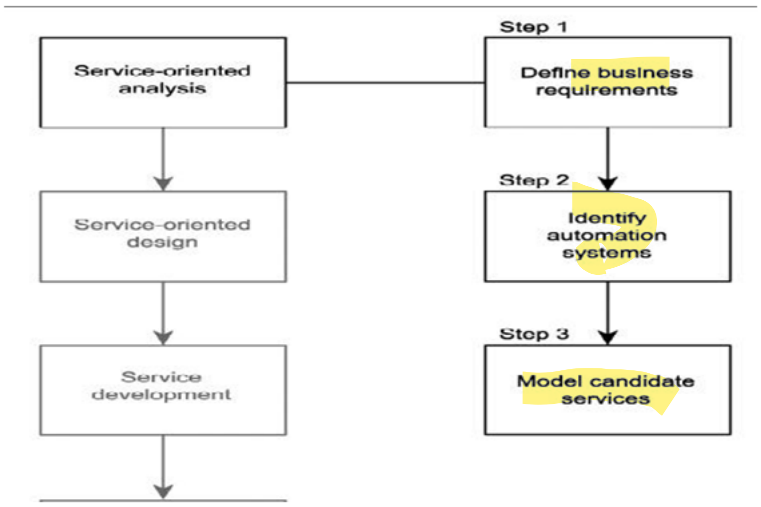
Service Oriented Analysis : The process of determining how business automation requirements can be represented through service-orientation. **Service-orientation emphasizes the creation of very specific design characteristics, while also de-emphasizing others.**

Questions that should be answered prior to proceeding with the service−oriented analysis include:

- What outstanding work is needed to establish the required business model(s)?
- What modeling tools will be used to carry out the analysis?
- Will the analysis be part of an SOA transition plan?



Service−oriented analysis steps

Step 1: Define business automation requirements

- Business requirements are gatheredand documented.
- Business requirements should be sufficiently mature so that a high−level automation process can be defined. This business process documentation will be used as the starting point of the service modeling process described in Step 3.

Step 2: Identify existing automation systems

- Existing application logic that is already, to whatever extent, automating any of therequirements identified in Step 1 needs to be identified.

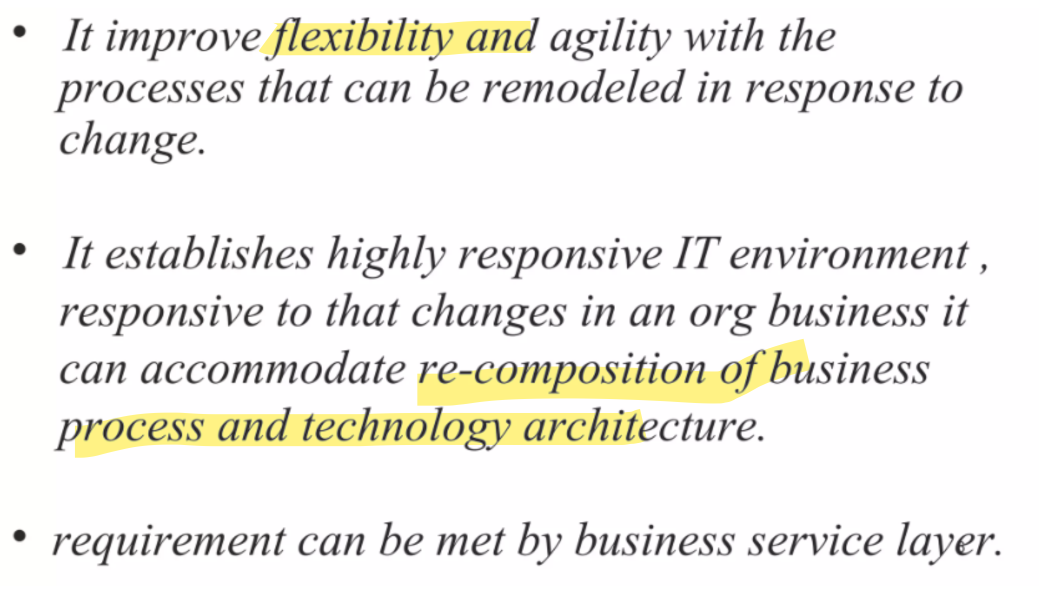
- This information will be used to help identify application service candidates during the service modeling process described in Step 3.

Step 3: Model candidate services

- A service–oriented analysis introduces the concept of service modeling process by which service operation candidates are identified and then grouped into a logical context.

Benefits of a business-centric SOA

1.Business services build agility into business models



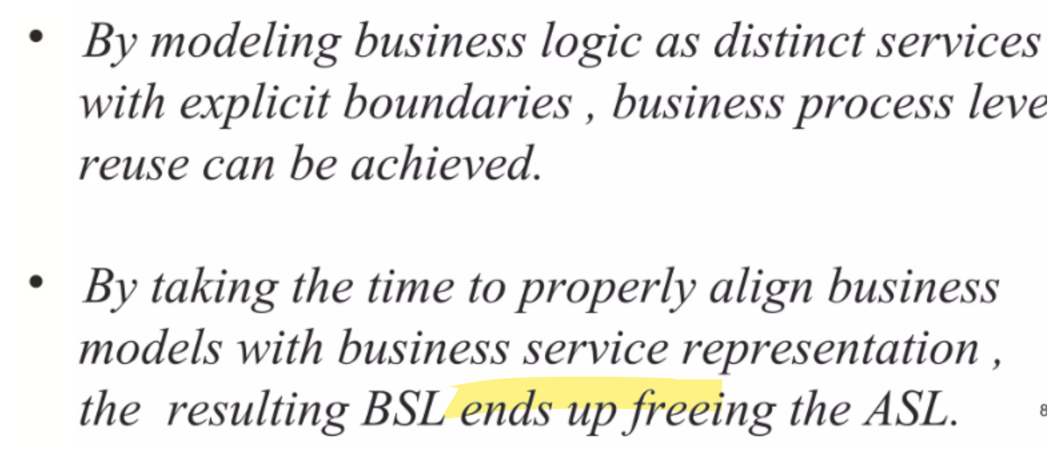
- *It improve flexibility and agility with the processes that can be remodeled in response to change.*
- *It establishes highly responsive IT environment , responsive to that changes in an org business it can accommodate re-composition of business process and technology architecture.*
- *requirement can be met by business service layer.*

2.Business services prepare a process for orchestration

The processes are modelled in a such a way that they can be easily moved to orchestration

3.ENABLE REUSE:

The creation of a business service layer promotes reuse in both business and application services, as follows:


- *By modeling business logic as distinct services with explicit boundaries , business process leve reuse can be achieved.*
- *By taking the time to properly align business models with business service representation , the resulting BSL ends up freeing the ASL.*

8

Deriving Business Services:

The inner workings of any organization, regardless of structure or size, can be decomposed into a collection of business services. A business service simply represents a logical unit of work, and pretty much anything any organization does consists of unitsof work. The service–orientation–aware analyst must be able to determine how to best map existing logic to services.

When building various types of services, it becomes evident that they can be categorized depending on:

- the type of logic they encapsulate
- the extent of reuse potential this logic has
- how this logic relates to existing domains within the enterprise

As a result, there are three common classifications that represent the primary service models

- Entity Services
- Task Services
- Utility Services

Entity Services

can be used by most [parent buisness.

In just about every enterprise, there will be business model documents that define the organization's relevant business entities. Examples of business entities include customer, employee, invoice, and claim. The *entity service* model (Figure 3.18) represents a business-centric service that bases its functional boundary and context on one or more related business entities. It is considered a highly reusable service because it is agnostic to most parent business processes. As a result, a single entity service can be leveraged to automate multiple parent business processes.

Entity services are also known as *entity-centric business services* or *business entity services*.

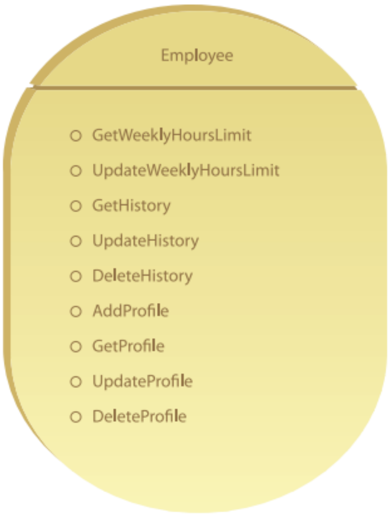


Figure 3.18
An example of an entity service. Several of its capabilities are reminiscent of traditional CRUD (create, read, update, delete) methods.

Task Services

A business service with a functional boundary directly associated with a specific parent business task or process is based on the *task service* model (Figure 3.19). This type of service tends to have less reuse potential and is generally positioned as the controller of a composition responsible for composing other, more process-agnostic services.

When discussing task services, one point of clarification often required is in relation to entity service capabilities. Each

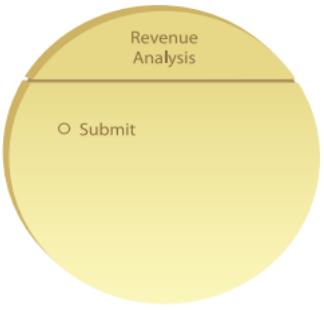


Figure 3.19
An example of a task service with a sole exposed capability required to initiate its encapsulated parent business process.

capability essentially encapsulates business process logic in that it carries out a sequence of steps to complete a specific task.

An entity Invoice service, for example, may have an Add capability that contains process logic associated with creating a new invoice record. How then is what a task service encapsulates different from what an entity service's capabilities contain? The primary distinction has to do with the functional scope of the capability.

The Invoice service’s Add capability is focused solely on the processing of an invoice document. To carry out this process may require that the capability logic interact with other services representing different business entities, but the functional scope of the capability is clearly associated with the functional context of the Invoice service.

If, however, we had a billing consolidation process that retrieved numerous invoice and PO records, performed various calculations, and further validated consolidation results against client history billing records, we would have process logic that spans multiple entity domains and does not fit cleanly within a functional context associated with a business entity.

This would typically constitute a “parent” process in that it consists of processing logic that needs to coordinate the involvement of multiple services. Services with a functional context defined by a parent business process or task can be developed as standalone Web services or components—or—they may represent a business process definition hosted within an orchestration platform. In the latter case, the design characteristics of the service are somewhat distinct due to the specific nature of the underlying technology. In this case, it may be preferable to qualify the service model label accordingly. This type of service is referred to as the orchestrated task service.

Task services are also known as task-centric business services

Utility Services

Each of the previously described service models has a very clear focus on representing business logic. However, within the realm of automation, there is not always a need to associate logic with a business model or process. In fact, it can be highly beneficial to deliberately establish a functional context that is *non*-business-centric. This essentially results in a distinct, technology-oriented service layer.

The *utility service* model (Figure 3.20) accomplishes this. It is dedicated to providing reusable, cross-cutting utility functionality, such as event logging, notification, and exception handling. It is ideally application agnostic in that it can consist of a series of capabilities that draw from multiple enterprise systems and resources, while making this functionality available within a very specific processing context.

Utility services are also known as *application services*, *infrastructure services*, or *technology services*.

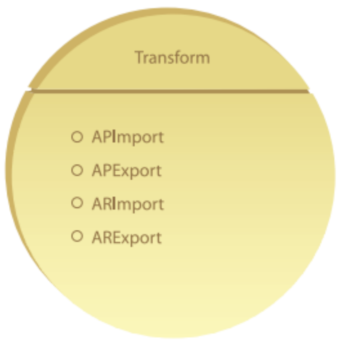


Figure 3.20
An example of a utility service providing a set of capabilities associated with proprietary data format transformation.

SERVICE DESCRIPTION:

Service description is a key to making the SOA loosely coupled and reducing the amount of required common understanding, custom programming, and integration between the service provider and the service requestor's applications.

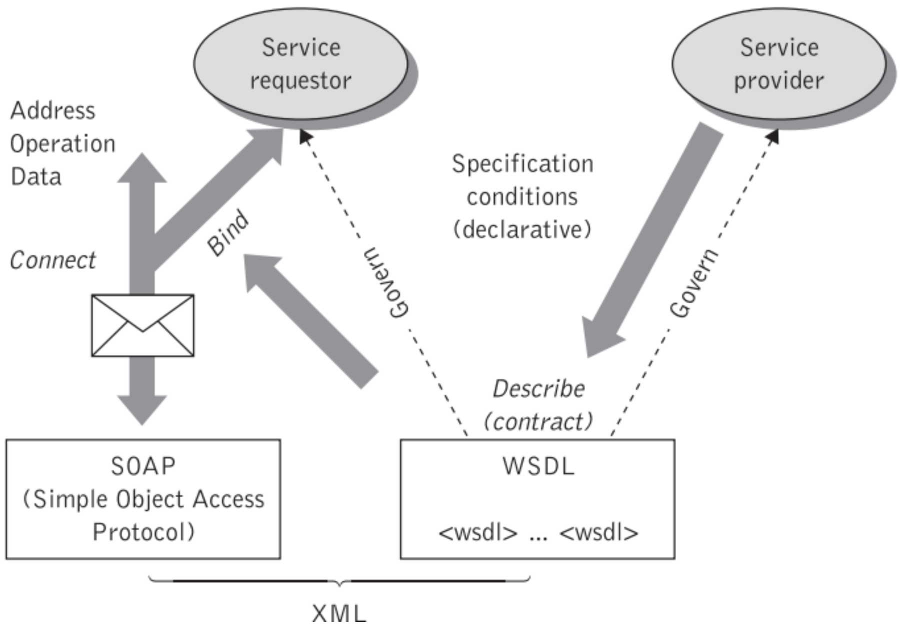
important NFPs of software systems include security, reliability, availability, efficiency, scalability, and fault-tolerance.

Service description is a machine-understandable specification describing the structure, operational characteristics, and non-functional properties of a Web service.

The primary service description document is the WSDL definition.

A WSDL describes the point of contact for a service provider, also known as the service endpoint or just endpoint. It provides a formal definition of the endpoint interface (so that requestors wishing to communicate with the service provider know exactly how to structure request messages) and also establishes the physical location (address) of the service.

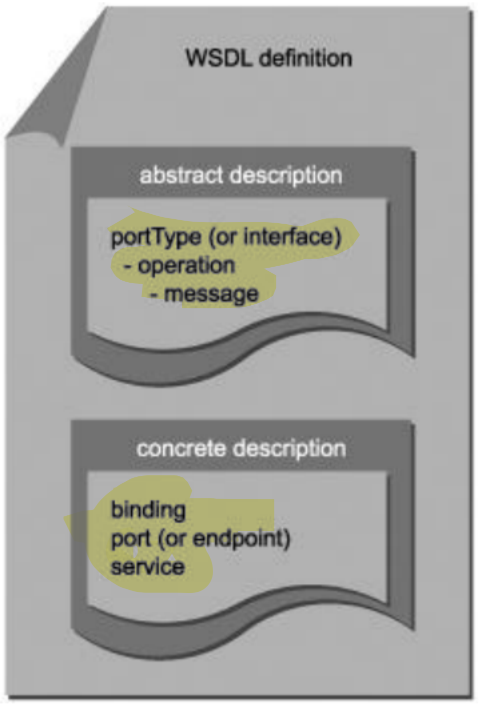
WSDL provides a mechanism by which service providers can describe the basic format of Web requests over different protocols (e.g., SOAP) or encoding (e.g., Multipurpose Internet Messaging Extensions or MIME). WSDL is used simply to describe operations. In doing so it does not indicate the order of execution of these operations.



WSDL service description governing interaction between a service requestor and a service provider

Let's dig a bit deeper into how the service description document itself is organized. A WSDL service description (also known as WSDL service definition or just WSDL definition) can be separated into two categories:

- abstract description
- concrete description



WSDL document consisting of abstract and concrete parts that collectively describe service endpoint

Abstract description :

An abstract description establishes the interface characteristics of the Web service without any reference to the technology used to host or enable a Web service to transmit messages.

By separating this information, the integrity of the service description can be preserved regardless of what changes might occur to the underlying technology platform.

Below is a description of the three main parts that comprise an abstract description

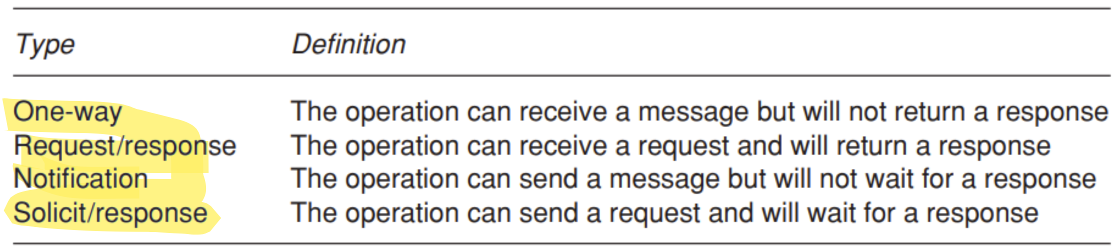
portType, operation, and message

The parent portType section of an abstract description provides a high-level view of the service interface by sorting the messages a service can process into groups of functions known as operations.

Each operation represents a specific action performed by the service.

Much like component methods, operations also have input and output parameters. Because Web services rely exclusively on messaging-based communication, parameters are represented as messages. Therefore, an operation consists of a set of input and output messages

Note that the transmission sequence of these messages can be governed by a predetermined message exchange pattern that also is associated with the operation.



Type	*Definition*
One-way	The operation can receive a message but will not return a response
Request/response	The operation can receive a request and will return a response
Notification	The operation can send a message but will not wait for a response
Solicit/response	The operation can send a request and will wait for a response

Summary of WSDL message exchange patterns

Concrete description

For a Web service to be able to execute any of its logic, it needs for its abstract interface definition to be connected to some real, implemented technology. This connection is defined in the concrete description portion of the WSDL file, which consists of three related parts

binding, port, and service

A WSDL description's binding describes the requirements for a service to establish physical connections with the service. In other words, a binding represents one possible transport technology the service can use to communicate. SOAP is the most common form of binding, but others also are supported. A binding can apply to an entire interface or just a specific operation.

Related to the binding is the port, which represents the physical address at which a service can be accessed with a specific protocol. This piece of physical implementation data exists separately to allow location information to be maintained independently from other aspects of the concrete description.

Within the WSDL language, the term service is used to refer to a group of related endpoints.

Difference between Abstract and Concrete WSDL:

- An abstract WSDL document describes what the web service does, but not how it does it or how to contact it. An abstract WSDL document defines:

  the operations provided by the web service.

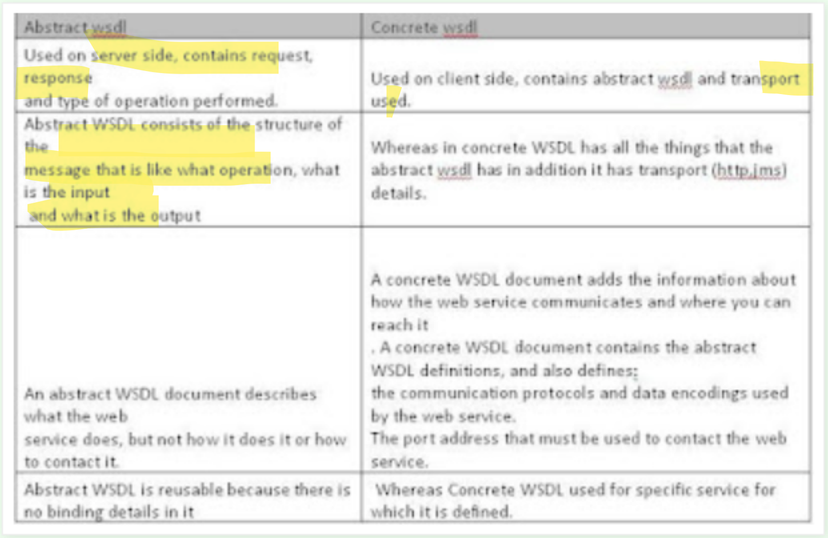
  The input, output and fault messages used by each operation to communicate with the web service, and their format.

- A concrete WSDL document adds the information about how the web service communicates and where you can reach it. A concrete WSDL document contains the abstract WSDL definitions, and also defines:

  the communication protocols and data encodings used by the web service.

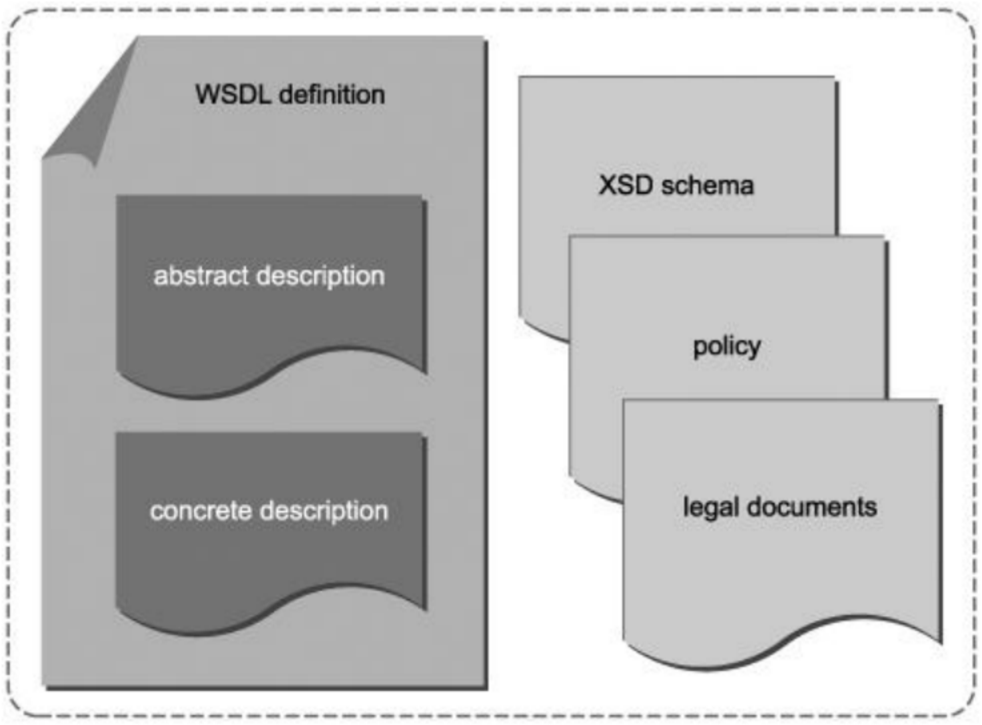
  The port address that must be used to contact the web service.

WSDL definitions frequently rely on XSD schemas to formalize the structure of incoming and outgoing messages. Another common supplemental service description document is a policy. Policies can provide rules, preferences, and processing details above and beyond what is expressed through the WSDL and XSD schema documents.



Abstract wsdl	Concrete wsdl
Used on server side, contains request, response and type of operation performed.	Used on client side, contains abstract wsdl and transport used.
Abstract WSDL consists of the structure of the message that is like what operation, what is the input and what is the output	Whereas in concrete WSDL has all the things that the abstract wsdl has in addition it has transport (http,jms) details.
An abstract WSDL document describes what the web service does, but not how it does it or how to contact it.	A concrete WSDL document adds the information about how the web service communicates and where you can reach it . A concrete WSDL document contains the abstract WSDL definitions, and also defines: the communication protocols and data encodings used by the web service. The port address that must be used to contact the web service.
Abstract WSDL is reusable because there is no binding details in it	Whereas Concrete WSDL used for specific service for which it is defined.

Note that a service contract can refer to additional documents or agreements not expressed by service descriptions. For example, a Service Level Agreement (SLA) agreed upon by the respective owners of a service provider and its requestor can be considered part of an overall service contract



service contract