2

Service-oriented architecture

This chapter provides an introduction to service-oriented architecture. It also introduces Web Services as an implementation of service-oriented architecture (SOA).

In this chapter, we discuss the following topics:

- Overview of service-oriented architecture
- Web services architecture
- Web services and service-oriented architecture
- Enterprise Service Bus
- Where to find more information

2.1 Overview of service-oriented architecture

In this section we briefly describe the evolution of service-oriented architecture. We then explore the relationship between component-based development and service-oriented architecture and show how components can be the cornerstones of the infrastructure for implementing services.

2.1.1 The business drivers for a new approach

While IT executives have been facing the challenge of cutting costs and maximizing the utilization of existing technology, at the same time they have to continuously strive to serve customers better, be more competitive, and be more responsive to the business's strategic priorities.

There are two underlying themes behind all of these pressures: *Heterogeneity* and *change*. Most enterprises today contain a range of different systems, applications, and architectures of different ages and technologies. Integrating products from multiple vendors and across different platforms were almost always a nightmare. But we also cannot afford to take a single-vendor approach to IT, because application suites and the supporting infrastructure are so inflexible.

Change is the second theme underlying the questions that today's IT executives face. Globalization and e-business are accelerating the pace of change. Globalization leads to fierce competition, which leads to shortening product cycles, as companies look to gain advantage over their competition. Customer needs and requirements change more quickly driven by competitive offerings and wealth of product information available over the Internet. In response the cycle of competitive improvements in products and services further accelerates.

Improvements in technology continue to accelerate, feeding the increased pace of changing customer requirements. Business must rapidly adapt to survive, let alone to succeed in today's dynamic competitive environment, and the IT infrastructure must enable businesses' ability to adapt.

As a result, business organizations are evolving from the vertical, isolated business divisions of the 1980's and earlier, to the horizontal business-process-focused structures of the 1980's and 1990's, towards the new ecosystem business paradigm. Business services now need to be componentized and distributed. There is a focus on the extended supply chain, enabling customer and partner access to business services. The CBDI Forum Report *Business Integration - Drivers and Directions* illustrates this evolution of business as shown in Figure 2-1 on page 19. You can access this CBDI report and a related CBDI workshop titled *Service Based Approach* at:

http://www.cbdiforum.com/

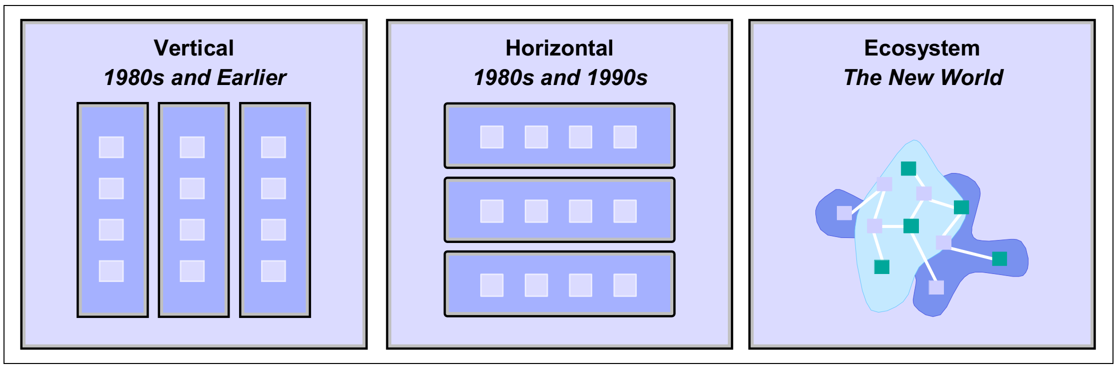

Figure 2-1 The evolution of business

How do I make my IT environment more flexible and responsive to the ever changing business requirements? How can we make those heterogeneous systems and applications communicate as seamlessly as possible? How can we achieve the business objective without bankrupting the enterprise?

The IT answers/enablers have been evolving in parallel with this evolution of business, as shown in Figure 2-2. Currently many IT executives and professionals alike believe that now we are getting really close to providing a satisfactory answer with service-oriented architecture.

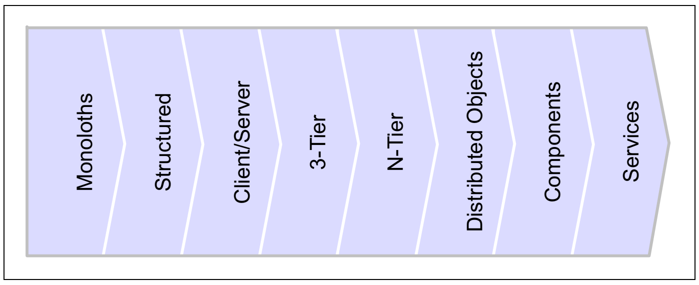

Figure 2-2 The evolution of architecture

In order to alleviate the problems of heterogeneity, interoperability and ever changing requirements, such an architecture should provide a platform for building application services with the following characteristics:

- Loosely coupled
- Location transparent
- Protocol independent

Based on such a service-oriented architecture, a service consumer does not even have to care about a particular service it is communicating with because the underlying infrastructure, or service “bus”, will make an appropriate choice on behalf of the consumer. The infrastructure hides as many technicalities as possible from a requestor. Particularly technical specificities from different implementation technologies such as J2EE or .NET should not affect the SOA users. We should also be able to reconsider and substitute a “better” service implementation if one is available, and with better quality of service characteristics.

2.1.2 Service-oriented architecture as a solution

Ever since the “software crisis” prompted the beginnings of software engineering, the IT industry has been struggling to find solutions to solve the above problems. Throughout the years, the following short list of core technology advancements have brought us to where we are today. We will briefly discuss those core technologies and our focus will be on how such technologies help resolve IT problems.

Object-oriented analysis and design

Larman describes the essence of the object-oriented analysis and design as considering “a problem domain and logical solution from the perspective of objects (things, concepts, or entities)” in *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design*. Jacobson, et al, define these objects as being “characterized by a number of operations and a state that remembers the effects of these operations” in *Object-Oriented Software Engineering: A Use Case Driven Approach*.

In object-oriented analysis, such objects are identified and described in the problem domain, while in object-oriented design, they are transitioned into logical software objects that will ultimately be implemented in a object-oriented programming language.

With object-oriented analysis and design, certain aspects of the object (or group of objects) can be encapsulated to simplify the analysis of complex business scenarios. Certain characteristics of the object(s) can also be abstracted so that only the important or essential aspects are captured, in order to reduce complexity.

Component-based design

Component-based design is not a new technology. It is naturally evolved from the object paradigm. In the early days of object-oriented analysis and design, fine-grained objects were marked as a mechanism to provide “reuse”, but those objects are at too low a level of granularity and the there are no standards in place to make widespread reuse practical. Coarse-grained components have become more and more a target for reuse in application development and system integration. These coarse-grained components provide certain well defined functionality from a cohesive set of finer-grained objects. In this way, packaged solution suites can also be encapsulated as such “components”.

Once the organization achieves a higher level of architectural maturity based on distinctly separate functional components, the applications that support the business can be partitioned into a set of increasingly larger grained components. Components can be seen as the mechanism to package, manage and expose services. They can use a set of technologies in concert: Large-grained enterprise components, that implement business-level use-cases, can be implemented using newer object-oriented software development in combination with legacy systems.

Service-oriented design

In *Component-Based Development for Enterprise Systems*, Allen includes the notion of services, describing a component as “an executable unit of code that provides physical black-box encapsulation of related services. Its service can only be accessed through a consistent, published interface that includes an interaction standard. A component must be capable of being connected to other components (through a communications interface) to a larger group”.

A service is generally implemented as a course-grained, discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled, message-based communication model. Figure 2-3 on page 22 shows important service-oriented terminology:

- Services: Logical entities, the contracts defined by one or more published interfaces.
- Service provider: The software entity that implements a service specification.
- Service consumer (or requestor): The software entity that calls a service provider. Traditionally, this is termed a “client”. A service consumer can be an end-user application or another service.
- Service locator: A specific kind of service provider that acts as a registry and allows for the lookup of service provider interfaces and service locations.
- Service broker: A specific kind of service provider that can pass on service requests to one or more additional service providers.

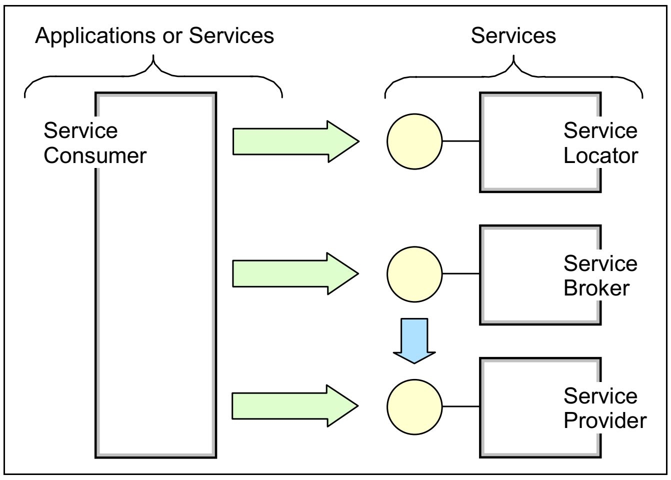

Figure 2-3 Service-oriented terminology

Interface-based design

In both component and service development, the design of the interfaces is done such that a software entity implements and exposes a key part of its definition. Therefore, the notion and concept of “interface” is key to successful design in both component-based and service-oriented systems. The following are some key interface-related definitions:

- Interface: Defines a set of public method signatures, logically grouped but providing no implementation. An interface defines a contract between the requestor and provider of a service. Any implementation of an interface must provide all methods.
- Published interface: An interface that is uniquely identifiable and made available through a registry for clients to dynamically discover.
- Public interface: An interface that is available for clients to use but is not published, thus requiring static knowledge on the part of the client.
- Dual interface: Frequently interfaces are developed as pairs such that one interface depends on another; for example, a client must implement an interface to call a requestor because the client interface provides some callback mechanism.

Figure 2-4 on page 23 shows the UML definition of a customer relationship management (CRM) service, represented as a UML component, that implements the interfaces AccountManagement, ContactManagement, and SystemsManagement. Only the first two of these are published interfaces, although the latter is a public interface. Note that the SystemsManagement interface and ManagementService interface form a dual interface. The CRM service can implement any number of such interfaces, and it is this ability of a service (or component) to behave in multiple ways depending on the client that allows for great flexibility in the implementation of behavior. It is even possible to provide different or additional services to specific classes of clients. In some run-time environments such a capability is also used to support different versions of the same interface on a single component or service.

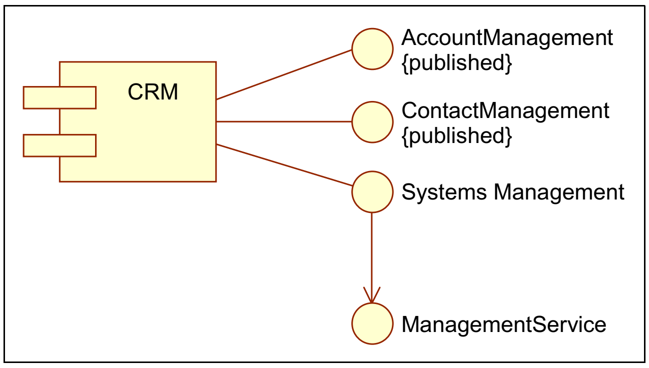

Figure 2-4 Implemented services

Layered application architectures

As mentioned before, object-oriented technology and languages are great ways to implement components. While components are the best way to implement services, though one has to understand that a good component-based application does not necessarily make an good service-oriented application. A great opportunity exists to leverage component developers and existing components, once the role played by services in application architecture is understood. The key to making this transition is to realize that a service-oriented approach implies an additional application architecture layer. Figure 2-5 on page 24 demonstrates how technology layers can be applied to application architecture to provide more coarse-grained implementations as one gets closer to the consumers of the application. The term coined to refer to this part of the system is “the application edge,” reflecting the fact that a service is a great way to expose an external view of a system, with internal reuse and composition using traditional component design.

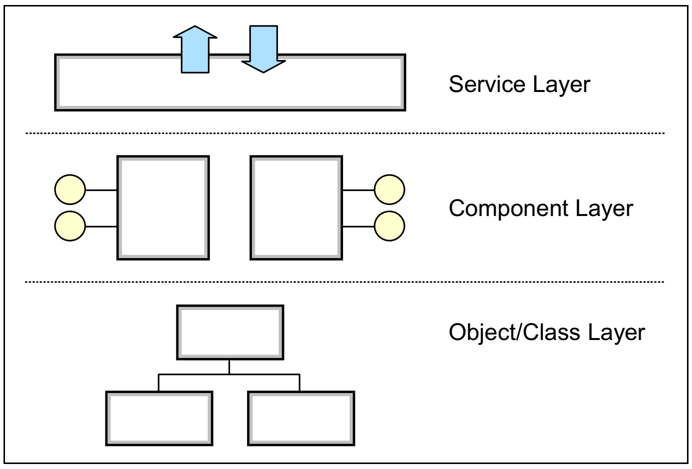

Figure 2-5 Application implementation layers: Services, components, objects

2.1.3 A closer look at service-oriented architecture

Service-oriented architecture presents an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services. It is comprised of elements that can be categorized into *functional* and *quality of service*. Figure 2-6 on page 25 shows the architectural stack and the elements that might be observed in a service-oriented architecture.

Note: Service-oriented architecture stacks can be a contentious issue, with several different stacks being put forward by various proponents. Our stack is not being positioned as *the services stack*. It is just presented as useful framework for structuring the SOA discussion in the rest of the publication.

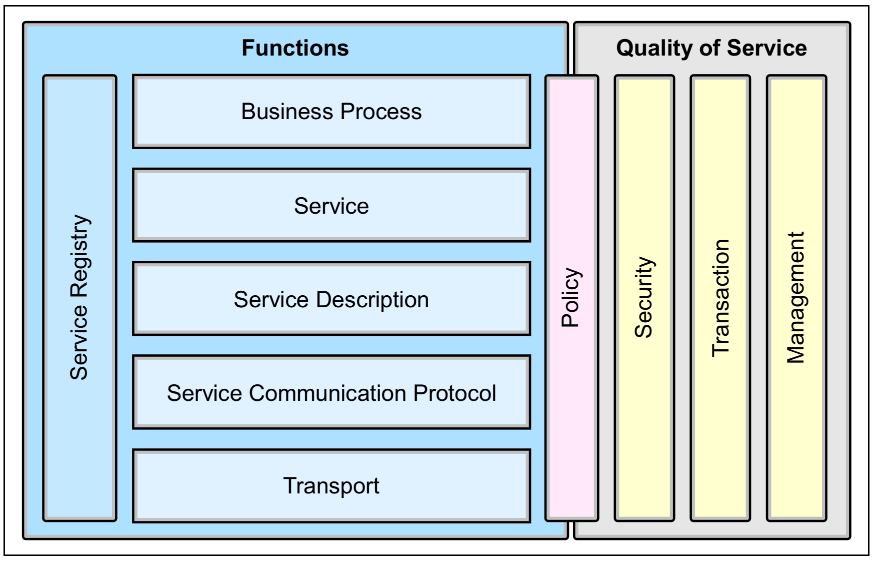

Figure 2-6 Elements of a service-oriented architecture

The architectural stack is divided into two halves, with the left half addressing the functional aspects of the architecture and the right half addressing the quality of service aspects. These elements are described in detail as follows:

- Functional aspects include:
  - *Transport* is the mechanism used to move service requests from the service consumer to the service provider, and service responses from the service provider to the service consumer.
  - *Service Communication Protocol* is an agreed mechanism that the service provider and the service consumer use to communicate what is being requested and what is being returned.
  - *Service Description* is an agreed schema for describing what the service is, how it should be invoked, and what data is required to invoke the service successfully.
  - *Service* describes an actual service that is made available for use.
  - *Business Process* is a collection of services, invoked in a particular sequence with a particular set of rules, to meet a business requirement. Note that a business process could be considered a service in its own right, which leads to the idea that business processes may be composed of services of different granularities.
  - The *Service Registry* is a repository of service and data descriptions which may be used by service providers to publish their services, and service

consumers to discover or find available services. The service registry may provide other functions to services that require a centralized repository.

- Quality of service aspects include:
  - *Policy* is a set of conditions or rules under which a service provider makes the service available to consumers. There are aspects of policy which are functional, and aspects which relate to quality of service; therefore we have the policy function in both functional and quality of service areas.
  - *Security* is the set of rules that might be applied to the identification, authorization, and access control of service consumers invoking services.
  - *Transaction* is the set of attributes that might be applied to a group of services to deliver a consistent result. For example, if a group of three services are to be used to complete a business function, all must complete or none must complete.
  - *Management* is the set of attributes that might be applied to managing the services provided or consumed.

SOA collaborations

Figure 2-7 shows the collaborations in a service-oriented architecture. The collaborations follows the “find, bind and invoke” paradigm where a service consumer performs dynamic service location by querying the service registry for a service that matches its criteria. If the service exists, the registry provides the consumer with the interface contract and the endpoint address for the service. The following diagram illustrates the entities in an service-oriented architecture that collaborate to support the “find, bind and invoke” paradigm.

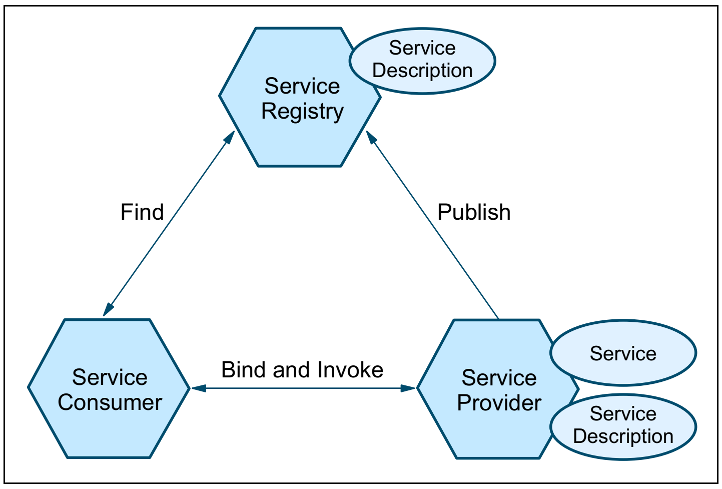

Figure 2-7 Collaborations in a service-oriented architecture

The roles in a service-oriented architecture are:

- Service consumer: The service consumer is an application, a software module or another service that requires a service. It initiates the enquiry of the service in the registry, binds to the service over a transport, and executes the service function. The service consumer executes the service according to the interface contract.
- Service provider: The service provider is a network-addressable entity that accepts and executes requests from consumers. It publishes its services and interface contract to the service registry so that the service consumer can discover and access the service.
- Service registry: A service registry is the enabler for service discovery. It contains a repository of available services and allows for the lookup of service provider interfaces to interested service consumers.

Each entity in the service-oriented architecture can play one (or more) of the three roles of service provider, consumer and registry.

The operations in a service-oriented architecture are:

- Publish: To be accessible, a service description must be published so that it can be discovered and invoked by a service consumer.
- Find: A service requestor locates a service by querying the service registry for a service that meets its criteria.
- Bind and invoke: After retrieving the service description, the service consumer proceeds to invoke the service according to the information in the service description.

The artifacts in a service-oriented architecture are:

- Service: A service that is made available for use through a published interface that allows it to be invoked by the service consumer.
- Service description: A service description specifies the way a service consumer will interact with the service provider. It specifies the format of the request and response from the service. This description may specify a set of preconditions, post conditions and/or quality of service (QoS) levels.

In addition to dynamic service discovery and definition of a service interface contract, a service-oriented architecture has the following characteristics:

- Services are self-contained and modular.
- Services support interoperability.
- Services are loosely coupled.
- Services are location-transparent.

- Services are composite modules, comprised of components.

These characteristics are also central to fulfilling the requirements for an e-business on demand™ operational environment, as defined in Chapter 10, “e-business on demand and Service-oriented architecture” on page 301.

Finally, service-oriented architecture is not a new notion. As shown in Figure 2-8, examples of technologies that are at least partly service-oriented include CORBA, DCOM and J2EE. Early adopters of the service-oriented architecture approach have also successfully created their own service-oriented enterprise architectures based on messaging systems such as IBM WebSphere MQ. Most recently, the SOA arena has expanded to include the World Wide Web (WWW) and Web Services.

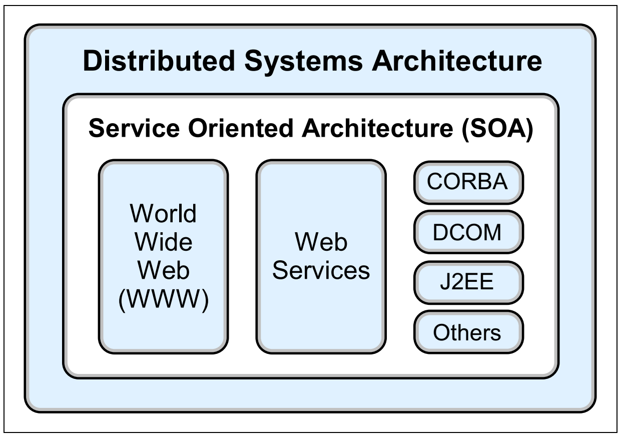
Distributed Systems Architecture
Service Oriented Architecture (SOA)
World Wide Web (WWW)
Web Services
CORBA
DCOM
J2EE
Others

Figure 2-8 Different implementations of service-oriented architecture

Services in the context of SOA

In service-oriented architecture, services map to the business functions that are identified during business process analysis. The services may be fine- or coarse-grained depending upon the business processes. Each service has a well-defined interface that allows it to be published, discovered and invoked. An enterprise can choose to publish its services externally to business partners or internally within the organisation. A service can also be composed from other services.

Services vs. components

A service is a coarse-grained processing unit that consumes and produces sets of objects passed-by-value. It is not the same as an object in programming language terms. Instead, it is perhaps closer to the concept of a business transaction such as a CICS® or IMS™ transaction than to a remote CORBA object.

A service consists of a collection of components that work in concert to deliver the business function that the service represents. Thus, in comparison, components are finer-grained than services. In addition, while a service maps to a business function, a component typically maps to business entities and the business rules that operate on them. As an example, let us look at the Purchase Order component model for the WS-I Supply Chain Management sample, shown in Figure 2-9.

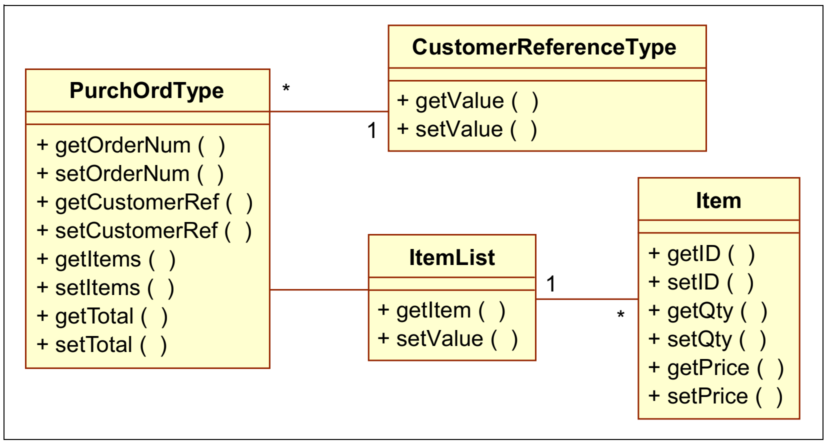

Figure 2-9 Purchase Order component model

In a component-based design, components are created to closely match business entities (such as Customer, Purchase Order, Order Item) and encapsulate the behavior that matches the entities' expected behavior.

For example, the Purchase Order component provides functions to obtain information about the list of products ordered and the total amount of the order; the Item component provides functions to obtain information about the quantity and price of the product ordered. The implementation of each component is encapsulated behind the interface. So, a user of the Purchase Order component does not know the schema of the Purchase Order table and the algorithm for calculating tax, rebates and/or discounts on the total amount of the order.

In a service-oriented design, services are not designed based on business entities. Instead, each service is a holistic unit that manages operations across a set of business entities. For example, a customer service will respond to any request from any other system or service that needs to access customer information. The customer service can process a request to update customer information; add, update, delete investment portfolios; and enquire about the customer's order history. The customer service owns all the data related to the customers it is managing and is capable of making other service inquiries on behalf of the calling party in order to provide a unified customer service view. This means a service is a *manager* object that creates and manages its set components.

2.1.4 Service-oriented architecture benefits

As discussed earlier, businesses are dealing with two fundamental concerns: The ability to change quickly, and the need to reduce costs. To remain competitive businesses must adapt quickly to internal factors such as acquisitions and restructuring, or external factors like competitive forces and customer requirements. Cost-effective, flexible IT infrastructure is need to support the business.

With a service-oriented architecture, we can realize several benefits to help organizations succeed in the dynamic business landscape of today:

- Leverage existing assets.

  SOAs provide a layer of abstraction that enables an organization to continue leveraging its investment in IT by wrapping these existing assets as services that provide business functions. Organizations potentially can continue getting value out of existing resources instead of having to rebuild from scratch.

- Easier to integrate and manage complexity.

  The integration point in a service-oriented architecture is the service specification and not the implementation. This provides implementation transparency and minimizes the impact when infrastructure and implementation changes occur. By providing a service specification in front of existing resources and assets built on disparate systems, integration becomes more manageable since complexities are isolated. This becomes even more important as more businesses work together to provide the value chain.

- More responsive and faster time-to-market.

  The ability to compose new services out of existing ones provides a distinct advantage to an organization that has to be agile to respond to demanding business needs. Leveraging existing components and services reduces the time needed to go through the software development life cycle of gathering requirements, performing design, development and testing. This leads to rapid development of new business services and allows an organization to respond quickly to changes and reduce the time-to-market.

- Reduce cost and increase reuse.

With core business services exposed in a loosely coupled manner, they can be more easily used and combined based on business needs. This means less duplication of resources, more potential for reuse, and lower costs.

- Be ready for what lies ahead.

  SOAs allows businesses be ready for the future. Business processes which comprise of a series of business services can be more easily created, changed and managed to meet the needs of the time. SOA provides the flexibility and responsiveness that is critical to businesses to survive and thrive.

Service-oriented architecture is by no means a silver bullet, and migration to SOA is not an easy task. Rather than migrating the whole enterprise to a service-oriented architecture overnight, the recommended approach is to migrate an appropriate subset of business functions as the business need arises or is anticipated.

2.2 Web services architecture

Web services are a relatively new technology that have received wide acceptance as an important implementation of service-oriented architecture. This is because Web services provides a distributed computing approach for integrating extremely heterogeneous applications over the Internet. The Web service specifications are completely independent of programming language, operating system, and hardware to promote loose coupling between the service consumer and provider. The technology is based on open technologies such as:

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

Using open standards provides broad interoperability among different vendor solutions. These principles mean that companies can implement Web services without having any knowledge of the service consumers, and vice versa. This facilitates just-in-time integration and allows businesses to establish new partnership easily and dynamically.

2.2.1 What Web services are

The W3C's Web Services Architecture Working Group has jointly come to agreement on the following working definition of a Web service:

"A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols."

Basic Web services combine the power of two ubiquitous technologies: XML, the universal data description language; and the HTTP transport protocol widely supported by browser and Web servers.

```
Web services = XML + transport protocol (such as HTTP)
```

Some of the key features of Web services are the following:

- Web services are self-contained.

  On the client side, no additional software is required. A programming language with XML and HTTP client support, for example, is enough to get you started. On the server side, merely a Web server and a servlet engine are required. It is possible to Web service enable an existing application without writing a single line of code.

- Web services are self-describing.

  Neither the client nor the server knows or cares about anything besides the format and content of request and response messages (loosely coupled application integration).

  The definition of the message format travels with the message. No external metadata repositories or code generation tools are required.

- Web services are modular.

  Web services are a technology for deploying and providing access to business functions over the Web; J2EE, CORBA, and other standards are technologies for implementing these Web services.

- Web services can be published, located, and invoked across the Web.

  The standards required to do so are:

  - Simple Object Access Protocol (SOAP), also known as service-oriented architecture protocol, an XML-based RPC and messaging protocol
  - Web Service Description Language (WSDL), a descriptive interface and protocol binding language
  - Universal Description, Discovery, and Integration (UDDI), a registry mechanism that can be used to look up Web service descriptions

- Web services are language independent and interoperable.

  The interaction between a service provider and a service requester is designed to be completely platform and language independent. This interaction requires a WSDL document to define the interface and describe the service, along with a network protocol (usually HTTP). Because the service provider and the service requester have no idea what platforms or languages the other is using, interoperability is a given.

- Web services are inherently open and standards based.

  XML and HTTP are the technical foundation for Web services. A large part of the Web service technology has been built using open source projects. Therefore, vendor independence and interoperability are realistic goals.

- Web services are dynamic.

  Dynamic e-business can become a reality using Web services because, with UDDI and WSDL, the Web service description and discovery can be automated.

- Web services are composable.

  Simple Web services can be aggregated to more complex ones, either using workflow techniques or by calling lower-layer Web services from a Web service implementation.

Figure 2-10 shows a typical Web service collaboration that is based on the SOA model shown previously in Figure 2-7 on page 26.

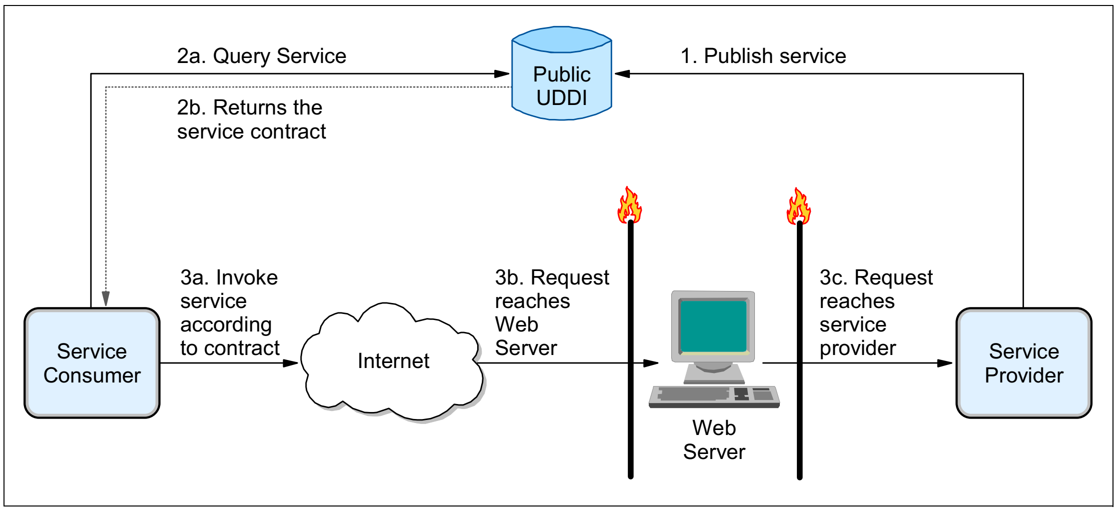
2a. Query Service
Public UDDI
1. Publish service
2b. Returns the service contract
3a. Invoke service according to contract
3b. Request reaches Web Server
3c. Request reaches service provider
Service Consumer
Internet
Web Server
Service Provider

Figure 2-10 Web service collaboration

2.2.2 Web service interoperability

Web services are one of the rising stars in the IT world, supporting the integration of existing systems and sharing of resources and data, both within and outside an organization. They are a relatively new technology, with Web services standards continuing to be refined and developed.

For the key promise of Web services interoperability to work, standards need to be carefully managed. In addition, guidance in interpretation and implementation of standards is essential to facilitate adoption of a technology. The Web Services Interoperability Organization has an important role in this area, as a *standards integrator* to help Web services advance in a structured and coherent manner.

IBM's commitment in this direction includes active participation in WS-I standards development, and early delivery of WS-I compliance in runtime and development products. In this publication, we place considerable emphasis on WS-I standards and guidelines as an enabler for Web services interoperability.

Web Services Interoperability Organization

Web Services Interoperability Organization (WS-I) is an open, industry consortium of about 150 companies, representing diverse industries such as automotive, consumer packaged goods, finance, government, insurance, media, telecommunications, travel and the computer industry. It is chartered to:

- Promote Web services interoperability across platforms, operating systems, and programming languages with the use of generic protocols for interoperable exchange of messages between services.
- Encourage Web services adoption.
- Accelerate deployment by providing guidance, best practices and other resources for developing interoperable Web services.

WS-I, as a standards integrator, supports the relationships with standards bodies who own specifications and fosters communication and cooperation with industry consortia and other organizations, as shown in Figure 2-11 on page 35.

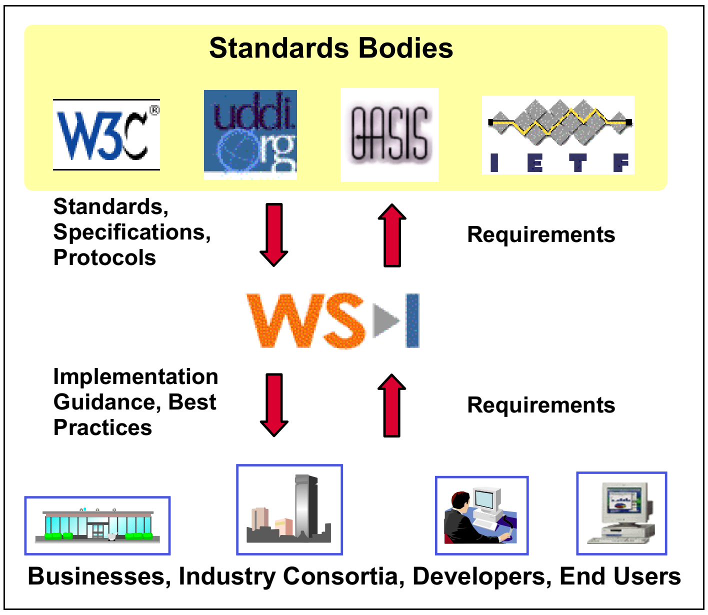
Standards Bodies
W3C
uddi.org
OASIS
IETF
Standards, Specifications, Protocols
Requirements
WS-I
Implementation Guidance, Best Practices
Requirements
Businesses, Industry Consortia, Developers, End Users

Figure 2-11 WS-I, standards and industry

WS-I has a set of deliverables to assist in the development and deployment of Web services, including its *profile* of interoperable Web services. A profile is defined in the WS-I Glossary as follows:

> "A collection of requirements to support interoperability. WS-I will deliver a collection of profiles that support technical requirements and specifications to achieve interoperable Web Services."

It includes the following deliverables:

- Profile Specification

  This includes a list of non-proprietary Web services-related specifications at certain version levels, plus a list of clarifications and restrictions on those specifications to facilitate the development of interoperable Web services.

- Use Cases and Usage Scenarios

  These capture the business and technical requirements, respectively, for the use of Web services. These requirements reflect the classes of real-world requirements supporting Web services solutions, and provide a framework to demonstrate the guidelines described in WS-I Profiles.

- Sample Applications

  These demonstrate the implementation of applications that are built from Web services Usage Scenarios and Use Cases, and that conform to a given set of Profiles. Implementations of the same Sample Application on multiple platforms, using different languages and development tools, allow WS-I to demonstrate interoperability in action, and to provide readily usable resources for the Web services practitioner.

- Testing Tools

  These are used to monitor and analyze interactions with a Web service to determine whether or not the messages exchanged conform to WS-I Profile guidelines.

For more information on WS-I, refer to:

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

WS-I Basic Profile 1.0

WS-I has delivered its first profile of interoperable Web services called *WS-I Basic Profile 1.0*, which focuses on the core foundation technologies upon which Web services are based: HTTP, SOAP, WSDL, UDDI, XML and XML Schema. Basic Profile 1.0 was unanimously approved on July 22, 2003, by the WS-I board of directors and members.

The WS-I Basic Profile 1.0 - Profile Specification consists of the following non-proprietary Web services related specifications:

- SOAP 1.1
- WSDL 1.1
- UDDI 2.0
- XML 1.0 (Second Edition)
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- RFC2246: The Transport Layer Security Protocol Version 1.0
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- RFC2616: HyperText Transfer Protocol 1.1
- RFC2818: HTTP over TLS
- RFC2965: HTTP State Management Mechanism
- The Secure Sockets Layer Protocol Version 3.0

The WS-I Basic Profile 1.0 - Usage Scenario consists of three usage scenarios, where a usage scenario is a design pattern of interacting entities including actor, roles and message exchange patterns:

- One-way Usage Scenario
  - Simplest usage scenario, where the message exchange is one-way, with a Consumer sending a request to a Provider
  - Should be used only where loss of information can be tolerated
- Synchronous request/response Usage Scenario

  Most commonly used usage scenario where a Consumer sends a request to a Provider, who processes the request and sends back a response
- Basic callback Usage Scenario
  - Used to simulate an asynchronous operation using synchronous operations
  - Composed of two synchronous request/response usage scenarios, one initiated by a Consumer and the other by a Producer

A mapping of the usage scenarios to the clarifications and restrictions of the profile specifications is done via a Web services usage stack to guide Web services developers.

The WS-I Supply Chain Management sample application depicts an application for a fictitious consumer electronics retailer. We will use the WS-I sample to illustrate concepts and scenarios throughout the publication.

See also the following IBM developerWorks® articles:

- *First look at the WS-I Basic Profile 1.0*, available at:

  `http://www.ibm.com/developerworks/webservices/library/ws-basicprof.html`
- *First look at the WS-I Usage Scenarios*, available at:

  `http://www.ibm.com/developerworks/webservices/library/ws-iuse/`
- *Preview of WS-I sample application*, available at:

  `http://www.ibm.com/developerworks/webservices/library/ws-wsisamp/`

2.3 Web services and service-oriented architecture

Web services are a technology that is well suited to implementing a service-oriented architecture. In essence, Web services are self-describing and modular applications that expose business logic as services that can be published, discovered, and invoked over the Internet. Based on XML standards, Web services can be developed as loosely coupled application components using any programming language, any protocol, or any platform. This facilitates the delivery of business applications as a service accessible to anyone, anytime, at any location, and using any platform.

It is important to point out that Web services are not the only technology that can be used to implement a service-oriented architecture. Many examples of organizations who have successfully implemented service-oriented architectures using other technologies can be found. Web services have also been used by others to implement architectures that are not service-oriented. In this publication, however, our focus is on using Web services to implement an SOA.

For more information on SOA and Web services, refer to:

http://www.ibm.com/software/solutions/webservices/resources.html

This Web site provides a collection of IBM resources on this topic.

2.4 Enterprise Service Bus

Web services based technologies are becoming more widely used in enterprise application development and integration. One of the critical issues arising now is finding more efficient and effective ways of designing, developing and deploying Web services based business systems; more importantly, moving beyond the basic point-to-point Web services communications to broader application of these technologies to enterprise-level business processes. In this context, the Enterprise Service Bus (ESB) model is emerging as a major step forward in the evolution of Web services and service-oriented architecture.

2.4.1 Basic Web services

Basic (point-to-point SOAP/HTTP) Web services provide a solid foundation for implementing a service-oriented architecture, but there are important considerations that affect their flexibility and maintainability in enterprise-scale architectures.

First, the point-to-point nature of basic Web services means that service consumers often need to be modified whenever the service provider interface changes. This is often not a problem on a small scale, but in large enterprises it could mean changes to many client applications. It can also become increasingly difficult to make such changes to legacy clients.

Second, you can end up with an architecture that is fragile and inflexible when large numbers of service consumers and providers communicate using point-to-point “spaghetti” style connections.

Last, basic Web services require that each consumer has a suitable protocol adapter for each provider it needs to use. Having to deploy multiple protocol adapters across many client applications adds to cost and maintainability issues.

Let us look at how the Enterprise Service Bus approach addresses these issues.

2.4.2 What an Enterprise Service Bus is

The Enterprise Service Bus concept is not a product, but an architectural best practice for implementing a service-oriented architecture. As shown in Figure 2-12, it establishes an enterprise-class messaging bus that combines messaging infrastructure with message transformation and content-based routing in a layer of integration logic between service consumers and providers.

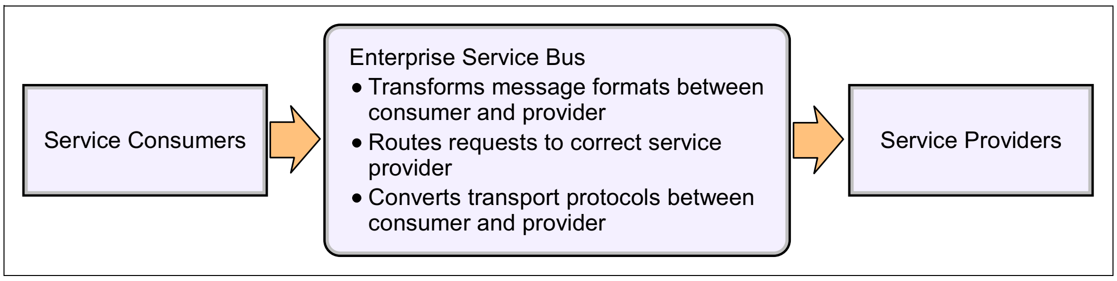
Service Consumers
Enterprise Service Bus
• Transforms message formats between consumer and provider
• Routes requests to correct service provider
• Converts transport protocols between consumer and provider
Service Providers

Figure 2-12 Enterprise Service Bus

The main aim of the Enterprise Service Bus is to provide virtualization of the enterprise resources, allowing the business logic of the enterprise to be developed and managed independently of the infrastructure, network, and provision of those business services. Resources in the ESB are modelled as services that offer one or more business operations.

Implementing an Enterprise Service Bus requires an integrated set of middleware services that support the following architecture styles:

- *Services oriented architectures,* where distributed applications are composed of granular re-usable services with well-defined, published and standards-compliant interfaces
- *Message-driven architectures,* where applications send messages through the ESB to receiving applications
- *Event-driven architectures,* where applications generate and consume messages independently of one another

The middleware services provided by an Enterprise Service Bus need to include:

- Communication middleware supporting a variety of communication paradigms (such as synchronous, asynchronous, request/reply, one-way, call-back), qualities of service (such as security, guaranteed delivery, performance, transactional), APIs, platforms, and standard protocols
- A mechanism for injecting intelligent processing of in-flight service requests and responses within the network
- Standard-based tools for enabling rapid integration of services
- Management system for loosely-coupled applications and their interactions

2.4.3 The IBM vision

As shown in Figure 2-13 on page 41, IBM is extending its Web services and service-oriented architecture vision with an Enterprise Service Bus architecture that provides a standards-based integration layer using the intermediary/mediator pattern.

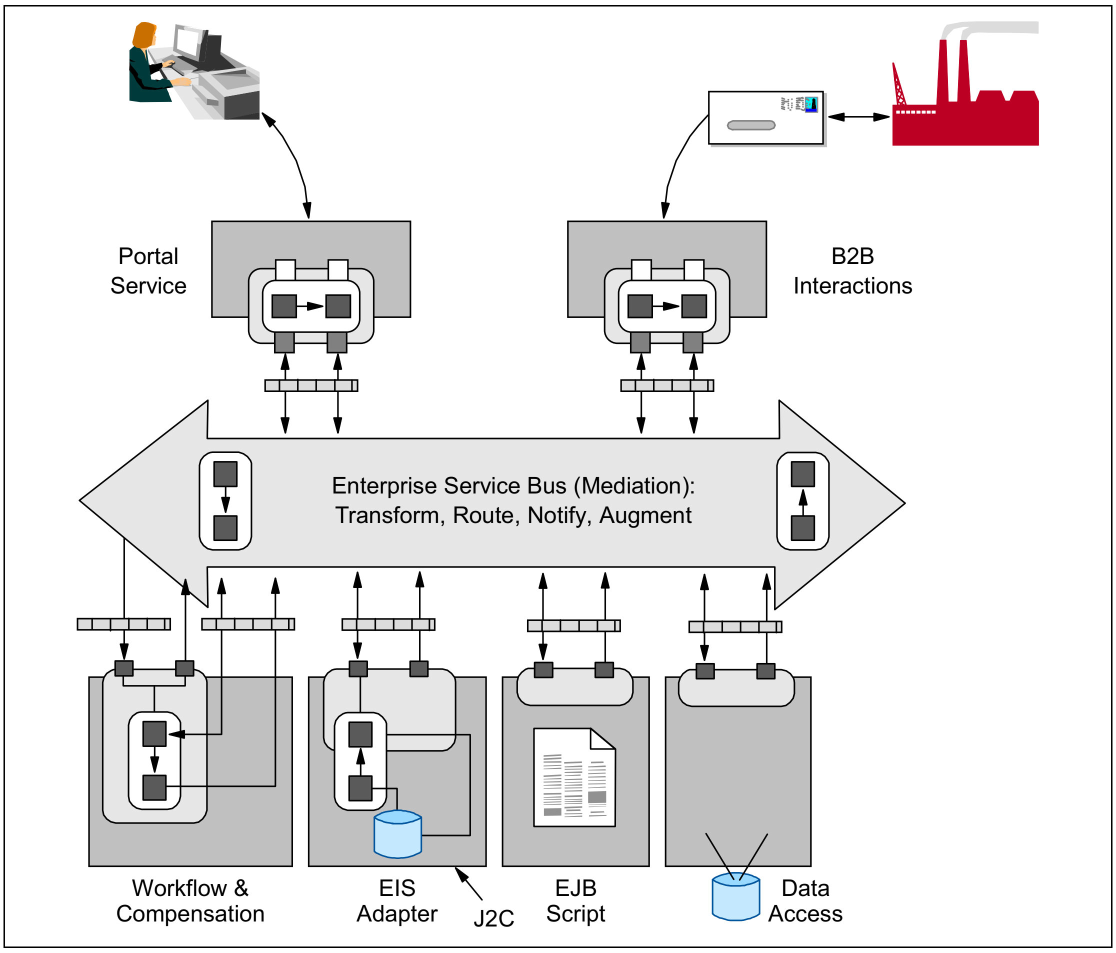

Figure 2-13 Enterprise Service Bus conceptual model

Intelligent mediations are invoked between service consumer and provider that facilitate the selection of services, logging, usage metrics, and so on. These mediations can be configured by policies that define consumer and provider capabilities and requirements. For example, if a provider expects encrypted messages, the requester mediation should include such capability. If a requester only supports SOAP/HTTP, an intermediary should be added to convert to SOAP/JMS.

The other main components in this model include WSDL services that are generated using tools, or implemented by programmers. These include:

- A workflow and compensation system that executes business processes, or workflows. Activity nodes in a business process typically map to invoking an operation on a Web service. This component supports J2EE transactions and a compensation model for long duration business processes.
- Enterprise Information System (EIS) adaptors based on the J2EE Connector Architecture allow integration with legacy systems.
- XML database access through a WSDL interface.
- Custom-built WSDL services implemented by programmers using the J2EE programming model.

The Enterprise Service Bus supports multiple protocols for communication between services, including SOAP, HTTP, JMS, RMI/IIOP, and so on.

The Enterprise Service Bus can be implemented today using currently available IBM WebSphere products, for example:

- IBM WebSphere Application Server V5.1 provides a J2EE runtime environment for services using SOAP over HTTP or JMS, and J2EE Connectors for EIS integration.
- IBM WebSphere MQ provides the JMS messaging infrastructure.
- The Web Services Gateway with IBM WebSphere Application Server Network Deployment V5.1 provides SOAP message routing, transformation, and protocol conversion. The UDDI Registry provides dynamic service discovery.
- IBM WebSphere Application Server Enterprise V5.0 provides the Process Choreographer for service orchestration or workflow.
- IBM DB2® XML Extender enables database access via the ESB.
- IBM WebSphere Portal allows integrated, personalized end-user access to business services.

You can also expect a strong focus on Enterprise Service Bus capabilities in future releases of WebSphere family products.

Note: Implementation of an Enterprise Service Bus is beyond the scope of this publication. The intention here is just to provide an introduction to this important architectural best practice.

2.5 Where to find more information

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

- The following IBM developerWorks articles:
  - *First look at the WS-I Basic Profile 1.0*, available at:

    `http://www.ibm.com/developerworks/webservices/library/ws-basicprof.html`
  - *First look at the WS-I Usage Scenarios*, available at:

    `http://www.ibm.com/developerworks/webservices/library/ws-iuse/`
  - *Preview of WS-I sample application*, available at:

    `http://www.ibm.com/developerworks/webservices/library/ws-wsisamp/`
  - *Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications*, available at:

    `http://www.ibm.com/developerworks/rational/library/510.html`
- W3C Working Group Note, *Web Services Architecture*, available at:

  `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`
- Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 2nd Ed., Prentice Hall, 2001
- Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992
- Paul Allen, *Component-Based Development for Enterprise Systems, Cambridge University Press*, 1998