2. *View Events*

**Key:**
- Goals are shown in regular text.
- Services are shown in *italic*.
- Value-adds (additional services needed that were not discovered during Domain Decomposition) are shown in **bold**.

*Figure 4-9  Goal-service model example*
Starting with the high-level business goal of increasing revenue, we identify lower-level subgoals and services needed to realize the business goals above. For the SCM sample, the top level goal of increasing revenue may be achieved by increasing sales. To increase sales we can provide more accessible and convenient self-service shopping capability. The Purchase Goods and Source Goods use cases identified in domain decomposition provide this capability.

Looking closer at the self-service shopping goal, it is important that interaction experience provided is user-friendly. To achieve this we need to provide the user with a shopping catalog that she can browse. In a real world scenario some sort of shopping cart facility would also be usual. This simple goal-service model helped us to identify that some additional services are needed to support the business goals. We will need to revisit the domain decomposition to incorporate the new services.

All the services identified are seen to directly support the business domain’s needs and goals. This traceability is a very significant factor in ensuring that the gap between business required services and IT system implementations remains minimal.

### 4.3.3 Subsystem analysis

After completing domain decomposition we have an idea of the functional areas in the business domain and how they interact within the value-chain for the business domain. We have a fractal (recursive or composite) view of the business domain: An extended enterprise with suppliers and customers, with individual business lines within an organization, and business processes and use cases within the business lines.

Now we move more into design- and architecture-driven decisions. Subsystem analysis refines the business use cases into system use cases that support a given business process. Subsystems are composed of business components (such as Customer, Order, and Product) and technical components (such as messaging, security, and logging).

During subsystem analysis, business and technical components are identified as follows:

- Analyze the process flow within the subsystem (often a sequence of use cases) to discover/identify candidate business components.
- Use non-functional requirements to find technical components.
- Identify the required functionality for each business component; that is, the system level use cases each component must it support.
The high-level business use cases discovered during domain decomposition are often good candidates to be placed on the interface of these subsystem components and expose the enterprise component's services. These business use cases often collaborate to support a business process.

In our SCM scenario, the four main subsystems identified are Retailer, Warehouse, Manufacturer, and Logging Facility. Each of these subsystems exposes a set of business services. For convenience, we can lump a set of finer grained services as a single callable service with several operations for the individual functions required.

Figure 4-10 shows the Retailer subsystem. After goal-service model creation, we split the Purchase Goods business use case into Get Catalog and Submit Order services.

![Retailer Enterprise Component](image)

*Figure 4-10  Business use cases are exposed on the subsystem*

Each business use case relies on a set of system use cases encapsulated in the subsystem. The subsystem leverages the business and technical components to realize the use case and support the exposed business service such as Submit Order.

The model shown in Figure 4-10 is only meant to provide a simple example of how subsystem analysis is performed. In practice, standard UML modeling techniques are used, with business use cases realized by subsystems, and system use cases realized by business and technical objects.
After completing subsystem analysis, we end up with the following larger-grained components that are being implemented as services:

- **Retailer Service** provides the functionality to access the product catalog and to place orders.

- **Warehouse Service** supports the shipment of products ordered and updates stock inventory as products ship. Where inventory falls below a threshold, it will submit a Purchase Order (PO) to a Manufacturer for finished goods.

- **Warehouse Callback Service** receives notification from a manufacturer that a PO has been processed, successful or not.

- **Manufacturer Service** accepts PO submission for finished goods and initiates the manufacturing process.

- **Logging Service** logs events as they happen and supports retrieval of events logged for display to the end-user.

At this point these coarse-grained services can be composed into an orchestration. Flow modeling tools such as IBM WebSphere Business Integration Modeler are an important aid in this area.

**Applying Application patterns**

Application patterns help structure the business and technical services needed for the SCM scenario at a high-level, as shown in Figure 4-11 on page 95.
In this simplified solution, we show the five primary services needed to support the business:

- Retailer service
- Warehouse service
- Warehouse callback service
- Logging facility service
- Manufacturer service

Notice the use of the Direct Connection application pattern between services within the Enterprise. The Direct Connection application pattern is the simplest interaction type based on a 1-to-1 topology. It allows a service consumer and provider within the organization to directly communicate with each other. For example, when a customer submits an order, the Retailer Service will then send a message to a Warehouse service to locate the ordered goods and request shipment.

The Exposed Router application pattern is applied to the business need of a Warehouse having to submit a Purchase Order to a specific Manufacturer for finished goods when the inventory falls below the pre-defined threshold limit. The retailer’s warehouse can interact with any of the three Manufacturers in the

Figure 4-11 Supply chain management Application patterns
sample application. The Router pattern applies to solutions where the source application initiates an interaction that is forwarded to, at most, one of multiple target applications. The selection of the target application is controlled by the Router tier.

When a Manufacturer has completed processing a purchase order submitted by a Warehouse, it notifies the Warehouse asynchronously via the Warehouse Callback Service of its completion. The Exposed Router pattern is used to control the services that can be invoked from outside the organization by external partners, such as the Manufacturers.

4.3.4 Service allocation

We have identified all the required services through a combination of domain decomposition and goal-service modeling. In this step, service allocation, we ensure that all the services identified have a “home” and that they are all traceable back to business goals and to components.

In this way we not only ensure that every service has business value, but also that all the services have been identified. Importantly, we are also managing the proliferation of services; like objects in the object-oriented paradigm, services tend to proliferate if not carefully managed.

Service allocation asks who (which component) will provide the implementation and management of each service. The answer will depend on whether you are a service provider or consumer.

Service consumers want the flexibility to replace an implementation based on new functional requirements, non-functional characteristics (such as higher volume handled, less down time), and economic factors (such as cheaper service).

Service providers, on the other hand, will want to implement the interface using one or more of their components or existing functionality (if not componentized).

Figure 4-12 on page 97 shows the traceability between the services identified in the goal-service model, and the new and existing components that implement and manage each service.
Service allocation is trivial in our supply chain management sample because we have only a small number of services, and we assume that there are no existing internal service providers that need to be considered. We do, however, need to allocate the responsibility to implement the Manufacturer service providers to our external manufacturer business partners.

The WS-I Supply Chain Management Sample Application Architecture document defines the following set of interfaces for the supply chain management sample:

- http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Retailer.wsdl
- http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Warehouse.wsdl
- http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl
- http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/LoggingFacility.wsdl

### 4.3.5 Component specification

After subsystem analysis, we have defined the subsystem interfaces, system use cases, business and technical components, their dependencies and flow. We have then ensured that each identified service is assigned to a component.
The next step is to develop specifications for each of the components needed. Properties, such as those shown in the template shown in Figure 4-13, need to be captured for each business or technical component that will participate in a release within the scope of the project.

**Component Specification**

- Rules
  - <rule 1>
  - <rule 2>
- Services
  - <service 2>
  - <service 1>
- Attributes
  - <data element 1>
  - <data element 2>
- Uses Components
  - <dependency on component 1>
- Variation Points
  - Pluggable Rules
  - [Configurable] Workflow
  - Configurability Requirements

*Figure 4-13  Component specification template*

### 4.3.6 Structure components and services using patterns

We have applied business and architectural analysis to identify and define the components providing the services. We have allocated services to components and developed specifications for each of the components.

We structure these components and services using patterns in this step.

**Applying Runtime patterns**

In this step, we can use Patterns for e-business Runtime patterns to establish the middleware structure needed to support the services identified, and then allocate each service to a logical middleware node. We applied the Runtime patterns shown in Figure 4-14 on page 99 to the SCM scenario.
Figure 4-14  Application of Runtime patterns during the structure components and services step

Here we have three distinct Runtime patterns for our SCM solution. Other variations of these Runtime patterns can be used to address availability and performance requirements, but are we focusing the simpler, entry-level Runtime patterns in this publication.

For the U2B interaction, we have variation 1 of the Stand-Alone Single Channel runtime pattern. This variation uses a Web server redirector containing the Web server and an application server, effectively splitting the function of a Web application server across two machines. The application server resides in the internal network to provide it with more security. The application server node will run both presentation and business logic. The Web server remains in the DMZ and serves static pages. A Web server redirector is used to forward the requests from the Web server to the application server.

For the interaction between services within the organization, the Direct Connection runtime pattern comprises of a source service accessing the target
service via a connector. Connectors provide the connectivity between two components. In this instance, we use the Direct Connection runtime pattern to implement a logical service bus, as described in “Direct Connection using a service bus” on page 65. The Exposed Direct Connection runtime pattern allows external access to the Logging service.

For the B2B interactions, the Router variation of the Exposed Broker runtime pattern is used. The Router can perform intelligent routing of messages to one target service at a time. It does not include the simultaneous distribution or decomposition capabilities that the Broker node provides. (Ideally, the Router variation of the Exposed Broker runtime pattern would also be used for external access to the Logging service. We have used the Exposed Direct Connection runtime pattern simply to demonstrate the use of this pattern.)

In Chapter 6 through Chapter 9 we examine in detail the implementation of this solution across four stages. Each stage builds on the previous stage, allowing us to work our way gradually to a comprehensive solution.

As stated earlier, valid alternate Runtime patterns are possible, include the following:

- Router instead of Direct Connection pattern for the interactions between:
  - Retailer Service and Logging Service
  - Warehouse Service and Logging Service
  - Warehouse Callback Service and Logging Service
  if there is a need for protocol conversion between the services.

- Serial Process instead of Direct Connection pattern for the Retailer-to-Warehouse service interaction if the process logic is separated from the application logic.

- Exposed Router instead of Exposed Direct Connection pattern for the Manufacturer-to-Logging service interaction. Use of the gateway to decouple the deployment and invocation of the service is a more prudent choice, especially in an inter-enterprise environment.

### 4.3.7 Technology realization mapping

Once the functionality of the services and components have been specified in detail, their implementation mechanism must be resolved. The choice of how to realize the implementation of the specification is a key architectural step.

You can build everything from scratch. Or you can out source it completely as a turn-key solution. Between these two extremes lies the most common needs of IT organizations: To decide what to build, what to buy. However, it is important to realize that these are not the only alternatives. As shown in Figure 4-15 on
page 101, there are various alternatives to the traditional “build versus buy” decision, namely:

- Build new component functionality (roll your own).
- Transform legacy to enable reuse of functionality exposed as services.
- Integrate by wrapping legacy systems.
- Buy and integrate with third party products.
- Subscribe and out source parts of the functionality, especially via Web services.

![Figure 4-15 Plan and design your service implementation carefully](image)

A service’s implementation can be realized by wrapping a legacy system with a message queue service or a Web service. But in most cases mere exposure of legacy functionality is not sufficient. Componentization of the legacy system or a small subset of the system must take place to properly expose the functionality required. This process is called transformation, where the legacy system is transformed from one state to a componentized state. A key factor is the scope of componentization: Avoiding the “boil the ocean” syndrome where all the legacy is broken into parts. Rather, an appropriate subset is selected and transformed through componentization. This is mostly an automated process. The IBM Legacy Transformation Offering focuses on these kinds of solutions.

Subscription assumes that an enterprise application Integration model has been implemented and there are services to subscribe to, in a hub and spokes architecture.

Once we have specified the components that we need, we can weigh the advantages and disadvantages of how to realize the components. This mapping is called a technology realization.

A table like Table 4-2 on page 102 can be created for this purpose.
Table 4-2  Technology realization mapping table

<table>
<thead>
<tr>
<th>Business process</th>
<th>Sub-process</th>
<th>Use case /service</th>
<th>Component</th>
<th>Current Implementation (if exists)</th>
<th>Future Technology Realization mechanism</th>
</tr>
</thead>
</table>

In our simple SCM example we move straight from the Runtime patterns identified in “Structure components and services using patterns” on page 98 to applying the Product mappings.

**Applying Product mappings**

The Patterns for e-business Product mappings can be used with component realization to help you identify and select proven and tested middleware implementations for your scenario. We applied the Product mappings shown in Figure 4-16 on page 103 to the SCM scenario.
We use IBM WebSphere Application Server V5.1 to host all the services in our SCM solution. WebSphere hosts the front-end, user interface application and each of the services implemented:

- Retailer service
- Warehouse service
- Warehouse callback service
- Logging facility service
- Manufacturer service

In Chapter 6, “HTTP service bus” on page 159, we look in detail at how the service consumers and providers in the scenario can be integrated using the SOAP/HTTP Web services support provided with IBM WebSphere Application Server V5.1 and the Direct Connection runtime pattern.
In Chapter 7, “JMS service bus” on page 229, we look at adding reliable messaging using SOAP/JMS Web services. We examine how IBM WebSphere Application Server V5.1 service consumers and providers can be reconfigured to communicate using the IBM WebSphere MQ JMS provider, instead of HTTP.

In Chapter 8, “Service directory” on page 251, we add the UDDI Registry provided with IBM WebSphere Application Server Network Deployment V5.1.

In Chapter 9, “Web service gateway” on page 279, we include the Web Services Gateway provided with IBM WebSphere Application Server Network Deployment V5.1 to allow integration between the Warehouse and Manufacturer services using the Broker runtime pattern.

**Note:** When integrating between J2EE application servers, RMI/IIOP is generally the preferred approach. The intention of the SCM product mapping is to demonstrate that WebSphere V5.1 can be used to implement all of the components of the service-oriented solution, and to demonstrate WebSphere conformance with all of the WS-I Basic Profile 1.0 features covered by the SCM sample.

### 4.4 Summary and conclusion

The method described here helps map a current business model and its underlying IT architecture to a service-oriented enterprise and application architecture. The combination of SOA and Patterns for e-business produces a powerful set of best-practices and design decisions that help expedite mapping of business processes to IT and the creation of a robust service-oriented architecture that is in line with the needs of the business.

The SOA approach discussed here is an extension that can be easily added to your current method. It combines a top-down and bottom-up approach to the modeling, design and implementation of a service-oriented architecture for on-demand computing.

### 4.5 Where to find more information

For more information on topics discussed in this chapter, see:

- The following Web Services Interoperability Organization documents:
  - *WS-I Basic Profile 1.0*
  - *WS-I Supply Chain Management Use Cases 1.0*
  - *WS-I Usage Scenarios 1.0*
WS-I Supply Chain Management Technical Architecture 1.0

WS-I documents are available at:
http://www.ws-i.org


Technology options

This chapter describes some of the Web services related technology options available to you when developing a service-oriented architecture. You can use the information in this chapter, along with pointers to other sources, as input to your architectural decision making when considering Runtime patterns and Product mappings for the application patterns you choose to deploy.

In “A closer look at service-oriented architecture” on page 24 we presented an architectural stack and the elements that might be observed in a service-oriented architecture. We use this stack to structure our discussion of Web service technology options, as shown in Figure 5-1 on page 108.
The industry is working quickly to develop the additional standards that are required to simplify the implementation of service-oriented architectures. Brief descriptions are provided in this chapter for both current and emerging standards for each level of the architecture stack. For more information, see the IBM developerWorks section on Web services standards:

http://www.ibm.com/developerworks/views/webservices/standards.jsp
5.1 Introduction

This chapter describes Web services technologies relevant to service-oriented architecture, including some of their advantages and disadvantages. For an overall perspective, some of the pluses and minuses of Web services are included in this section. It is not recommended that Web services are seen as the solution to all your integration problems, and should not be considered as the best architectural approach for all future solutions. Just as with any other technology or architectural approach, there are inherent advantages of using Web services in the right place and for the right reasons.

5.1.1 Advantages of Web services

When appropriately selected and implemented, the use of Web services technologies can enable a business to:

- Deliver new IT solutions faster and at lower cost by focusing their code development on core business, and using Web services applications for non-core business programming.
- Protect their investment in IT legacy systems by using Web services to wrap legacy software systems for integration with modern IT systems.
- Integrate their business processes with customers and partners at less cost. Web services make this integration feasible by allowing businesses to share processes without sharing technology. With lower costs, even small businesses will be able to participate in B2B integration.
- Enter new markets and widen their customer base. Web services listed in UDDI registries can be “discovered” and thus are “visible” to the entire Web community.

5.1.2 Disadvantages of Web services

All new technologies have problems and disadvantages that should be taken into consideration before they are used. To try and assist you with identifying if Web services is appropriate or not, the following list gives some issues you should consider when selecting to use Web services:

- Binding to Web services dynamically requires that the contents of the UDDI registry be trusted. Currently, only private UDDI networks can provide such control over the contents.
- The SOAP server footprint is significant and the technology is relatively new, so adding the Web service provider stack to existing enterprise systems can be a problem.
Standards for integration of business processes, management of transactions, and the awareness of the policies of interchanging partners are all still under development. To realize the promise of Web services, these types of standards should be available in implementation products.

5.2 Transport

The transport layer in our architectural stack, as shown in Figure 5-2, is related to the mechanisms used to move service requests from the service consumer to the service provider, and service responses from the service provider to the service consumer. There are a number of standards in use today for Web services, but the most common one is HTTP.

Figure 5-2  The transport layer

5.2.1 HTTP

The HTTP protocol is commonly used for the transport of service requests and responses. HTTP extensions and HTTP/1.1 are stable specifications used as the standard transport protocol of the World Wide Web.

The World Wide Web Consortium (W3C) has closed the HTTP Activity, as they have achieved the goal of creating a successful standard. The W3C has started
a new activity to extend the XML protocol. The XML Protocol Working Group will define new HTTP bindings for XML, as a higher-level protocol.

**Advantages of HTTP**
There are a number of advantages of using HTTP as a transport for Web services interactions, including:

- HTTP is a widely adopted protocol. Any organization with a Web server has implemented HTTP, and any client using a Web browser uses HTTP. Therefore the HTTP infrastructure is widely available.

- The HTTP protocol is open and deployed on many different system types, including non-traditional computing devices such as PDAs.

- Most enterprises allow HTTP to travel freely through protocol firewalls. Therefore, there are fewer barriers to extended enterprise use of HTTP as a transport for Web services.

**Disadvantages of HTTP**
HTTP is a light-weight and stateless protocol that was not originally designed to carry application data. Some of the disadvantages of using it for Web services include:

- The protocol is stateless. If any state data is required to maintain an application session, the applications must create and manage the state data.

- HTTP is not a reliable protocol. If reliable delivery of application data is required, the application must either:
  - Develop a reliability framework, such as exchanging receipt messages.
  - Use a more reliable protocol.

### 5.2.2 Java Message Service

Messaging middleware is a popular choice for accessing existing enterprise systems in an asynchronous manner. Messaging communication is loosely coupled, as compared to tightly coupled technologies such as Remote Method Invocation (RMI) or Remote Procedure Calls (RPC). The sender does not need to know anything about the receiver for communication. The message to be delivered is sent to a destination (queue) by a sender component, and the recipient picks it up from there. Moreover, the sender and receiver do not both have to be available at the same time to communicate. Messaging is one of the options if you are implementing a solution based on the Message variation of the Direct Connection pattern in an intra-enterprise scenario.
Messaging middleware may be an appropriate transport protocol when there is a requirement for Web services to communicate:

- *Asynchronously*, where the sender of a message does not wait for a reply to the message
- *Reliably*, where the sender is assured that the message will be delivered

A standard way for using messaging middleware from a Java application is using the Java Message Service (JMS) interface. JMS offers Java programmers a common way to create, send, receive and read enterprise messages. The JMS specification was developed by Sun Microsystems with the active involvement of IBM, other enterprise messaging vendors, transaction processing vendors, and RDBMS vendors.

JMS has two messaging styles, or in other words, two domains:

- One-to-one, or point-to-point model
- Publish/subscribe model

**IBM JMS implementations**

IBM provides two implementations of JMS:

- A JMS provider included with WebSphere Application Server V5.0. This can be used for asynchronous communication between applications running on WebSphere V5.0 servers.
- IBM WebSphere MQ V5.3 includes built-in JMS Provider support with enhanced performance features for integrating JMS applications with other applications. IBM WebSphere MQ takes care of network interfaces, assures once and once only delivery of messages, deals with communications protocols, dynamically distributes workload across available resources, and handles recovery after system problems. IBM WebSphere MQ is available for most popular operating system platforms.

**Advantages of JMS**

There are a number of advantages of using JMS as a transport for Web services interactions, including:

- JMS provides a more reliable transport than alternatives, such as HTTP.
- Asynchronous requests can be readily deployed.
- It leverages existing, enterprise-proven messaging systems.
Disadvantages of JMS

Although JMS is an open standard for Java-based systems, the actual transport system must be provided by a software product. Therefore, there are several considerations, including:

- The communicating Web services must have access to JMS providers that can communicate with each other. Generally, this implies the same product must be installed. For example, both systems must have IBM WebSphere MQ installed.
- JMS is a Java-based standard and is not as readily accessible to systems that are not based on Java.

5.2.3 Simple Mail Transfer Protocol

Simple Mail Transfer Protocol (SMTP) is one of the protocols that has made the Internet extremely popular, and is the basis of Internet e-mail. Its objective is to transfer electronic mail reliably and efficiently between people. It is then different from other messaging protocols like IBM WebSphere MQ or JMS which have been designed to handle information transfer between programs.

SMTP is independent of the particular transmission subsystem and requires only a reliable ordered data stream channel. An important feature of SMTP is its capability to relay mail across transport service environments. It has been published originally as RFC 821 by the Internet Engineering Task force (IETF) in 1982. Since then new RFCs like RFC 2821 have enhanced the original standard to take into account new technical capabilities.

As a widely used messaging standard, SMTP is a potential transport for SOAP messages. The writers of the W3C SOAP 1.1 Note state that “SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework”. The W3C SOAP 1.1 Note is available at:

http://www.w3.org/TR/soap/

Most implementations currently use HTTP to transport SOAP messages, but there is no reason why you cannot use other layers, such as SMTP.

Apache SOAP SMTP transport

Apache SOAP has been included as a WebSphere SOAP engine since WebSphere V4.0. It has now been superseded by Apache Axis and by Web Services for J2EE.
The Apache SOAP distribution includes classes which permit the servicing of SOAP requests using e-mail. It does this using a combination of SMTP and POP. A class called SMTP2HTTPBridge must be running in a separate JVM on the server. As the name suggests, this class operates as a bridge, mapping requests between HTTP and SMTP. Strictly speaking, this is not an independent SMTP transport; what it really does is convert e-mail SOAP messages to and from HTTP SOAP messages.

SMTP and Web Services for J2EE
The Java API for XML Messaging (JAXM) Optional Package enables applications to send and receive document-oriented XML messages using a pure Java API. JAXM implements SOAP 1.1 with Attachments messaging so that developers can focus on building, sending, receiving, and decomposing messages for their applications instead of programming low level XML communications routines. The specifications ensure that message delivery can be accomplished by supporting a number of communications infrastructures and key networking transports including, but not limited to, HTTP(S) and SMTP. However, we have not been able to find a reference to an implementation on SMTP.

JAXM adds support for plugging-in higher level messaging protocols like ebXML. JAXM is not a full-fledged messaging API like JMS.

J2EE also provides a mail API. The JavaMail 1.3.1 API defines a set of abstract classes that model an e-mail system. The API provides a platform independent and protocol independent framework to build Java technology-based mail and messaging applications. The JavaMail API is implemented as a Java platform optional package and is also available as part of J2EE. The reference implementation includes the core JavaMail packages and IMAP, POP3, and SMTP service providers.

Where to find more information
For more information on SMTP, see:
- The SOAP Protocol:
  [http://www.w3.org/TR/SOAP/](http://www.w3.org/TR/SOAP/)
- Apache SOAP:
- SMTP:
  [http://www.ietf.org](http://www.ietf.org)
5.2.4 HTTPR

Reliable HTTP (HTTPR) was a proposed protocol for offering the reliable delivery of HTTP packets between a server and a client, but was withdrawn in favor of the OASIS evolving WS-ReliableMessaging standards. The HTTPR proposal offers reliability by adding extensions to HTTP, whereas WS-ReliableMessaging, as covered in further detail in the following section, delivers reliability at the SOAP envelope level.

5.2.5 Emerging standards for transport

Commonly used transport protocols for Web services are currently dominated by HTTP, for reasons discussed in “Advantages of HTTP” on page 111. However, for enterprises to depend on Web services for business critical processing, more reliable transport protocols are required. Some emerging work in this area is discussed in this section.

WS-ReliableMessaging

As discussed in “Disadvantages of JMS” on page 113, current reliable message transport mechanisms require communicating parties to be using the same infrastructure, such as IBM WebSphere MQ. The WS-ReliableMessaging draft standard has been developed to provide a framework for interoperability between different reliable transport infrastructures.

The draft standard was released in March 2003 by IBM, BEA, Microsoft, and TIBCO. The protocol defined in the standard provides four delivery assurance descriptions that must be implemented by partners in the communication:

- AtMostOnce
- AtLeastOnce
- ExactlyOnce
- InOrder

The WS-ReliableMessaging draft standard uses WS-Policy (see 5.8, “Policy” on page 144) and associated standards as a framework for determining the capabilities and requirements of partners in a reliable messaging exchange. The authors also strongly recommend that communication be secured using WS-Security and associated standards (see 5.9.4, “Emerging standards for security” on page 151).

Where to find more information

For more information on WS-ReliableMessaging, see:

BEEP

Blocks Extensible Exchange Protocol (BEEP) is a generic application protocol framework for peer-to-peer asynchronous interactions over a TCP/IP connection. Unlike HTTP, BEEP does not have a notion of a server or client, and rather initiates a message-based communication session when a requestor initiates a request for connection with a provider.

Standardized by the Internet Engineering Task Force (IETF), BEEP supplies a protocol framework to manage peer-to-peer connection, authentication, message transport and error handling. But all this comes at a cost, and as such BEEP does not lend itself for communications that could be categorized as one-shot such as DNS look-up or tightly coupled RPC protocols like NFS. The appropriate environment for the use of BEEP is when you require an application protocol framework that is:

- Connection-oriented: Any BEEP based communication is expected to be initiated and disconnected for each interaction. In other words BEEP is expecting your application to connect, undertake the required task, and disconnect.

- Message-oriented: Just as with the fundamental concepts of SOAs, applications using BEEP to send data will do so based on well defined and structured data, giving the ability for the applications to be loosely coupled with limited knowledge of each others’ implementation.

- Asynchronous: BEEP, unlike HTTP, is a peer-to-peer style communication framework, which does not restrict the interactions to be in a particular order.

Where to find more information

For more information on BEEP specifications, see:

http://www.beepcore.org

5.3 Service communication protocol

The service communication protocol layer of our architectural stack, as shown in Figure 5-3 on page 117, describes and defines the technologies and standards required to supply a transport mechanism between integrated services. If we consider the transport layer of the stack (discussed in the previous section) to be represented by a road between two end points, then the service communication protocol layer would be the vehicles traveling on the road, facilitating the transport of a package between the package sender and the receiver.
5.3.1 SOAP

Simple Object Access Protocol (SOAP) is a network-, transport-, and programming language-neutral protocol that allows a service consumer to call a remote service provider. The message format is XML. The currently adopted standard is W3C’s SOAP 1.1 specification, while SOAP 1.2 is in the review process.

SOAP has the following characteristics:

- SOAP is designed to be simple and extensible.
- SOAP provides a framework to describe message content and process instructions, and an optional set of encoding rules for representing defined data-types.
- All SOAP messages are encoded using XML.
- SOAP is transport protocol independent. HTTP is one of the supported transports. Hence, SOAP can be run over an existing Internet infrastructure.
- There is no distributed garbage collection. Therefore, call by reference is not supported by SOAP; a SOAP client does not hold any stateful references to remote objects.
SOAP is operating system independent and not tied to any programming language or component technology. It is object model neutral.

Due to these characteristics, it does not matter what technology is used to implement the service consumer, as long as the consumer can issue XML messages. Similarly, the service provider can be implemented in any language, as long as it can process XML messages.

As shown in Figure 5-4, a simple SOAP message consists of three main parts: Envelope, optional header(s) and a body.

![Figure 5-4  Overview of a SOAP message](image)

**SOAP with Attachments**

The SOAP architecture is based on XML documents. However, some issues arise when these documents contain binary data (such as images) or encapsulate other XML documents. To handle these situations, the SOAP standard has been enhanced with the SOAP with the Attachments feature. This feature allows SOAP messages to be composed of several parts to improve the handling of specific payloads.

**SOAP encoding and performance**

There are several ways to encode messages in SOAP messages.

- SOAP Remote Procedure Call (RPC encoding), as defined as the SOAP 1.1 chapter 5 specification
- SOAP Remote Procedure Call Literal encoding (SOAP RPC-literal), which uses a user-defined method to marshal and unmarshal the XML data
- SOAP document-style encoding, also known as message-style or document-literal encoding
These techniques bring different benefits and limitations. The choice of a encoding technique for a particular scenario may be a critical success factor for a given project.

Figure 5-5 shows the expected benefits of each technique.

![Positioning of SOAP encoding techniques](image)

**SOAP and the WS-I Basic Profile**

The WS-I Basic Profile 1.0 (see “WS-I Basic Profile 1.0” on page 36) precludes the use of SOAP encoding. SOAP encoding is used to indicate the use of a particular scheme in the encoding of data into XML. This introduces complexity. It has proven to be a frequent source of interoperability problems.

The WS-I Basic Profile 1.0 therefore requires use of either the RPC/literal or Document/literal forms of the WSDL SOAP binding. Considerable detail is provided in the specification describing correct use of the SOAP binding extension elements, to ensure a consistent and interoperable description of the RPC/literal and Document/literal forms of the SOAP binding. The aim is that WSDL tools will generate code that will be interoperable with regards to the SOAP messages produced and/or consumed.

**SOAP and reliable messaging services**

The SOAP standard is independent of any transport. However, the only binding that is used as a reference implementation is HTTP. This means that SOAP does not yet have a standard binding for reliable messaging. Several vendors offer reliable messaging solutions, with the IBM offering based on the WebSphere MQ family of middleware.

Another environment sometimes considered in this context is ebXML (see 5.4.3, “ebXML” on page 125). The ebXML messaging service has the same objectives as the WebSphere MQ offering, but implementations are still very recent, as the standard was only published in 2001 as part of the global ebXML architecture.
Accessing CICS via SOAP
Until a recent enhancement to the mainframe CICS Transaction Server, programmers had to write Java programs accessing the CICS functions via a J2EE connector like the CICS Transaction Gateway. Then the Java programs could be exposed as Web services.

The new IBM SOAP for CICS feature enables programmers to access CICS transactions directly via SOAP calls. The aim of this new feature is to provide more flexibility for accessing legacy business functions. It is intended to provide an additional form of connectivity appropriate for some applications, especially those used within service-oriented or extended enterprise architectures.

5.4 Service description
One of the main benefits of Web services is to allow for loosely coupled architectures. To achieve that goal, the service provider and the service consumer should be as independent as possible. A structured service description, highlighted in the architectural stack in Figure 5-6, is key to enabling that independence. Services can be provided without the need for the provider or the consumer to care about the other’s technical platform or programming language.
There are two levels of service description:

- Operational service description (XML and WSDL)
- Complete service description

### 5.4.1 XML

XML is the de facto syntax used to exchange message data between the Web service consumers and providers. It allows for a customized markup language with tags defined in a Document Type Definition (DTD) or XML Schema.

DTD is inherently flawed, as it has limited data typing and cannot support date formats, numbers or other common data types. It uses its own language to define XML syntax which is not XML specification compliant and hence makes it difficult to manipulate a DTD programmatically.

To solve these problems, the World Wide Web Consortium (W3C) defined a new standard to define XML documents called XML Schema. XML Schema provides the following advantages over DTDs:

- Strong typing for elements and attributes
- Standardized way to represent null values for elements
- Key mechanism that is directly analogous to relational database foreign keys
- Defined as XML documents, making them programmatically accessible

XML is not a prerequisite for defining messages. Other formats such as OMG’s Interface Definition Language (IDL) or a simple fixed record format could be used instead. However, the parties must agree on what format to use. Where many organizations are involved, managing numerous non-standard message formats would be cumbersome. Hence XML is gaining wide acceptance as the standard message format. Industry-specific vocabularies have also been developed in accounting, construction, education, finance, government, health care, insurance, legal, manufacturing, telecommunications and travel (to name a few!) to facilitate communication within each industry.

XML allows for the representation of data in a standard and structured format. It provides the syntax of a language but it does not convey the meaning associated with the data. Various industries may use the same word differently and they may have different words that mean the same thing.

OASIS has developed the Universal Business Language (UBL) in an effort to define a common XML business document library. UBL will provide a set of XML building blocks and a framework that will enable trading partners to unambiguously identify and exchange business documents in specific contexts.
Transformation of XML documents
With XML's flexibility in the development of different vocabularies, there is a need to be able to transform one XML format to another.

Two W3C specifications that are part of the Extensible Stylesheet Language (XSL) family are used for transforming XML documents into other XML documents:

- *Extensible Stylesheet Language Transformations (XSLT)* is the language for transforming XML. It defines the set of rules used in the transformation from a source tree into a target tree. A transformation defined in XSLT is called a stylesheet.
- *XML Path Language (XPath)* is an expression language used by XSLT to access or refer to parts of an XML document.

An XSLT processor is used for the actual transformation and typically has a performance overhead, so online processing of larger documents can be slow, although the use of XSL just-in-time compilers may speed up the transformation time.

With XSLT, XML-based applications can be linked to Web services. It can convert an XML document into a different XML format such as WSDL and SOAP. It can also transform the logical structure into a presentation format such as HTML pages.

Advantages of XML
There are many advantages of XML in a broad range of areas. Some of the factors that influenced the wide acceptance of XML are:

- **Acceptability of use for data transfer**
  XML is a standard way of putting information in a format that can be processed and exchanged across different hardware devices, operating systems, software applications, and the Web.

- **Uniformity and conformity**
  XML gives you an common format that could be developed upon and is accepted industry-wide.

- **Simplicity and openness**
  Information coded in XML is human readable.

- **Flexible and extensible**
  The tag-based format of XML makes it flexible and easily extendable. It can be customized to support an organization's needs. When a new piece of information is required, a tag is simply added to the structure. There is no dependency on the position of the information in the structure, unlike a fixed
record format. An application that is unaware of this new information in the structure is not affected by the extra data.

- **Separation of data and display**
  The representation of the data is separated from the presentation and formatting of the data for display in a browser or other device.

- **Industry acceptance**
  XML has been accepted widely by the information technology and computing industry. Numerous tools and utilities are available, along with new products for parsing and transforming XML data to other data, or for display.

### Disadvantages of XML

Some XML issues to consider are:

- **Complexity**
  While XML tags can allow software to recognize meaningful content within documents, this is only useful to the extent that the software reading the document knows what the tagged content means in human terms, and knows what to do with it.

- **Standardization**
  When multiple applications use XML to communicate with each other they need to agree on the tag names they are using. While industry-specific standard tag definitions often do exist, you can still declare your own non-standard tags.

- **Large size**
  XML documents tend to be larger in size than other forms of data representation. This has performance implications and may be unsuitable for high-performing systems.

### 5.4.2 WSDL

The W3C has adopted the Web Services Description Language (WSDL) as a standard for base-level service description. At the time of writing this book, the current version of WSDL is 1.1.

WSDL specifies the operational characteristics of a Web service using an XML document. It provides a notation to answer the following questions:

- What (is this service about)?
- Where (does it reside)?
- How (can it be invoked)?
A Web service is considered as a set of endpoints operating on messages containing either document-oriented or procedure-oriented (RPC) messages. WSDL offers a standard way to abstractly describe the operations and messages: The service interface definition. These descriptions are bound to a specific network protocol and message format to create an endpoint: The service implementation definition. Figure 5-7 shows this combination.

![Diagram of WSDL components](image)

The service interface definition can be instantiated and referenced by multiple service implementations.

A WSDL document is defined in one or more physical files according to the implementation. When several files are used an import element is used to link each file. The WSDL is generally divided into an implementation WSDL file and an interface WSDL file. The binding may be defined in a separate, third file.

WSDL does not prescribe any specific message format or network protocol. However, the WSDL 1.1 specification only describes bindings for SOAP 1.1 over HTTP, direct HTTP 1.1 request (HTTP GET and POST), and Multipurpose Internet Mail Extensions (MIME).

**Advantages of WSDL**

As a fundamental requirement for the implementation of Web services, WSDL is required to publish the interface description contract for other services to invoke upon.
Disadvantages of WSDL
The WSDL document does not provide some of the information a potential user may require, such as:

- Who provides the service?
- What kind of business provides the service?
- What the other services are available from this provider?
- What quality of service should be expected from this provider?
- Is the service free or fee-based?

The UDDI standard provides a potential source of this information, in order to build a more complete view of a given service. See “UDDI” on page 141.

5.4.3 ebXML

ebXML stands for Electronic Business using XML. It provides a modular suite of specifications that enables enterprises to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes.

It is a joint development effort between Organization for the Advancement of Structured Information Standards (OASIS) and the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT). OASIS (formerly known as SGML group) has brought XML expertise, while UN/CEFACT, who was the main sponsor of Electronic Data Transmission (EDI), has brought business expertise.

Figure 5-8 on page 126 presents a simplified view of the main components of the ebXML architecture.
The major specifications in the ebXML suite are (working our way up from the bottom left of Figure 5-8):

- **Reliable messaging with ebXML Message Service Specification (ebMS):**
  Provides guaranteed, once-only delivery, secure, SOAP-based communication.

- **Partner profile and agreements with ebXML Collaboration Protocol Profile and Agreement (ebXML CPP/A):**
  Describes an organization, its services, business processes and technical abilities. It holds configuration information for partners' runtime systems and stores quality-of-service (QOS) information.

- **Business process specifications with ebXML Business Process Specification Schema (ebXML BPSS):**
  Defines business activities, collaborations, and transactions and describes their relationships. Also provides a machine-readable specification instance. It enables collaborative “Business” Web services.

- **Registries and repositories with ebXML Registry/Repository (ebXML Reg/Rep):**
  Provides a powerful classification and storage mechanism for artifacts, including BPSS process specifications and CPP/A partners profile. It may be considered to B2B applications what databases were to enterprise applications.
Semantics and models with ebXML Core Component (ebCC):

The Core Library is a set of standard “parts” that may be used in larger ebXML elements. For example, Core Processes may be referenced by Business Processes. The Core Library is contributed by the ebXML initiative itself, while larger elements may be contributed by specific industries or businesses. It enables B2B interoperability by a common vocabulary.

The term SOAP used here refers to a suite of specifications broader than SOAP itself. It includes Web Service Definition Language (WSDL) and Universal Description, Discovery, and Integration (UDDI), also called the WUS (WSDL, UDDI, SOAP) stack as a whole. This stack is seen by ebXML sponsors as less powerful and feature-rich than the ebXML suite of specifications but simpler to use and more suitable for satisfying alternate requirements.

For example, SOAP over HTTP alone is not sufficient to provide reliable messaging at the application level. Also, the qualities of service that can be captured in ebXML with CPP/A are more detailed and sophisticated than can be realized with SOAP and WSDL.

**ebXML and Web services**

The ebXML architecture appears to have many similarities with the Web services architecture. However, the ebXML organization views the ebXML standard not as an alternative to Web services, but as the standard for “Business” Web services. “Business” Web services are based on a peer-to-peer collaborative business process model, while the basic Web services are based on a client-server, RPC style model.

ebXML provides a modular suite of specifications that is designed to enable standards-based, peer-to-peer, collaborative, business communication between enterprises. ebXML is complimentary to basic Web services and builds upon them to enable “Business” Web services.

**ebXML registry**

An e-business registry is a software product that organizes the information needed to conduct e-business in an automated way. It covers various capabilities:

- Registration of businesses and their capabilities via categories
- Registration of service descriptions
- Discovery of businesses and services

The ebXML registry is a central component of the ebXML initiative to provide a complete framework for electronic business. It manages the storage and the discovery mechanisms for the various elements that are needed to do
e-business within the ebXML framework, and has some advanced features regarding the business aspects of a transaction:

- The Collaboration Protocol Agreement (CPA) defines the capabilities that two parties need to agree upon before engaging in a business collaboration.
- The Collaboration Protocol Profile (CPP) describes the message exchange capabilities of a participant.
- The Business Process Specification Schema (BPSS) provides a standard framework to allow the combination of various transactions. It can be compared to BPEL4WS (see “BPEL4WS” on page 137).

The UDDI registry, as covered in 5.7.2, “UDDI” on page 141, is currently focused on the discovery aspects of automated e-business. The ebXML registry adds collaboration features. The two registries offer some level of interoperability in terms of discovery. At the time of writing, the UDDI Technical Committee is preparing a technical note to provide guidance on how to use UDDI registries within the ebXML framework. The intent is to leverage the complementary strengths of each registry. However, there is no mechanism to move data from one type of registry to the other. The request APIs are specific to each.

UDDI registries seem to be more frequently used than ebXML registries.

**Where to find more information**
For more information on ebXML, see:

- ebXML
  
  [http://www.ebxml.org](http://www.ebxml.org)

- OASIS
  
  [http://www.oasis-open.org](http://www.oasis-open.org)

### 5.5 Service

The service layer of our architectural stack, as shown in Figure 5-9 on page 129, represents the implemented software that can be located and invoked based on a published WSDL interface description.
In this section we examine two different programming models for accessing Web services:

- Web Services for J2EE
- Web Services Invocation Framework (WSIF)

### 5.5.1 Web services and J2EE

Web services are intended to provide interoperability standards between systems, regardless of the architecture or implementation approach of end-point systems.

The Java 2 Platform, Enterprise Edition (J2EE) is an important programming model and architecture, which IBM supports with WebSphere Application Server and WebSphere Studio. Since many of the Product mappings described in this book are based on WebSphere Application Server, it is instructive to review the state of J2EE and Java standards for implementation of Web services on J2EE platforms.

**A new set of Java Specification Requests**

The technologies used by the J2EE application servers to provide Web services facilities are evolving very quickly. The Java community has recently adopted a
set of standards to define the different aspects of how Web services can be supported in a J2EE-compliant application server.

These standards are described as Java Specification Requests (JSRs). JSRs are used as the tracking mechanism for all Java specifications, from proposal through to acceptance or rejection. Information about the Java Community Process, which manages the development of specifications, and the JSRs themselves, can be found at:

http://jcp.org

The main JSR concerning this domain is JSR 109, *Implementing Enterprise Web Services* (also known as Web Services for J2EE). It reached the final release status in November 2002.

The aim of JSR 109 is to define the programming model and runtime architecture for implementing Web services in Java. It federates the work done on several other JSRs. This JSR was led by IBM.

In much the same way that servlets tied together a set of concepts like cookies and HttpSession; and EJBs tied together techniques such as RMI, JTA/JTS, JDBC, and so on with a programming model and runtime model; the promoters of this JSR view it as doing the same for implementing and using Web services.

The Web Services for J2EE 1.0 specification is an addition to J2EE 1.3. J2EE 1.4 and requires support for Web Services for J2EE 1.1. There are minor differences between the J2EE 1.3 version (JSR-109 1.0) and the J2EE 1.4 version (JSR-109 1.1).

Specifications have also been opened for defining APIs to specific parts of the Web services stack:

- **JSR 67: Java APIs for XML Messaging**
  JAXM provides an API for packaging and transporting business transactions using on-the-wire protocols being defined by ebXML.org, OASIS, W3C, and IETF.

- **JSR 93: Java APIs for XML Registry**
  JAXR provides an API for a set of distributed registry services that enable business-to-business integration between business enterprises, using the protocols being defined by ebXML.org, OASIS, and ISO 11179.

- **JSR 101: Java APIs for XML-Based RPC**
  JAX-RPC defines APIs to support emerging industry XML-based RPC standards.
JSR 110: *Java APIs for WSDL*

This JSR provides a standard set of APIs for representing and manipulating services described by WSDL documents. These APIs define a way to construct and manipulate models of service descriptions.

**Introduction in WebSphere Application Server**

Support for JSR 109 and the other related JSRs first appeared in WebSphere Application Server V5.0 as technology previews. IBM WebSphere Application Server V5.1 provides a more complete implementation, in addition to the Web services components that were available previously.

**Supported standards**

The following standards are supported by the Web Services for J2EE component of IBM WebSphere Application Server V5.1:

- SOAP 1.1
- WSDL 1.1
- Web Services for J2EE (JSR-109) 1.0
- Java API for XML-Based RPC (JAX-RPC) 1.0
- SOAP with Attachments API for Java (SAAJ) 1.1

**SOAP considerations**

Apache SOAP 2.3 shipped with WebSphere Application Server V4.0 and V5.0. It continues to co-exist with Web Services for J2EE. Apache SOAP is a proprietary API, and applications written for it are not portable to other SOAP implementations. Applications written for Web Services for J2EE should be portable to any vendor's implementation that supports Web Services for J2EE.

The Web services technology preview in WebSphere V5.0 leveraged the work that IBM contributed to the Apache Axis code base. The Web Services for J2EE support included with WebSphere V5.0.2 is derived from Apache Axis, but has diverged and contains many IBM-specific features to enhance performance, scalability, reliability, interoperability, and integration with the WebSphere Application Server.

The Apache SOAP channel and the SOAP/HTTP channel both support SOAP applications that are SOAP 1.1 compatible (for example, Apache SOAP 2.3 and Axis SOAP 1.0). So if you have an application that uses a production-supported Axis 1.0 SOAP stack, generating SOAP 1.1, then this application can use either channel.

If you are using the Apache SOAP channel, then the SOAP message format must be *RPC* style. To handle *document* style SOAP messages, you must use...
the SOAP/HTTP channel (which supports both RPC style and document style SOAP messages).

If you deploy Web services that pass attachments in a MIME message, then these Web services can only be accessed using the SOAP/HTTP channel.

There is currently no specification for SOAP JMS; each vendor chooses its own implementation technique. Therefore, interoperability is not possible using this protocol at this stage.

**Interoperability**

Web Services for J2EE intends to conform to the WS-I Basic Profile 1.0, and should interoperate with any other vendor conforming to this specification. At the time of the writing, the Basic Profile 1.0 had not been completed, so it is possible that minor incompatibilities exist.

**Development tools**

WebSphere Studio Application Developer V5.1 provides IDE support for the development of Web Services for J2EE. IBM WebSphere Application Server V5.1 provides command line tools for Web services enabling WebSphere applications.

### 5.5.2 Web Services Invocation Framework

The Apache Web Services Invocation Framework (WSIF) provides a standard Java API to invoke services, no matter how or where the service is provided, as long it is described in WSDL.

WSIF enables the developer to move away from the native APIs of the underlying service, and interact with representations of the services instead. This allows the developer to work with the same programming model regardless of how the service is implemented and accessed.

WSIF is WSDL-driven and it provides a uniform interface to invoke services using WSDL documents. So if a SOAP service you are using becomes available as an EJB, for example, you can change to RMI/IIOP by just modifying the WSDL service description, without needing to modify your applications that use the service.

This API is used by tools such as WebSphere Studio Integration Edition and runtimes such as WebSphere Application Server Enterprise to construct and manipulate services defined in WSDL documents. The architecture allows new bindings to be added at runtime.
For more details on the Web Services Invocation Framework see:

http://ws.apache.org/wsif/

**Advantages of WSIF**

WSIF has the following advantages:

- Multiple bindings can be offered for services, and bindings can be decided at runtime.
- Services can be used either by a set of stub classes (static) or by a dynamic interface invocation (dynamic).
- You have the flexibility to switch protocols, location, etc., without having to recompile your client code.

**Disadvantages of WSIF**

WSIF is a Java API, and so cannot be used in environments that are not based on Java.

### 5.6 Business process

Using industry standards, Web services can advertise their interfaces, be discovered and invoked upon, and communicate with each other to deliver end-to-end functionality. But to support the business process, a further level of abstraction is required. The business process layer in our architectural stack, shown in Figure 5-10 on page 134, allows us to create and define complex processes or workflows. Processes are composed from the operations supplied by Web services that can be nested and sequenced according to business requirements.
Figure 5-10  The business process layer

By isolating the business process from the implementation of the underlying Web services, as shown in Figure 5-11 on page 135, you can more readily adapt to changing business conditions.
Chapter 5. Technology options

Using a consistent modeling approach simplifies communication between all parties involved in the business process. In addition, models can be shared between partners without dictating the development tools or run-time environment that each of the partners in the process must use. A consistent modeling approach based on open standards also allows the activities in the process to be loosely coupled to the process itself, thereby minimizing the time and effort required to implement changes to the process as the business environment or requirements change.

5.6.1 WSFL and XLANG

Both the Web Services Flow Language (WSFL) from IBM and Microsoft XLANG were early suggestions for business process execution standards. These both have been combined and further developed to create a common standard backed by most of the current industry leaders as BPEL4WS, which is described in further detail in “Emerging standards for business process” on page 136.