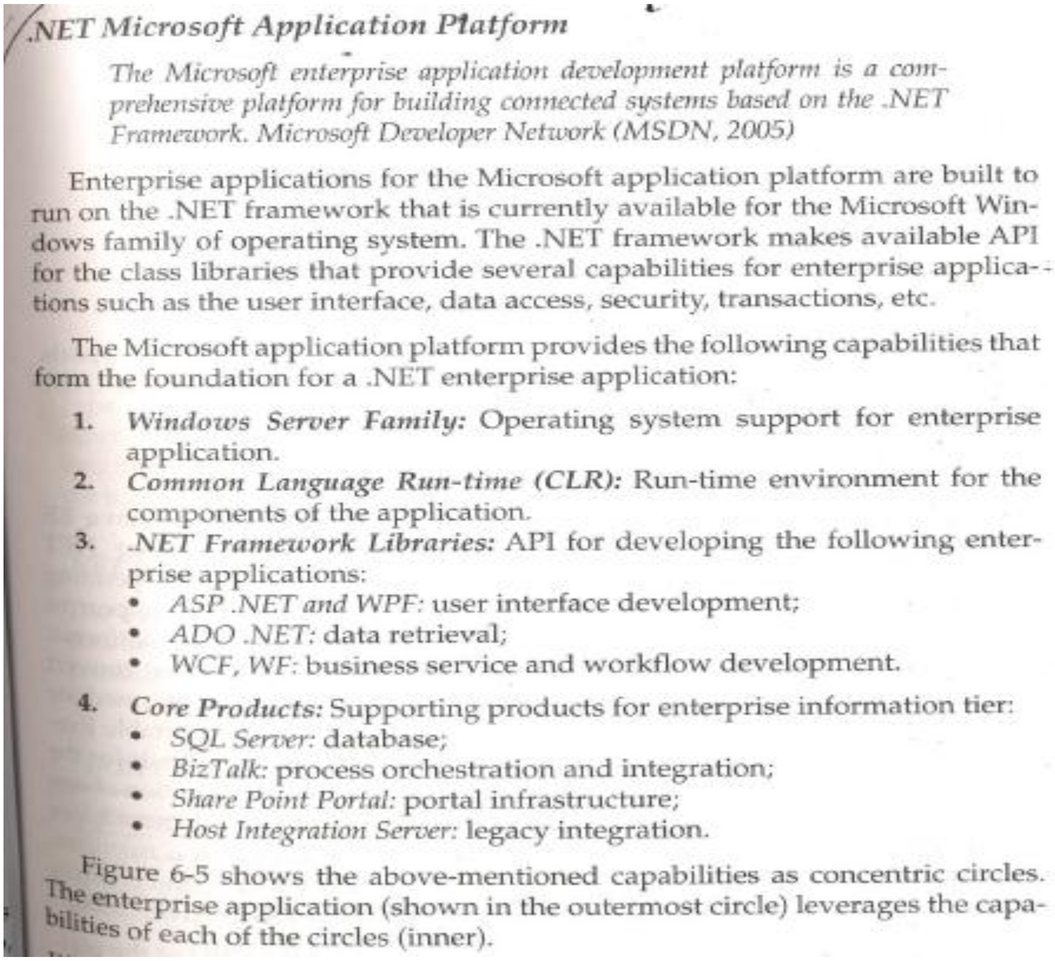
Service Oriented Architecture – 16MCA553

Nov 2018 – Internal Test - 3 – Answer Key

1. Elaborate support for SOA in .NET platform



.NET Microsoft Application Platform

> *The Microsoft enterprise application development platform is a comprehensive platform for building connected systems based on the .NET Framework. Microsoft Developer Network (MSDN, 2005)*

Enterprise applications for the Microsoft application platform are built to run on the .NET framework that is currently available for the Microsoft Windows family of operating system. The .NET framework makes available API for the class libraries that provide several capabilities for enterprise applications such as the user interface, data access, security, transactions, etc.

The Microsoft application platform provides the following capabilities that form the foundation for a .NET enterprise application:

1. ***Windows Server Family:*** Operating system support for enterprise application.
2. ***Common Language Run-time (CLR):*** Run-time environment for the components of the application.
3. ***.NET Framework Libraries:*** API for developing the following enterprise applications:
   - *ASP .NET and WPF:* user interface development;
   - *ADO .NET:* data retrieval;
   - *WCF, WF:* business service and workflow development.
4. ***Core Products:*** Supporting products for enterprise information tier:
   - *SQL Server:* database;
   - *BizTalk:* process orchestration and integration;
   - *Share Point Portal:* portal infrastructure;
   - *Host Integration Server:* legacy integration.

Figure 6-5 shows the above-mentioned capabilities as concentric circles. The enterprise application (shown in the outermost circle) leverages the capabilities of each of the circles (inner).

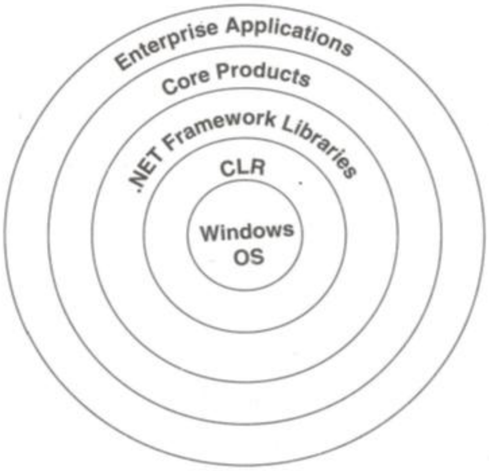


FIGURE 6-5 .NET Microsoft application platform.

Windows OS :

Microsoft family of OS supports the .NET framework and CLR essential for an enterprise application. While in theory CLR can be ported to other OS only Windows family of OS supports it as of now. The open source Mono project is an effort to provide the >NET framework and CLR o the Unix family of OS.

Common Language Run-Time (CLR)

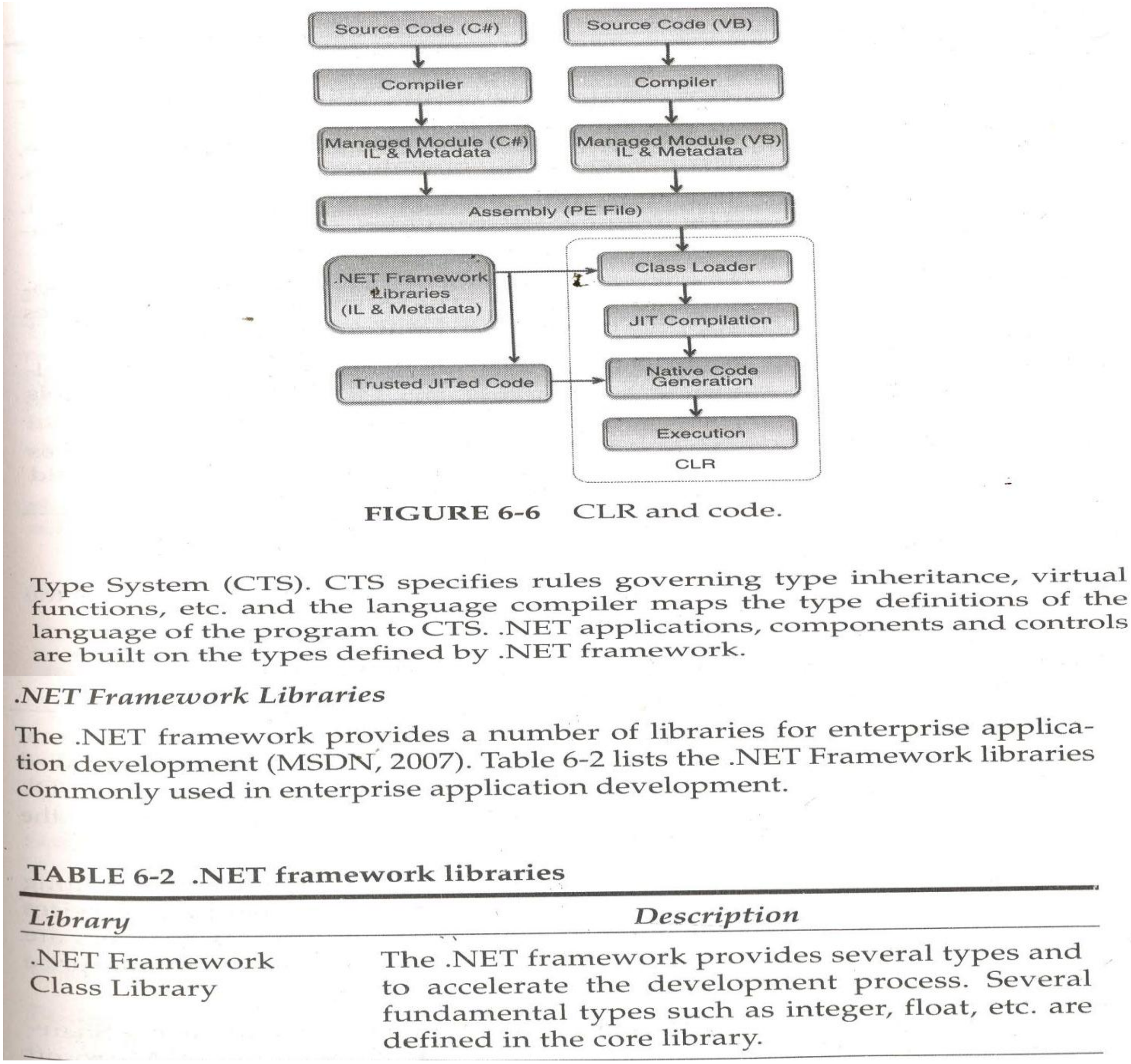

Source Code (C#)
Source Code (VB)
Compiler
Compiler
Managed Module (C#) IL & Metadata
Managed Module (VB) IL & Metadata
Assembly (PE File)
.NET Framework Libraries (IL & Metadata)
Class Loader
JIT Compilation
Trusted JITed Code
Native Code Generation
Execution
CLR

FIGURE 6-6 CLR and code.

Type System (CTS). CTS specifies rules governing type inheritance, virtual functions, etc. and the language compiler maps the type definitions of the language of the program to CTS. .NET applications, components and controls are built on the types defined by .NET framework.

.NET Framework Libraries

The .NET framework provides a number of libraries for enterprise application development (MSDN, 2007). Table 6-2 lists the .NET Framework libraries commonly used in enterprise application development.

TABLE 6-2 .NET framework libraries

Library	*Description*
.NET Framework Class Library	The .NET framework provides several types and to accelerate the development process. Several fundamental types such as integer, float, etc. are defined in the core library.

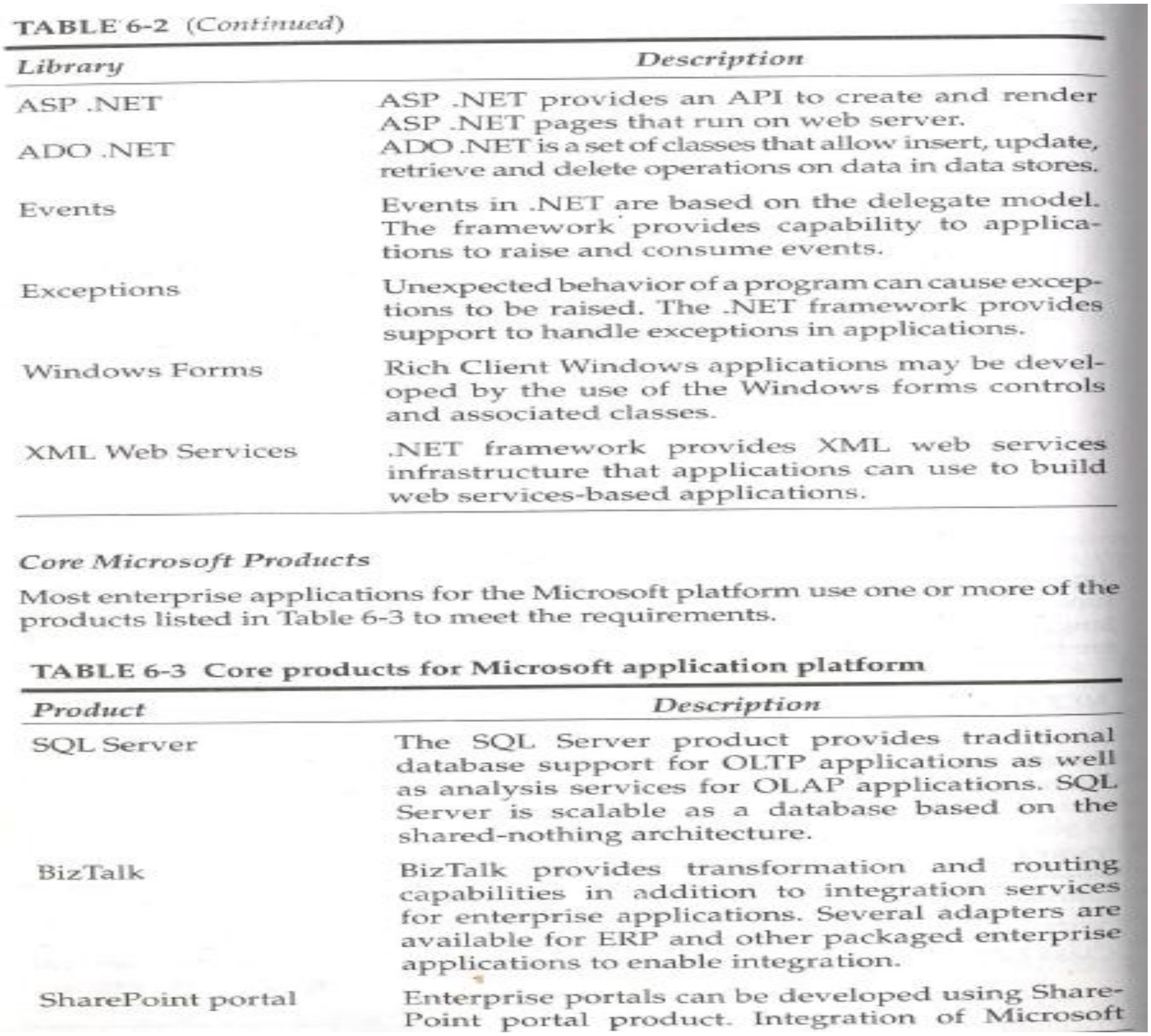


TABLE 6-2 (*Continued*)

Library	*Description*
ASP .NET	ASP .NET provides an API to create and render ASP .NET pages that run on web server.
ADO .NET	ADO .NET is a set of classes that allow insert, update, retrieve and delete operations on data in data stores.
Events	Events in .NET are based on the delegate model. The framework provides capability to applications to raise and consume events.
Exceptions	Unexpected behavior of a program can cause exceptions to be raised. The .NET framework provides support to handle exceptions in applications.
Windows Forms	Rich Client Windows applications may be developed by the use of the Windows forms controls and associated classes.
XML Web Services	.NET framework provides XML web services infrastructure that applications can use to build web services-based applications.

Core Microsoft Products

Most enterprise applications for the Microsoft platform use one or more of the products listed in Table 6-3 to meet the requirements.

TABLE 6-3 Core products for Microsoft application platform

Product	*Description*
SQL Server	The SQL Server product provides traditional database support for OLTP applications as well as analysis services for OLAP applications. SQL Server is scalable as a database based on the shared-nothing architecture.
BizTalk	BizTalk provides transformation and routing capabilities in addition to integration services for enterprise applications. Several adapters are available for ERP and other packaged enterprise applications to enable integration.
SharePoint portal	Enterprise portals can be developed using SharePoint portal product. Integration of Microsoft

TABLE 6-3 (*Continued*)

Product	*Description*
	Office with SharePoint services (MOSS) has resulted in more sophisticated Microsoft Office front-ends for SharePoint server.
Host Integration Server	Host Integration server provides integration of .NET enterprise applications with mainframe systems.

Reference Model

The logical organization of a .NET enterprise application in presentation, business and data access layers is shown in Figure 6-7. It represents a reference model for a layered web-based .NET application that addresses the concerns of each layer based on the concept of design patterns applicable to .NET platform. As with any reference model, the objective is to provide a template for architecture rather than be comprehensive in coverage of possible scenarios.

Several design patterns have been identified for .NET platform (Trowbridge et al., 2003). Table 6-4 lists some of the key patterns that form the basis for a reference model shown in Figure 6-7.

The presentation layer shown in the reference model includes the MVC pattern to address the user interface concerns of the enterprise application. Other

TABLE 6-4 Key patterns for .NET reference model

Pattern	*Description*
Model–view–controller	Pattern suitable for providing multiple views of data. Separates three types of objects – *models* that maintains data, *views* that displays all or a portion of the data, and *controller* that handles events that affect the model or view.
Service interface	Pattern to decouple presentation layer from business layer while exposing functionality of business layer as services. The service interface supports and encapsulates network protocol used for communication between service consumer and the service provider.
Broker	Pattern to communicate with remote objects. The broker pattern hides the implementation details of communication with remote objects by encap-

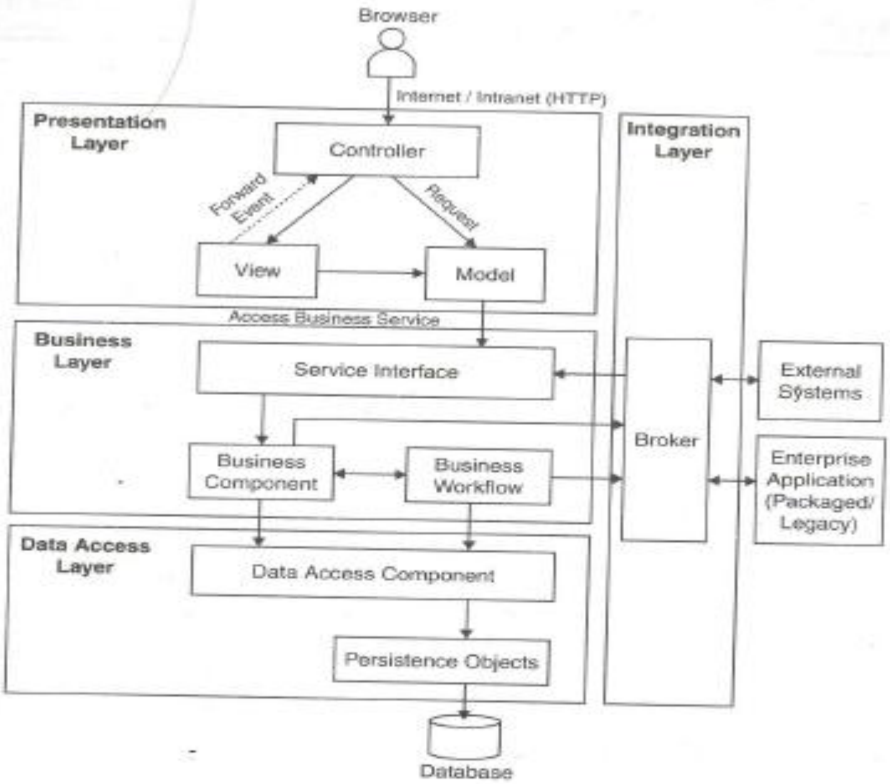
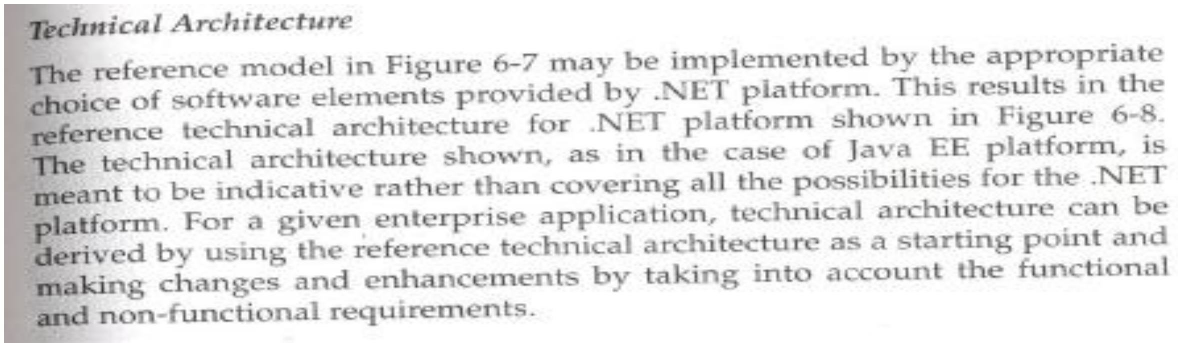


FIGURE 6-7 Reference model of .NET enterprise application.

patterns such as *model–view–presenter* and *page controller* can also be applied to address the concerns of presentation layer. ASP .NET, with the concept of *code behind*, provides most of the support needed for handling HTTP requests from users and implementing the MVC pattern.

The *service interface* pattern encapsulates the functionality provided by the business layer. It supports multiple transports such as Web Services, MSMQ and .NET Remoting and hence the same functionality can be exposed to different types of clients by the use of *service interface* pattern. Windows Communication Foundation (WCF) can provide the capability required to implement the *service interface* pattern. Business components and workflows implement the business functionality of the enterprise application. BizTalk and Windows Foundation (WF) are two options to implement workflows required in an enterprise application.

The *data access components* encapsulate access to the databases. ADO .NET model provides support for different operations on databases and can be used by *data access components* for efficient access of data.



Technical Architecture

The reference model in Figure 6-7 may be implemented by the appropriate choice of software elements provided by .NET platform. This results in the reference technical architecture for .NET platform shown in Figure 6-8. The technical architecture shown, as in the case of Java EE platform, is meant to be indicative rather than covering all the possibilities for the .NET platform. For a given enterprise application, technical architecture can be derived by using the reference technical architecture as a starting point and making changes and enhancements by taking into account the functional and non-functional requirements.

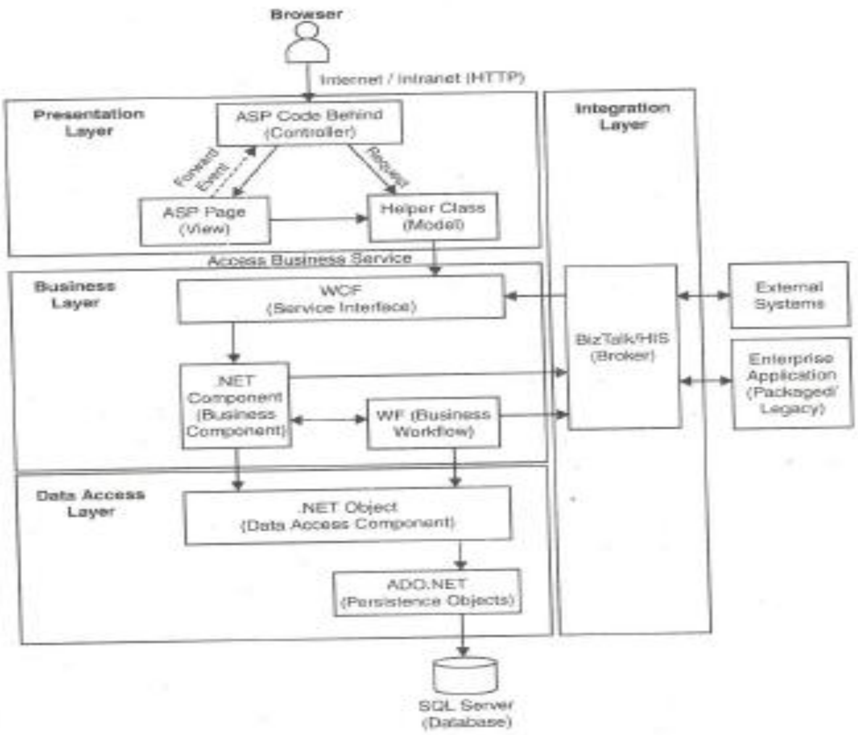
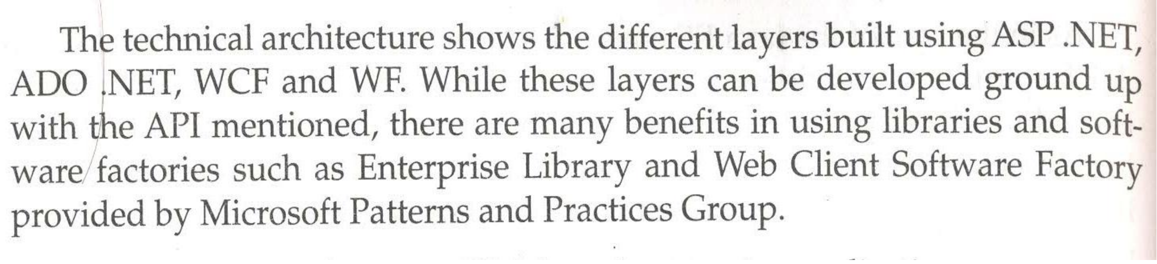


FIGURE 6-8 Technical architecture (indicative) of .NET enterprise application.



The technical architecture shows the different layers built using ASP .NET, ADO .NET, WCF and WF. While these layers can be developed ground up with the API mentioned, there are many benefits in using libraries and software factories such as Enterprise Library and Web Client Software Factory provided by Microsoft Patterns and Practices Group.

2. Describe about WS-Notification Framework.

As with other WS-* frameworks, what is represented by WS-Notification is a family of related extensions that have been designed with composability in mind.

- WS-BaseNotification—Establishes the standardized interfaces used by services involved on either end of a notification exchange.
- WS-Topics—Governs the structuring and categorization of topics.
- WS-BrokeredNotification—Standardizes the broker intermediary used to send and receive messages on behalf of publishers and subscribers.

Situations, notification messages, and topics

- The notification process typically is tied to an event that is reported on by the publisher.
- This event is referred to as a *situation*. Situations can result in the generation of one or more *notification messages*. These messages contain information about (or relating to) the situation, and are categorized according to an available set of *topics*.
- Through this categorization, notification messages can be delivered to services that have subscribed to
- corresponding topics.

Notification producers and publishers

- The term *publisher* represents the part of the solution that responds to situations and is responsible for generating notification messages. However, a publisher is not necessarily required to distribute these messages. Distribution of notification messages is the task of the *notification producer*.
- This service keeps track of subscriptions and corresponds directly with subscribers. It ensures that notification messages are organized by topic and delivered accordingly.

Notification consumers and subscribers

- A *subscriber* is the part of the application that submits the subscribe request message to the notification producer. This means that the subscriber is not necessarily the recipient of the notification messages transmitted by the notification producer.
- The recipient is the *notification consumer*, the service to which the notification messages are delivered

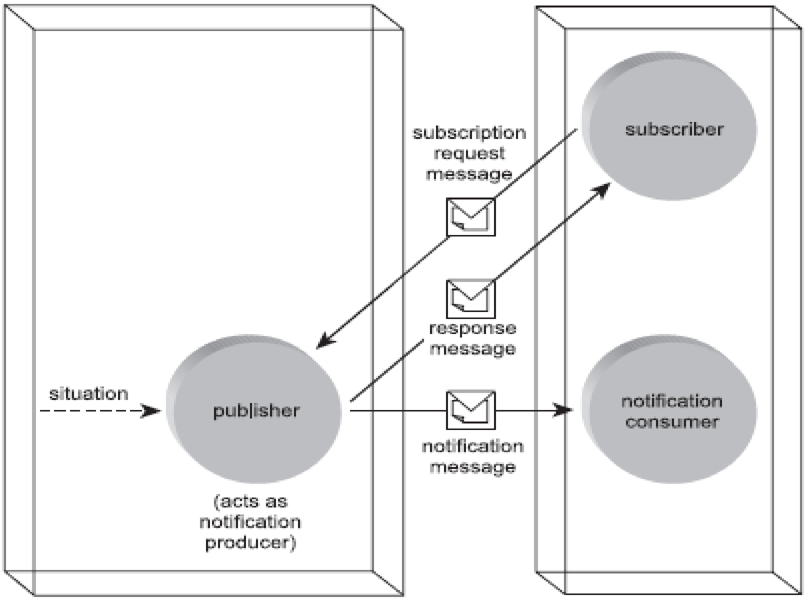


Figure 7.38
A basic notification architecture.

Notification broker, publisher registration manager, and subscription manager

- The *notification broker*—A Web service that acts on behalf of the publisher to perform the role of the notification producer. This isolates the publisher from any contact with subscribers. Note that when a notification broker receives notification messages from the publisher, it temporarily assumes the role of notification consumer.

- The *publisher registration manager*—A Web service that provides an interface for subscribers to search through and locate items available for registration. This role may be assumed by the notification broker, or it may be implemented as a separate service to establish a further layer of abstraction.
- The *subscription manager*—A Web service that allows notification producers to access and retrieve required subscriber information for a given notification message broadcast. This role also can be assumed by either the notification producer or a dedicated service.

Figure 7.39

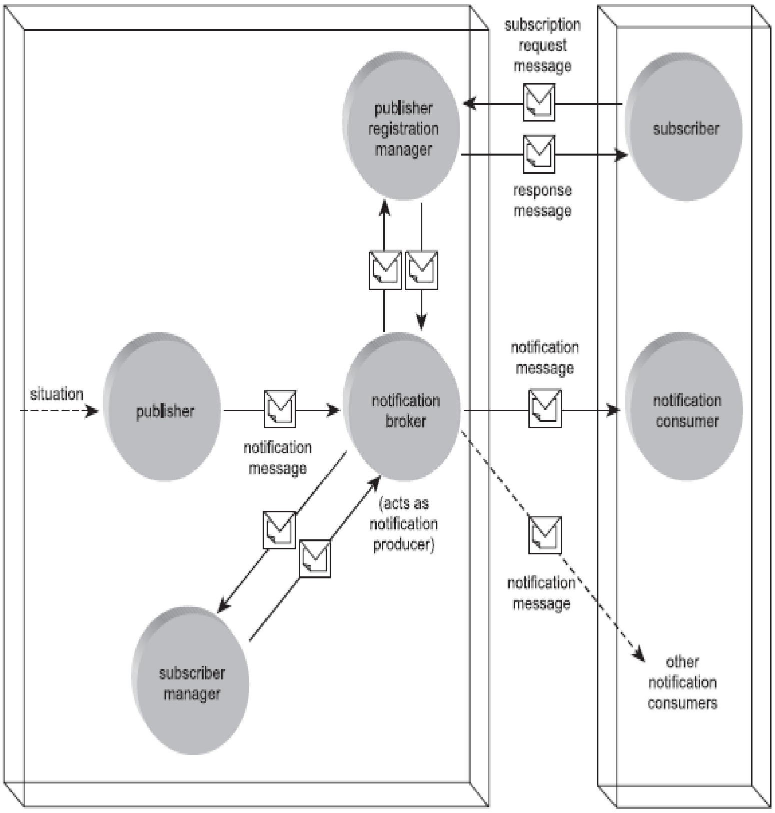


Figure 7.39
A notification architecture including a middle tier.

3.a Explain the Application service layer

Provide reusable functions related to processing data within legacy or new application environments

□ Characteristics

□ they expose functionality within a specific processing context

□ they draw upon available resources within a given platform

□ they are solution-agnostic

□ they are generic and reusable

□ they can be used to achieve point-to-point integration with other application services

□ they are often inconsistent in terms of the interface granularity they expose

□ they may consist of a mixture of custom-developed services and third-party services that have been purchased or leased

□ **Utility service**

□ When a separate business service layer exists, then turn all application services into generic utility services

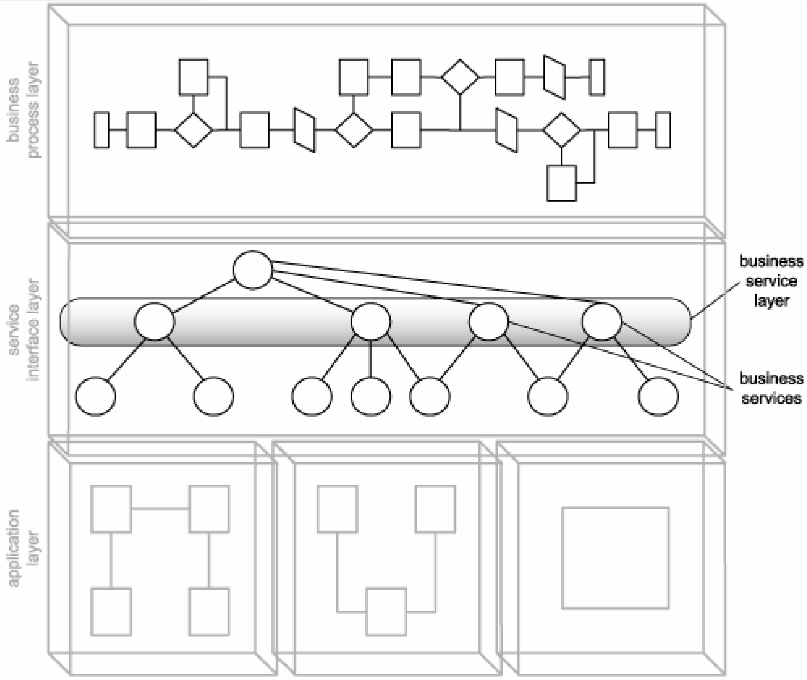
- **Wrapper service**
- Wrapper services most often are utilized for integration purposes. They consist of services that encapsulate ("wrap")
some or all parts of a legacy environment to expose legacy functionality to service requestors
- **Proxy service or auto-generated WSDL**
- Another variation of the wrapper service model
- This simply provides a WSDL definition that mirrors an existing component interface

Hybrid application services/hybrid services

- Services that contain both application and business logic can be referred to as **hybrid application services** or just **hybrid services**. This service model is commonly found within traditional distributed architectures
- **Application integration services /Integration services**
- Application services that exist solely to enable integration between systems often are referred to as **application integration services** or simply
integration services. Integration services often are implemented as controllers

3.b Discuss in detail about the Business service layer

While application services are responsible for representing technology and application logic, the business service layer introduces a service concerned solely with representing business logic, called the business service



Business service layer abstraction leads to the creation of two further business service models:

1. Task-centric business service A service that encapsulates business logic specific to a task or business process. This type of service generally is required when business process logic is not centralized as part of an orchestration layer. Task-centric business services have limited reuse potential.
2. Entity-centric business service A service that encapsulates a specific business entity (such as an invoice or timesheet). Entity-centric services are useful for creating highly reusable and business process-agnostic services that are composed by an orchestration layer or by a service layer consisting of task-centric business services (or both).

4. What is mean by technology framework? Briefly explain web service framework

- A technology framework is a collection of things.
- It can include one or more architectures, technologies, concepts, models, and even sub-frameworks.

Framework is characterized by:

- an abstract (vendor-neutral) existence defined by standards organizations and implemented by (proprietary) technology platforms
- core building blocks that include Web services, service descriptions, and messages
- a communications agreement centered around service descriptions based on WSDL
- a messaging framework comprised of SOAP technology and concepts
- a service description registration and discovery architecture sometimes realized through UDDI
- a well-defined architecture that supports messaging patterns and compositions
- a second generation of Web services extensions (also known as the WS-* specifications) continually broadening its underlying feature-set

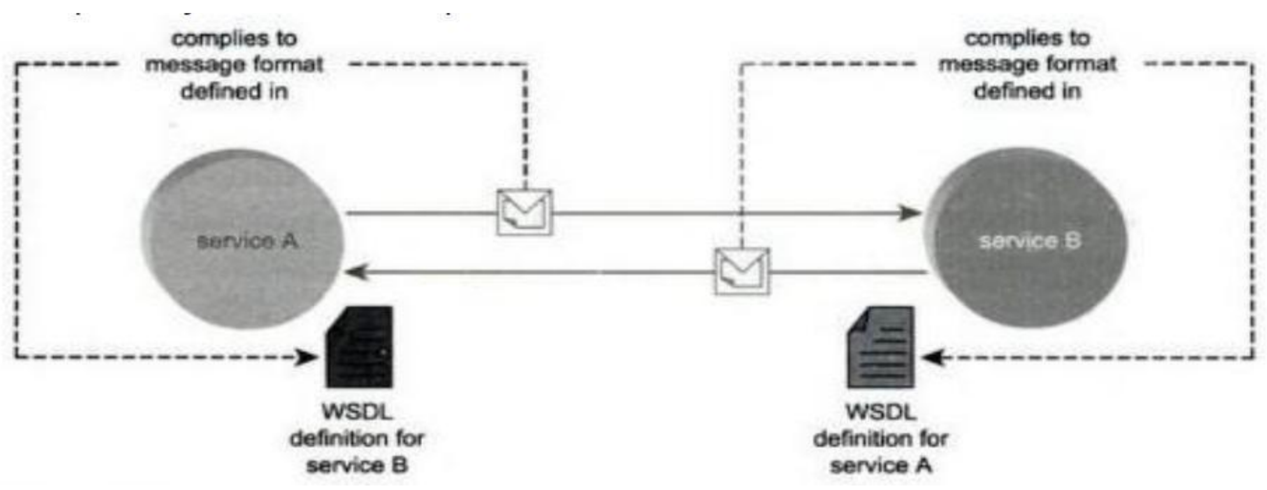
. Services (as Web services)

- services - how they provide a means of encapsulating various extents of logic.
- Manifesting services in real world automation solutions requires the use of a technology capable of preserving fundamental service-orientation, while implementing real world business functionality.
- Web services provide the potential of fulfilling these primitive requirements
- Web services framework is flexible and adaptable.
- Web services can be designed to duplicate the behavior and functionality found in proprietary distributed systems, or they can be designed to be fully SOA-compliant.
- This flexibility has allowed Web services to become part of many existing application environments
- Fundamentally, every Web service can be associated with:
  - a temporary classification based on the roles it assumes during the runtime processing of a message
  - a permanent classification based on the application logic it provides and the roles it assumes within a solution environment
- We explore both of these design classifications in the following two sections:
  - service roles (temporary classifications)
  - service models (permanent classifications)

WSDL (Web Services Description Language**)**

- Service Description provides the key to establishing a consistently loosely coupled form of communication between services implemented as Web services.
- Description documents are required to accompany any service wanting to act as an ultimate receiver.

The primary service description document is the WSDL definition


complies to message format defined in
complies to message format defined in
service A
service B
WSDL definition for service B
WSDL definition for service A

SOAP (Simple Object Access Protocol**)**

All communication between services is message-based,

 The messaging framework chosen must be standardized so that all services, regardless of origin, use the same format and transport protocol.

 Message-centric application design that an increasing amount of business and application logic is embedded into messages.

 The SOAP specification was chosen to meet all of these requirements

 Universally accepted as the standard transport protocol for messages processed by Web services

5.4.1. Messages

Simple Object Access *Protocol*, the SOAP specification's main purpose is to define a standard message format.

The structure of this format is quite simple, but its ability to be extended and customized

Envelope, header, and body

Every SOAP message is packaged into a container known as an *envelope*.

Much like the metaphor this conjures up, the envelope is responsible for housing all parts of the message

UDDI (Universal Description Discovery and Integration)

Private and public registries

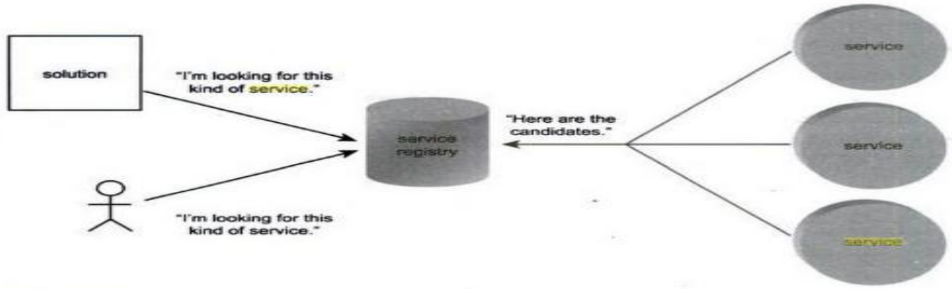
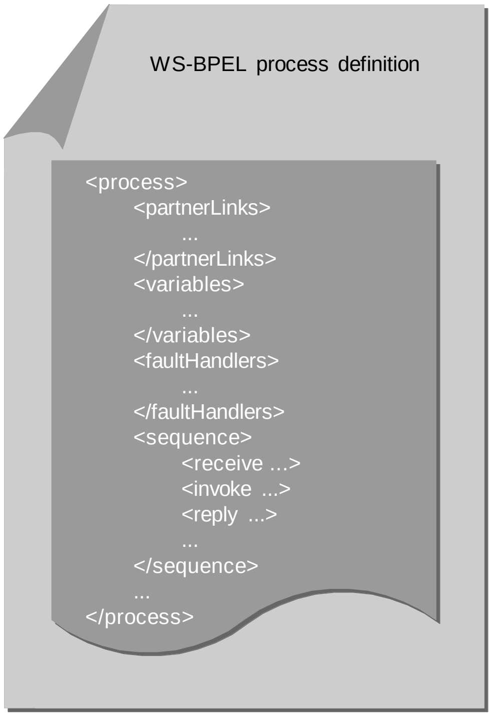

solution
"I'm looking for this kind of service."
service registry
"Here are the candidates."
service
service
service
"I'm looking for this kind of service."

Figure 5.18
Service description locations centralized in a registry.

- UDDI accepted standard for structuring registries that keep track of service descriptions
- These registries can be searched manually and accessed programmatically via a standardized API.

5. Explain the basics of WS-BPEL Language

WS-BPEL language basics



```
<process>
      <partnerLinks>
            ...
      </partnerLinks>
      <variables>
            ...
      </variables>
      <faultHandlers>
            ...
      </faultHandlers>
      <sequence>
            <receive ...>
            <invoke ...>
            <reply ...>
            ...
      </sequence>
      ...
</process>
```

Figure 16-1 ***A common WS-BPEL process definition structure.***

A brief history of BPEL4WS and WS-BPEL

- The Business Process Execution Language for Web Services (BPEL4WS) was first conceived in July, 2002 with the release of the BPEL4WS 1.0 specification, a joint effort by IBM, Microsoft, and BEA.
- This document proposed an orchestration language inspired by previous variations, such as IBM’s Web Services Flow Language (WSFL) and Microsoft’s XLANG specification.
- Next version of BPEL4WS is WS-BPEL Prerequisites

The process element

- BPEL processes are exposed as WSDL services †
    - Message exchanges map to WSDL operations †
    - WSDL can be derived from partner definitions and the role played by the process in interaction with partners †
    - BPEL processes interact with WSDL services exposed by business partners

```
<process name="TimesheetSubmissionProcess"
    targetNamespace="http://www.xmltc.com/tls/process/"
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:bpl="http://www.xmltc.com/tls/process/"
    xmlns:emp="http://www.xmltc.com/tls/employee/"
    xmlns:inv="http://www.xmltc.com/tls/invoice/"
    xmlns:tst="http://www.xmltc.com/tls/timesheet/"
    xmlns:not="http://www.xmltc.com/tls/notification/">
      <partnerLinks>
            ...
      </partnerLinks>
      <variables>
            ...
      </variables>
      <sequence>
            ...
      </sequence>
      ...
</process>
```

Example 16-1 ***A skeleton process definition.***

The process construct contains a series of common child elements

The partnerLinks and partnerLink elements

A partnerLink element establishes the port type of the service (partner) that will be participating during the execution of the business process.
Partner services can act as a client to the process, responsible for invoking the process service.
Alternatively, partner services can be invoked by the process service itself.
The contents of a partnerLink element represent the communication exchange between two partners – the process service being one partner and another service being the other.

```
<partnerLinks>
      <partnerLink name="client"
            partnerLinkType="tns:TimesheetSubmissionType"
            myRole="TimesheetSubmissionServiceProvider"/>
      <partnerLink name="Invoice"
            partnerLinkType="inv:InvoiceType"
            partnerRole="InvoiceServiceProvider"/>
      <partnerLink name="Timesheet"
            partnerLinkType="tst:TimesheetType"
            partnerRole="TimesheetServiceProvider"/>
      <partnerLink name="Employee"
            partnerLinkType="emp:EmployeeType"
            partnerRole="EmployeeServiceProvider"/>
      <partnerLink name="Notification"
            partnerLinkType="not:NotificationType"
            partnerRole="NotificationServiceProvider"/>
```

```
</partnerLinks>
```

Example 16-2 ***The partnerLinks construct containing one partnerLink element in which the process service is invoked by an external client partner, and four partnerLink elements that identify partner services invoked by the process service.***

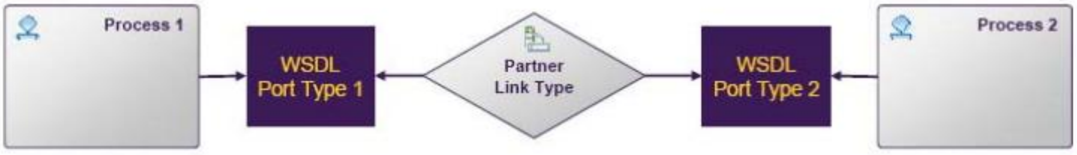
The partnerLinkType element

For each partner service involved in a process, partnerLinkType elements identify the WSDL portType elements referenced by the partnerLink elements within the process definition.
The partnerLinkType construct contains one role element for each role the service can play
Therefore, a partnerLinkType will have either one or two child role elements.

```
<definitions name="Employee"
        targetNamespace="http://www.xmltc.com/tls/employee/wsdl/"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
        ...
>
        ...
        <plnk:partnerLinkType name="EmployeeServiceType"
                xmlns="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
                <plnk:role name="EmployeeServiceProvider">
                        <portType name="emp:EmployeeInterface"/>
                </plnk:role>
        </plnk:partnerLinkType>
  ...
</definitions>
```

Example 16-3 ***A WSDL definitions construct containing a partnerLinkType construct.***

Note that multiple partnerLink elements can reference the same partnerLinkType. This is useful for when a process service has the same relationship with multiple partner services. All of the partner services can therefore use the same process service portType elements.



The variables element

Variables are used to define data containers „

- WSDL messages received from or sent to partners „
- Messages that are persisted by the process „
- XML data defining the process state
- messageType, element, or type.
- The messageType attribute allows for the variable to contain an entire WSDL-defined message,

- Element attribute simply refers to an XSD element construct.
- The type attribute can be used to just represent an XSD simpleType, such as string or integer.

```
<variables>
   <variable name="ClientSubmission"
               messageType="bpl:receiveSubmitMessage"/>
   <variable name="EmployeeHoursRequest"
               messageType="emp:getWeeklyHoursRequestMessage"/>
   <variable name="EmployeeHoursResponse"
               messageType="emp:getWeeklyHoursResponseMessage"/>
   <variable name="EmployeeHistoryRequest"
               messageType="emp:updateHistoryRequestMessage"/>
   <variable name="EmployeeHistoryResponse"
               messageType="emp:updateHistoryResponseMessage"/>
         ...
</variables>
```

Example 16-4 ***The variables construct hosting only some of the child variable elements used later by the Timesheet Submission Process.***

The getVariableProperty and getVariableData functions

getVariableProperty(variable name, property name)

- accepts the variable and property names as input and returns the requested value.

getVariableData(variable name, part name, location path)

This function is required to provide other parts of the process logic access to this data.
The getVariableData function has a mandatory variable name parameter, and two optional arguments that can be used to specify a specific part of the variable data.

In our examples we use the getVariableData function a number of times to retrieve message data from variables.

```
getVariableData('InvoiceHoursResponse','ResponseParameter')
getVariableData('input','payload','/tns:TimesheetType/Hours/...')
```

Example 16-5 ***Two getVariableData functions being used to retrieve specific pieces of data from different variables.***

The sequence element

The sequence construct allows you to organize a series of activities so that they are executed in a predefined, sequential order.
WS-BPEL provides numerous activities that can be used to express the workflow logic within the process definition.

```
<sequence>
   <receive>
               ...
   </receive>
   <assign>
               ...
```

```
    </assign>
    <invoke>
                ...
    </invoke>
    <reply>
                ...
    </reply>
</sequence>
```

Example 16-6 ***A skeleton sequence construct containing only some of the many activity elements provided by WS-BPEL.***

The invoke element

The invoke element is equipped with five common attributes which further specify the details of the invocation (Table 16.1).

Attribute	Description
partnerLink	This element names the partner service via its corresponding partnerLink.
portType	The element used to identify the portType element of the partner service.
operation	The partner service operation to which the process service will need to send its request.
inputVariable	The input message that will be used to communicate with the partner service operation. Note that it is referred to as a variable because it is referencing a WS-BPEL variable element with a messageType attribute.
outputVariable	This element is used when communication is based on the request-response MEP. The return value is stored in a separate variable element.

Table 16-1 ***invoke element attributes.***

```
<invoke name="ValidateWeeklyHours"
        partnerLink="Employee"
        portType="emp:EmployeeInterface"
        operation="GetWeeklyHoursLimit"
        inputVariable="EmployeeHoursRequest"
        outputVariable="EmployeeHoursResponse"/>
```

Example 16-7 ***The invoke element identifying the target partner service details.***

The receive element

The receive element allows us to establish the information a process service expects upon receiving a request from an external client partner service.

The receive element contains a set of attributes, each of which is assigned a value relating to the expected incoming communication (Table 16.2).

Attribute	Description
partnerLink	The client partner service identified in the corresponding partnerLink construct.
portType	The partner service’s portType involved in the message transfer.
operation	The partner service’s operation that will be issuing the request to the

	process service.
variable	The process definition variable construct in which the incoming request message will be stored.
createInstance	When this attribute is set to “yes” the receipt of this particular request may be responsible for creating a new instance of the process.

Table 16-2 ***receive element attributes.***

Note that this element can also be used to receive callback messages during an asynchronous message exchange.

```
<receive name="receiveInput"
            partnerLink="client"
            portType="tns:TimesheetSubmissionInterface"
            operation="Submit"
            variable="ClientSubmission"
            createInstance="yes"/>
```

Example 16-8 ***The receive element used in the Timesheet Submission Process definition to indicate the client partner service responsible for launching the process with the submission of a timesheet document.***

The reply element

The reply element is responsible for establishing the details of returning a response message to the requesting client partner service.

Attribute	Description
partnerLink	The same partnerLink element established in the receive element.
portType	The same portType element displayed in the receive element.
operation	The same operation element from the receive element.
variable	The process service variable element that holds the message that is returned to the partner service.
messageExchange	It is being proposed that this optional attribute be added by the WS-BPEL 2.0 specification. It allows for the reply element to be explicitly associated with a message activity capable of receiving a message (such as the receive element).

Table 16-3 ***reply element attributes.***

```
<reply partnerLink="client"
            portType="TimeSubmissionProcessInterface"
    operation="SubmitTimesheet"
    variable="TimesheetSubmissionResponse"/>
```

Example 16-9 ***A potential companion reply element to the previously displayed receive element.***

The switch, case, and otherwise elements

The switch element establishes the scope of the conditional logic
multiple case constructs can be nested to check for various conditions using a condition attribute.
condition attribute resolves to “true,” the activities defined within the corresponding case construct are executed.
The otherwise element can be added as a catch all at the end of the switch construct.
Should all preceding case conditions fail, the activities within the otherwise construct are executed.

```
<switch>
            <case condition="getVariableData(‘EmployeeResponseMessage’,‘ResponseParameter’)=0">
                        ...
```

```
        </case>
        <otherwise>
                ...
        </otherwise>
</switch>
```

Example 16-10 ***A skeleton case element wherein the condition attribute uses the getVariableData function to compare the content of the EmployeeResponseMessage variable to a zero value.***

Note: It has been proposed that the switch, case, and otherwise elements be replaced with if, elseif, and else elements in WS-BPEL 2.0.

The assign, copy, from, and to elements

This set of elements simply gives us the ability to copy values between process variables

```
<assign>
        <copy>
                <from variable="TimesheetSubmissionFailedMessage"/>
      <to variable="EmployeeNotificationMessage"/>
        </copy>
        <copy>
                <from variable="TimesheetSubmissionFailedMessage"/>
      <to variable="ManagerNotificationMessage"/>
        </copy>
</assign>
```

Example 16-11 ***Within this assign construct, the contents of the TimesheetSubmissionFailedMessage variable are copied to two different message variables.***

Note that the copy construct can process a variety of data transfer functions
from and to elements can contain optional part and query attributes that allow for specific parts or values of the variable to be referenced.

faultHandlers, catch, and catchAll elements

This construct can contain multiple catch elements, each of which provides activities that perform exception handling for a specific type of error condition.
Faults can be generated by the receipt of a WSDL-defined fault message, or they can be explicitly triggered through the use of the throw element.
The faultHandlers construct can consist of (or end with) a catchAll element to house default error handling activities.

```
<faultHandlers>
        <catch faultName="SomethingBadHappened"
                faultVariable="TimesheetFault">
                ...
        </catch>
        <catchAll>
                ...
        </catchAll>
</faultHandlers>
```

Example 16-12 ***The faultHandlers construct hosting catch and catchAll child constructs.***

Other WS-BPEL elements

Table 16.4 provides brief descriptions of other relevant parts of the WS-BPEL language.

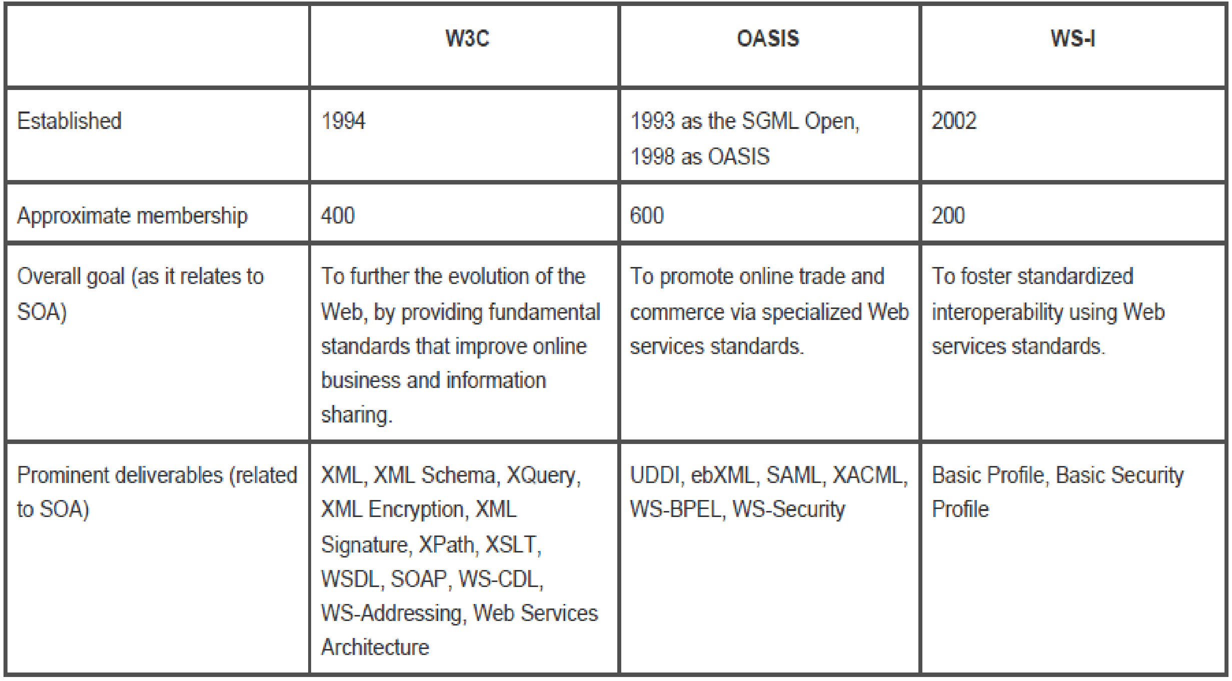
Element	Description
compensationHandler	A WS-BPEL process definition can define a compensation process that kicks in a series of activities when certain conditions occur to justify a compensation. These activities are kept in the compensationHandler construct. (For more information about compensations, see the *Business activities* section in Chapter 6.)
correlationSets	WS-BPEL uses this element to implement correlation, primarily to associate messages with process instances. A message can belong to multiple correlationSets. Further, message properties can be defined within WSDL documents.
empty	This simple element allows you to state that no activity should occur for a particular condition.
eventHandlers	The eventHandlers element enables a process to respond to events during the execution of process logic. This construct can contain onMessage and onAlarm child elements that trigger process activity upon the arrival of specific types of messages (after a predefined period of time, or at a specific date and time, respectively).
exit	See the terminate element description below.
flow	A flow construct allows you to define a series of activities that can occur concurrently and are required to complete after all have finished executing. Dependencies between activities within a flow construct are defined using the child link element.
pick	Similar to the eventHandlers element, this construct can also contain child onMessage and onAlarm elements, but is used more to respond to external events for which process execution is suspended.
scope	Portions of logic within a process definition can be sub-divided into scopes using this construct. This allows you to define variables, faultHandlers, correlationSets, compensationHandler, and eventHandlers elements local to the scope.
terminate	This element effectively destroys the process instance. The WS-BPEL 2.0 specification proposes that this element be renamed to exit.
throw	WS-BPEL supports numerous fault conditions. Using the throw element allows you to explicitly trigger a fault state in response to a specific condition.
wait	The wait element can be set to introduce an intentional delay within the process. Its value can be a set time or a predefined date.
while	This useful element allows you to define a loop. As with the case element, it contains a condition attribute that, as long as it continues resolving to “true”, will continue to execute the activities within the while construct.

Table 16-4 ***Quick reference table providing short descriptions for additional WS-BPEL elements (listed in alphabetical order).***

6.a) List and compare any three standard organization that contribute SOA

- The World Wide Web Consortium – (W3C)
- Organization for the Advancement of Structured Information Standards (OASIS)
- The Web Services Interoperability Organization (WS-I)

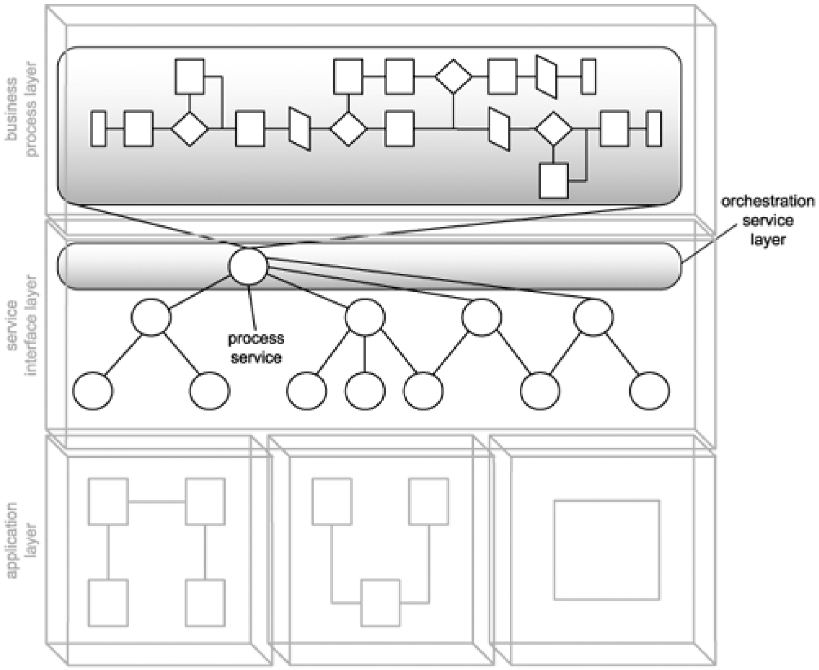
Table 4.1. A Comparison of Standards Organizations



	W3C	OASIS	WS-I
Established	1994	1993 as the SGML Open, 1998 as OASIS	2002
Approximate membership	400	600	200
Overall goal (as it relates to SOA)	To further the evolution of the Web, by providing fundamental standards that improve online business and information sharing.	To promote online trade and commerce via specialized Web services standards.	To foster standardized interoperability using Web services standards.
Prominent deliverables (related to SOA)	XML, XML Schema, XQuery, XML Encryption, XML Signature, XPath, XSLT, WSDL, SOAP, WS-CDL, WS-Addressing, Web Services Architecture	UDDI, ebXML, SAML, XACML, WS-BPEL, WS-Security	Basic Profile, Basic Security Profile

6.b) Write short note about Orchestration Service Layer

- It allows us to directly link process logic to service interaction within our workflow logic
- Orchestration brings the business process into the service layer, positioning it as a master composition controller.
- The orchestration service layer introduces a parent level of abstraction to ensure that service operations are executed in a specific sequence.
- This promotes agility and reusability
- Within the orchestration service layer, *process services* compose other services that provide specific sets of functions, independent of the business rules and scenario-specific logic required to execute a process instance.



7. Summarize the steps to design service oriented business process

Step 1: Map out interaction scenarios.

By using the following information gathered so far, we can define the message exchange requirements of our process service:

- Available workflow logic produced during the service modeling process in Chapter 12.
- The process service candidate created in Chapter 12.
- The existing service designs created in Chapter 15.

This information is now used to form the basis of an analysis during which all possible interaction scenarios between process and partner services are mapped out. The result is a series of processing requirements that will form the basis of the process service design we proceed to in Step 2.

The result of mapping out interaction scenarios establishes that the process service has one potential client partner service, and four potential partner services from which it may need to invoke up to five operations (Figure 16.10).

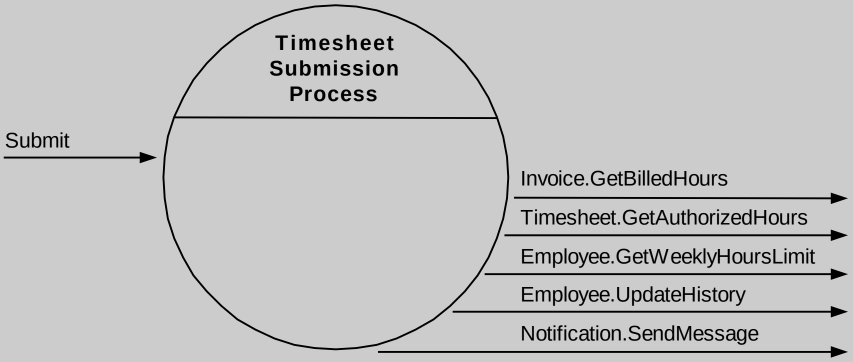

Timesheet Submission Process
Submit
Invoice.GetBilledHours
Timesheet.GetAuthorizedHours
Employee.GetWeeklyHoursLimit
Employee.UpdateHistory
Notification.SendMessage

Figure 16-2 ***The incoming and outgoing request messages expected to be processed by the Timesheet Submission Process Service.***

Step 2: Design the process service interface.

Now that we understand the message exchange requirements, we can proceed to define a service definition for the process service. When working with process modeling tools, the process service WSDL will typically be auto-generated for you. However, you should also be able to edit the source markup code or even import your own WSDL.

Either way, it is best to review the WSDL being used and revise it as necessary. Here are some suggestions:

- Document the input and output values required for the processing of each operation, and populate the `types` section with XSD schema types required to process the operations. Move the XSD schema information to a separate file, if required.
- Build the WSDL definition by creating the `portType` (or `interface`) area, inserting the identified `operation` constructs. Then, add the necessary `message` constructs containing the `part` elements which reference the appropriate schema types. Add naming conventions that are in alignment with those used by your other WSDL definitions.
- Add meta information via the `documentation` element.
- Apply other design standards within the confines of the modeling tool.

There is less opportunity to incorporate the other steps from the service design processes described in Chapter 15. For example, applying the service-orientation principle of statelessness is difficult, since process services maintain state so that other services don’t have to.

Example

It looks like the Timesheet Submission Process Service interface will be pretty straight-forward. It only requires one operation used by a client to initiate the process instance (Figure 16.11).

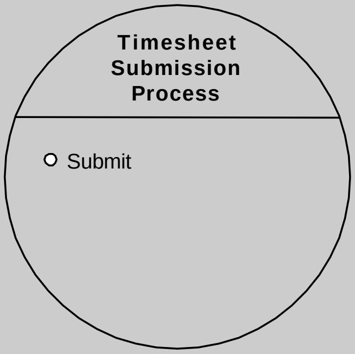


Figure 16-3 ***Timesheet Submission Process Service design.***

Below is the corresponding WSDL definition.

```
<definitions name="TimesheetSubmission"
     targetNamespace="http://www.xmltc.com/tls/process/wsdl/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:ts="http://www.xmltc.com/tls/timesheet/schema/"
```

```
      xmlns:tsd="http://www.xmltc.com/tls/timesheetservice/schema/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:tns="http://www.xmltc.com/tls/timesheet/wsdl/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
<types>
      <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.xmltc.com/tls/timesheetsubmissionservice
/schema/">
            <xsd:import
namespace="http://www.xmltc.com/tls/timesheet/schema/"
schemaLocation="Timesheet.xsd"/>
            <xsd:element name="Submit">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="ContextID" type="xsd:integer"/>
                        <xsd:element name="TimesheetDocument"
type="ts:TimesheetType"/>
                  </xsd:sequence>
            </xsd:complexType>
            </xsd:element>
      </xsd:schema>
</types>
<message name="receiveSubmitMessage">
      <part name="Payload" element="tsd:TimesheetType"/>
</message>
<portType name="TimesheetSubmissionInterface">
      <documentation>
            Initiates the Timesheet Submission process. </documentation>
      <operation name="Submit">
            <input message="tns:receiveSubmitMessage"/>
      </operation>
</portType>
<plnk:partnerLinkType name="TimesheetSubmissionType">
      <plnk:role name="TimesheetSubmissionService">
            <plnk:portType name="tns:TimesheetSubmissionInterface"/>
      </plnk:role>
</plnk:partnerLinkType>
```

```
</definitions>
```

Example 16-13 ***The abstract service definition for the Timesheet Submission Process Service.***

Note the bolded `plnk:parnterLinkType` construct at the end of this WSDL definition. This is added to every partner service.

Step 3: Formalize partner service conversations.

We now begin our WS-BPEL process definition by establishing details about the services with which our process service will be interacting.
The following steps are suggested:

1. Define the partner services that will be participating in the process and assign each the role it will be playing within a given message exchange.
2. Add `parterLinkType` constructs to the end of the WSDL definitions of each partner service.
3. Create `partnerLink` elements for each partner service within the process definition.
4. Define `variable` elements to represent incoming and outgoing messages exchanged with partner services.

This information essentially documents the possible conversation flows that can occur within the course of the process execution. Depending on the process modeling tool used, completing these steps may simply require interaction with the user-interface provided by the modeling tool.

Example

Now that the Timesheet Submission Process Service has an interface, TLS can begin to work on the corresponding process definition. It begins by looking at the information it gathered in Step 1. As you may recall, TLS determined the process service as having one potential client partner service, and four potential partner services from which it may need to invoke up to five operations.

Roles are assigned to each of these services, labeled according to how they relate to the process service. These roles are then formally defined by appending existing service WSDL definitions with a `partnerLinkType` construct.

The example below shows how the Employee Service definition (as designed in Chapter 15) is amended to incorporate the WS-BPEL `partnerLinkType` construct and its corresponding namespace.

```
<definitions
      name="Employee"
      targetNamespace="http://www.xmltc.com/tls/employee/wsdl/"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:act="http://www.xmltc.com/tls/employee/schema/accounting/"
      xmlns:hr="http://www.xmltc.com/tls/employee/schema/hr/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:tns="http://www.xmltc.com/tls/employee/wsdl/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
...
```

```
     <plnk:partnerLinkType name="EmployeeType">
          <plnk:role name="EmployeeService">
               <plnk:portType name="tns:EmployeeInterface"/>
          </plnk:role>
     </plnk:partnerLinkType>
</definitions>
```

Example 16-14 ***The request messages expected to be processed by the Timesheet Submission Process Service.***

This is formalized within the process definition through the creation of `partnerLink` elements that reside within the `partnerLinks` construct. TLS analysts and architects work with a process modeling tool to drag and drop `partnerLink` objects, resulting in the following code being generated.

```
<partnerLinks>
     <partnerLink name="client"
          partnerLinkType="bpl:TimesheetSubmissionProcessType"
          myRole="TimesheetSubmissionProcessServiceProvider"/>
     <partnerLink name="Invoice"
          partnerLinkType="inv:InvoiceType"
          partnerRole="InvoiceServiceProvider"/>
      <partnerLink name="Timesheet"
          partnerLinkType="tst:TimesheetType"
          partnerRole="TimesheetServiceProvider"/>
      <partnerLink name="Employee"
          partnerLinkType="emp:EmployeeType"
          partnerRole="EmployeeServiceProvider"/>
      <partnerLink name="Notification"
          partnerLinkType="not:NotificationType"
          partnerRole="NotificationServiceProvider"/>
</partnerLinks>
```

Example 16-15 ***The `partnerLinks` construct containing `partnerLink` elements for each of the process partner services.***

Next the input and output messages of each partner service are assigned to individual `variable` elements, as part of the `variables` construct. A `variable` element is also added to represent the Timesheet Submission Process Service Submit operation that is called by the HR client application to kick off the process.

```
<variables>
      <variable name="ClientSubmission"
messageType="bpl:receiveSubmitMessage"/>
```

```
      <variable name="EmployeeHoursRequest"
messageType="emp:getWeeklyHoursRequestMessage"/>

      <variable name="EmployeeHoursResponse"
messageType="emp:getWeeklyHoursResponseMessage"/>

      <variable name="EmployeeHistoryRequest"
messageType="emp:updateHistoryRequestMessage"/>

      <variable name="EmployeeHistoryResponse"
messageType="emp:updateHistoryResponseMessage"/>

      <variable name="InvoiceHoursRequest"
messageType="inv:getBilledHoursRequestMessage"/>

      <variable name="InvoiceHoursResponse"
messageType="inv:getBilledHoursResponseMessage"/>

      <variable name="TimesheetAuthorizationRequest"
messageType="tst:getAuthorizedHoursRequestMessage"/>

      <variable name="TimesheetAuthorizationResponse"
messageType="tst:getAuthorizedHoursResponseMessage"/>

      <variable name="NotificationRequest" messageType="not:sendMessage"/>

</variables>
```

Example 16-16 ***The `variables` construct containing individual `variable` elements representing input and output messages from all partner services and for the process service itself.***

If you check back to the Employee Service definition TLS designed in Chapter 15, you'll notice that the `name` values of the `message` elements correspond to the values assigned to the `messageType` attributes in the above displayed `variable` elements.

Step 4: Define process logic.

Finally, everything is in place for us to complete the process definition. This step is a process in itself, as it requires that all existing workflow intelligence be transposed and implemented via a WS-BPEL process definition.

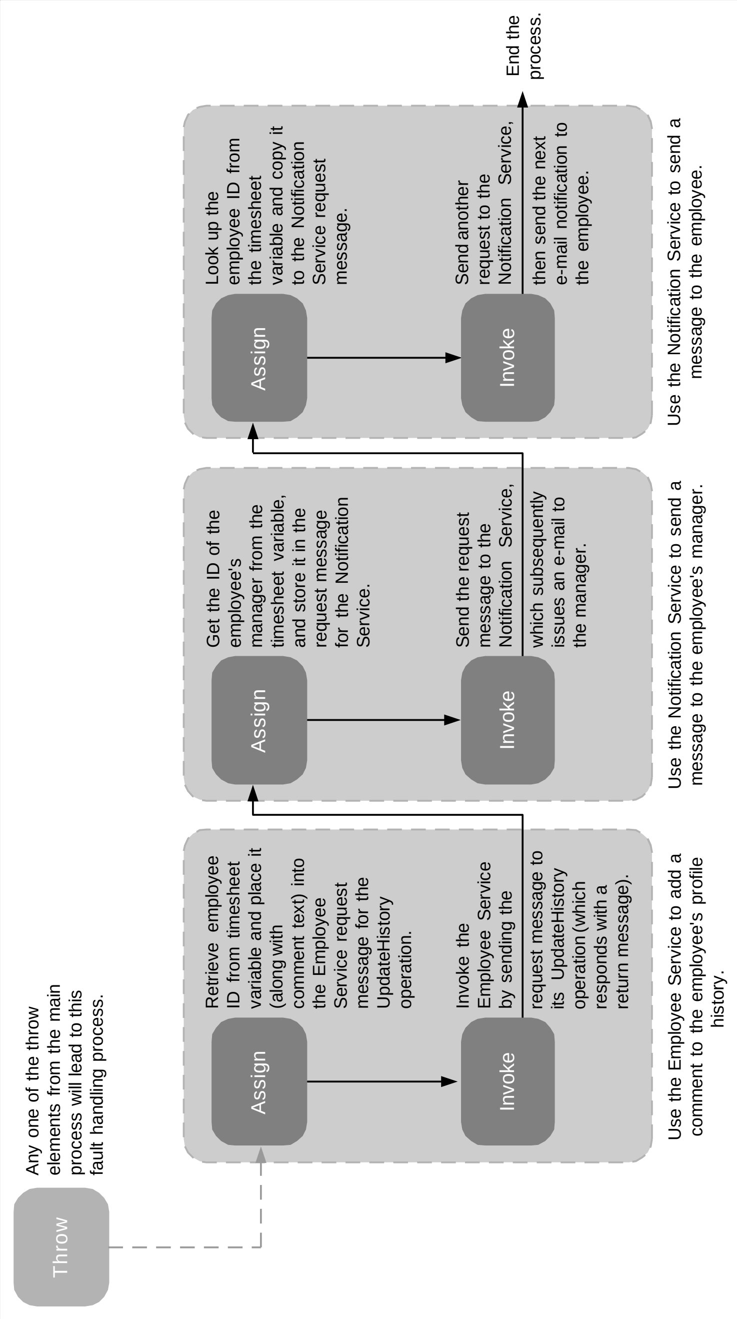

Throw
Any one of the throw elements from the main process will lead to this fault handling process.
Assign
Retrieve employee ID from timesheet variable and place it (along with comment text) into the Employee Service request message for the UpdateHistory operation.
Invoke
Invoke the Employee Service by sending the request message to its UpdateHistory operation (which responds with a return message).
Use the Employee Service to add a comment to the employee's profile history.
Assign
Get the ID of the employee's manager from the timesheet variable, and store it in the request message for the Notification Service.
Invoke
Send the request message to the Notification Service, which subsequently issues an e-mail to the manager.
Use the Notification Service to send a message to the employee's manager.
Assign
Look up the employee ID from the timesheet variable and copy it to the Notification Service request message.
Invoke
Send another request to the Notification Service, then send the next e-mail notification to the employee.
Use the Notification Service to send a message to the employee.
End the process.

Figure 16-4 ***A visual representation of the process logic within the `faultHandlers` construct.***

Note that the following, abbreviated markup code samples reside within the `sequence` child construct of the parent `faultHandlers` construct established in the previous example.

First up is the markup code for the “Update employee profile history” task.

```
<assign name="SetEmployeeMessage">
     <copy>
          <from variable="ClientSubmission" .../>
          <to variable="EmployeeHistoryRequest" .../>
```

```
      </copy>
      <copy>
            <from expression="..."/>
            <to variable="EmployeeHistoryRequest" .../>
      </copy>
</assign>
<invoke name="UpdateHistory"
      partnerLink="Employee"
      portType="emp:EmployeeInterface"
      operation="UpdateHistory"
      inputVariable="EmployeeHistoryRequest"
      outputVariable="EmployeeHistoryResponse"/>
```

Example 16-17 ***Two `copy` elements used to populate the EmployeeHistoryRequest message.***

To perform the first task of updating the employee history, the fault handler routine uses an `assign` construct with two `copy` constructs. The first retrieves the EmployeeID value from the ClientSubmission variable, while the latter adds a static employee profile history comment.

The `invoke` element then launches the Employee Service (used previously for its GetWeeklyHoursLimit operation), and submits the EmployeeHistoryRequest message to its UpdateHistory operation in order to log the profile history comment.

The next block of markup code takes care of both the remaining “Send notification” tasks.

```
<assign name="GetManagerID">
      <copy>
            <from expression="getVariableData(...)"/>
            <to variable="NotificationRequest" .../>
      </copy>
</assign>
<invoke name="SendNotification"
      partnerLink="Notification"
      portType="not:NotificationInterface"
      operation="SendMessage"
      inputVariable="NotificationRequest"/>
<assign name="GetEmployeeID">
      <copy>
            <from expression="getVariableData(...)"/>
            <to variable="NotificationRequest" .../>              </copy>
</assign>
```

```
<invoke name="SendNotification"
     partnerLink="Notification"
     portType="not:NotificationInterface"
     operation="SendMessage"
     inputVariable="NotificationRequest"/>
<terminate name="EndTimesheetSubmissionProcess"/>
```

Example 16-18 ***The last activities in the process.***

Finally, the `faultHandlers` construct contains two more `assign` + `invoke` element pairs. Both use the Notification Service's SendMessage operation, but in different ways. The first `assign` construct extracts the ManagerID value from the ClientSubmission variable, which is then passed to the Notification Service. It is the sole parameter that the service subsequently uses to look up the corresponding e-mail address used to send the notification message.

Next, the second `assign` construct retrieves the EmployeeID value from the same ClientSubmission variable, which the Notification Service ends up using to send a message to the employee.

`terminate`, the very last element in the construct, halts all further processing.

Step 5: Align interaction scenarios and refine process. (Optional)

This final, optional step encourages you to perform two specific tasks: revisit the original interaction scenarios created in Step 1 and review the WS-BPEL process definition to look for optimization opportunities.

Let's start with the first task. Bringing the interaction scenarios in alignment with the process logic expressed in the WS-BPEL process definition provides a number of benefits, including:

- The service interaction maps (as activity diagrams or in whatever format you created them) are an important part of the solution documentation, and will be useful for future maintenance and knowledge transfer requirements.
- The service interaction maps make for great test cases, and can spare testers from having to perform speculative analysis.
- The implementation of the original workflow logic as a series of WS-BPEL activities may have introduced new or augmented process logic. Once compared to the existing interaction scenarios, the need for additional service interactions may arise, leading to the discovery of new fault or exception conditions that can then be addressed back in the WS-BPEL process definition.

Secondly, spending some extra time to review your WS-BPEL process definition is well worth the effort. WS-BPEL is a multi-feature language that provides different approaches for accomplishing and structuring the same overall activities. By refining your process definition, you may be able to:

- Consolidate or restructure activities to achieve performance improvements.
- Streamline the markup code to make maintenance easier.
- Discover features that were previously not considered.

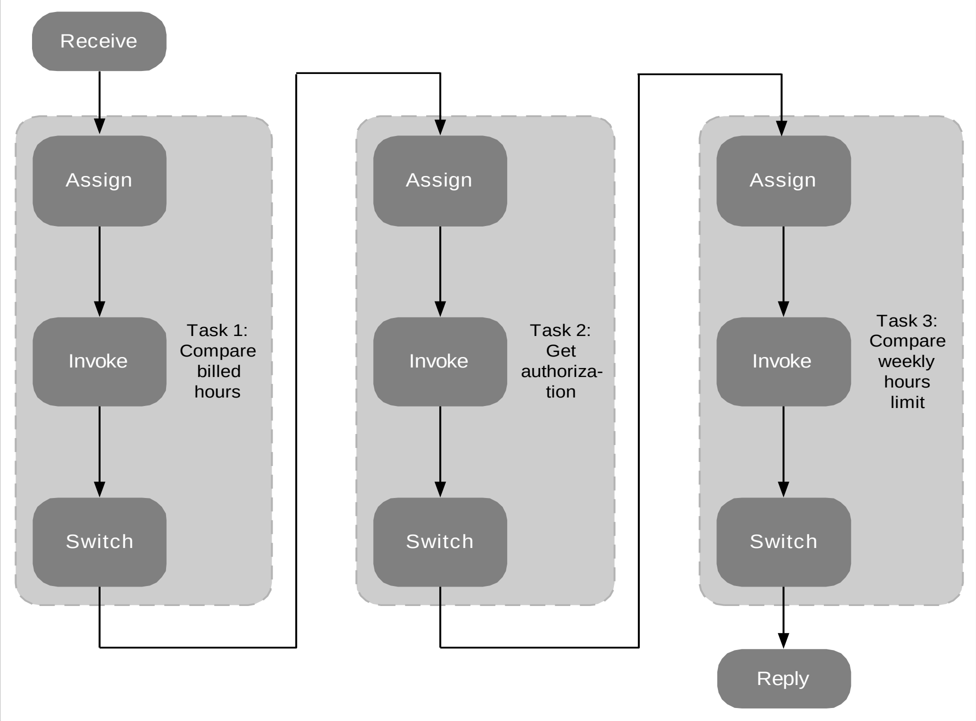

Receive
Assign
Invoke
Task 1: Compare billed hours
Switch
Assign
Invoke
Task 2: Get authorization
Switch
Assign
Invoke
Task 3: Compare weekly hours limit
Switch
Reply

Figure 16-5 ***Sequential, synchronous execution of process activities.***

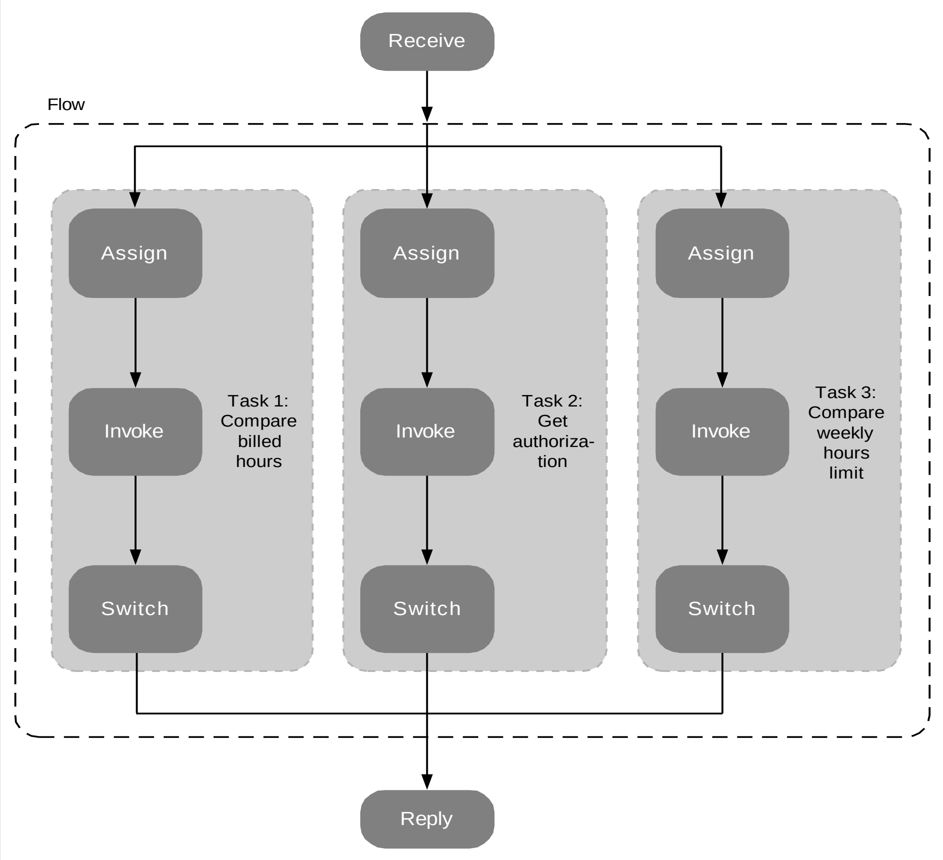

Receive
Flow
Assign
Invoke
Task 1: Compare billed hours
Switch
Assign
Invoke
Task 2: Get authorization
Switch
Assign
Invoke
Task 3: Compare weekly hours limit
Switch
Reply

Figure 16-6 ***Concurrent execution of process activities using the `Flow` construct.***

Finally, while reviewing the structure of the fault handling routine, a further refinement is suggested. Because the last two activities invoke the same Notification Service, they can be collapsed into a `while` construct that loops twice through the `invoke` element.

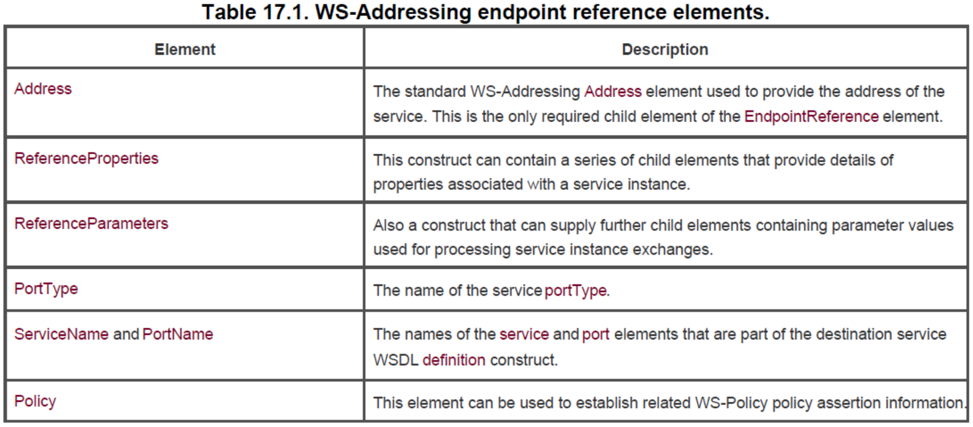
8. Explain WS-Addressing language basics

17.1.1. The EndpointReference element

The EndpointReference element is used by the From, ReplyTo, and FaultTo elements described in the *Message information header elements* section. This construct can be comprised of a set of elements that assist in providing service interface information (including supplementary metadata), as well as the identification of service instances.

The WS-Addressing elements described in Table 17.1 can be associated with an EndpointReference construct.

Table 17.1. WS-Addressing endpoint reference elements.

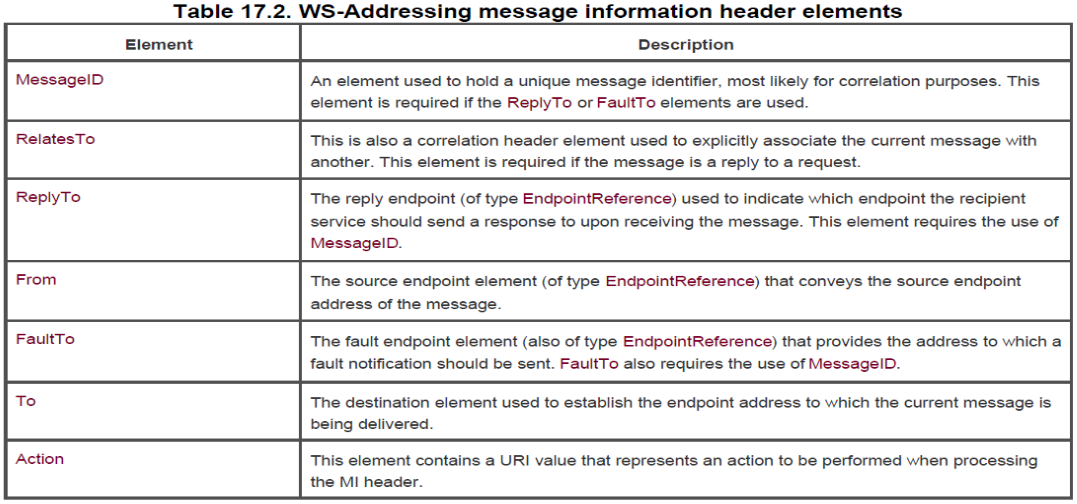


Element	Description
Address	The standard WS-Addressing Address element used to provide the address of the service. This is the only required child element of the EndpointReference element.
ReferenceProperties	This construct can contain a series of child elements that provide details of properties associated with a service instance.
ReferenceParameters	Also a construct that can supply further child elements containing parameter values used for processing service instance exchanges.
PortType	The name of the service portType.
ServiceName and PortName	The names of the service and port elements that are part of the destination service WSDL definition construct.
Policy	This element can be used to establish related WS-Policy policy assertion information.

17.1.2. Message information header elements

This collection of elements (first introduced as concepts in Chapter 7) can be used in various ways to assemble metadata-rich SOAP header blocks. Table 17.2 lists the primary elements and provides brief descriptions.

Table 17.2. WS-Addressing message information header elements



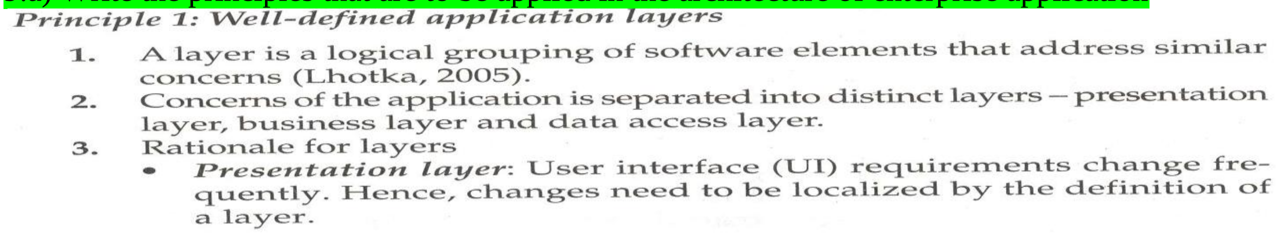
Element	Description
MessageID	An element used to hold a unique message identifier, most likely for correlation purposes. This element is required if the ReplyTo or FaultTo elements are used.
RelatesTo	This is also a correlation header element used to explicitly associate the current message with another. This element is required if the message is a reply to a request.
ReplyTo	The reply endpoint (of type EndpointReference) used to indicate which endpoint the recipient service should send a response to upon receiving the message. This element requires the use of MessageID.
From	The source endpoint element (of type EndpointReference) that conveys the source endpoint address of the message.
FaultTo	The fault endpoint element (also of type EndpointReference) that provides the address to which a fault notification should be sent. FaultTo also requires the use of MessageID.
To	The destination element used to establish the endpoint address to which the current message is being delivered.
Action	This element contains a URI value that represents an action to be performed when processing the MI header.

17.1.3. WS-Addressing reusability

The endpoint identification and message routing mechanisms provided by WS-Addressing establish a generic set of extensions useful to custom service-oriented solutions but also reusable by other WS-* specifications. As such, WS-Addressing can be viewed as a utility specification that further supports the notion of composability within SOA.

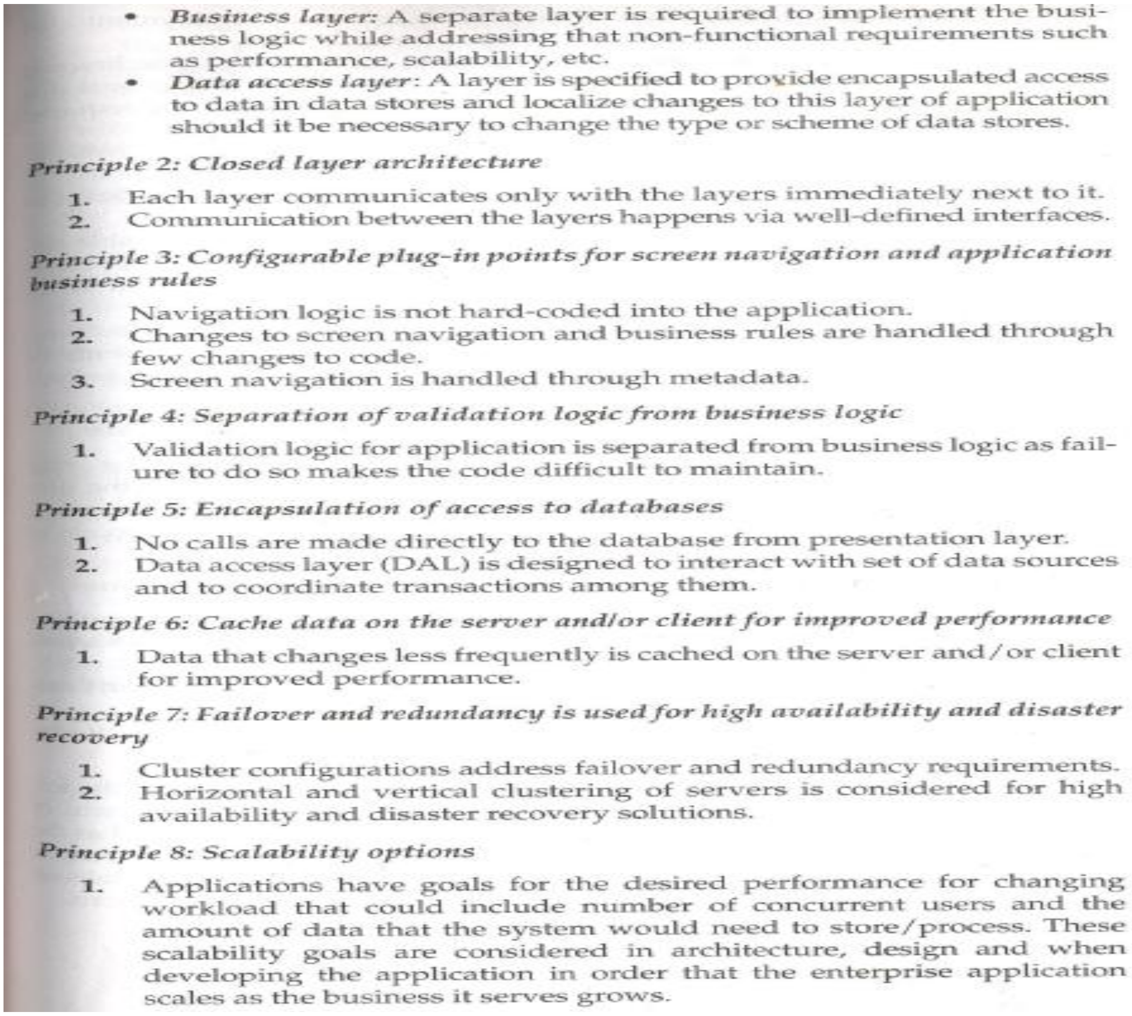
Although we don't discuss the WS-Notification or WS-Eventing languages in any detail, let's take a brief glimpse at their Header constructs for some examples of how WS-Addressing message information header elements are reused in support of other WS-* extensions.

9.a) Write the principles that are to be applied in the architecture of enterprise application



Principle 1: Well-defined application layers

1. A layer is a logical grouping of software elements that address similar concerns (Lhotka, 2005).
2. Concerns of the application is separated into distinct layers – presentation layer, business layer and data access layer.
3. Rationale for layers
   - ***Presentation layer***: User interface (UI) requirements change frequently. Hence, changes need to be localized by the definition of a layer.



   - ***Business layer:*** A separate layer is required to implement the business logic while addressing that non-functional requirements such as performance, scalability, etc.
   - ***Data access layer***: A layer is specified to provide encapsulated access to data in data stores and localize changes to this layer of application should it be necessary to change the type or scheme of data stores.

Principle 2: Closed layer architecture

1. Each layer communicates only with the layers immediately next to it.
2. Communication between the layers happens via well-defined interfaces.

Principle 3: Configurable plug-in points for screen navigation and application business rules

1. Navigation logic is not hard-coded into the application.
2. Changes to screen navigation and business rules are handled through few changes to code.
3. Screen navigation is handled through metadata.

Principle 4: Separation of validation logic from business logic

1. Validation logic for application is separated from business logic as failure to do so makes the code difficult to maintain.

Principle 5: Encapsulation of access to databases

1. No calls are made directly to the database from presentation layer.
2. Data access layer (DAL) is designed to interact with set of data sources and to coordinate transactions among them.

Principle 6: Cache data on the server and/or client for improved performance

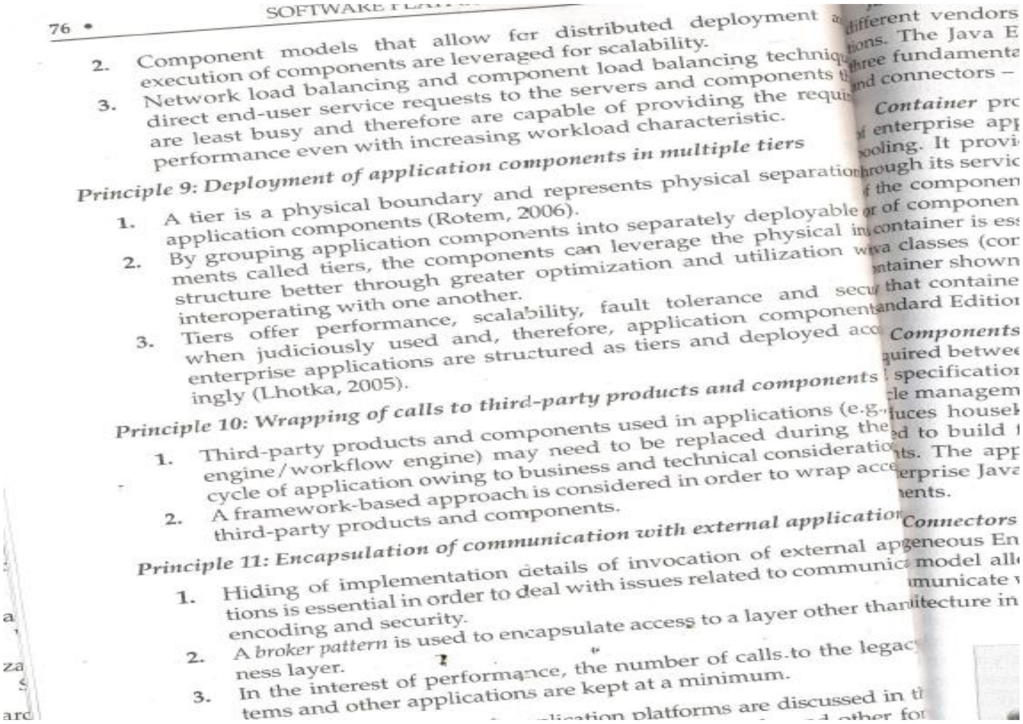
1. Data that changes less frequently is cached on the server and / or client for improved performance.

Principle 7: Failover and redundancy is used for high availability and disaster recovery

1. Cluster configurations address failover and redundancy requirements.
2. Horizontal and vertical clustering of servers is considered for high availability and disaster recovery solutions.

Principle 8: Scalability options

1. Applications have goals for the desired performance for changing workload that could include number of concurrent users and the amount of data that the system would need to store/process. These scalability goals are considered in architecture, design and when developing the application in order that the enterprise application scales as the business it serves grows.



2. Component models that allow for distributed deployment a
execution of components are leveraged for scalability.
3. Network load balancing and component load balancing techniq
direct end-user service requests to the servers and components th
are least busy and therefore are capable of providing the requi
performance even with increasing workload characteristic.

Principle 9: Deployment of application components in multiple tiers

1. A tier is a physical boundary and represents physical separation
application components (Rotem, 2006).
2. By grouping application components into separately deployable
ments called tiers, the components can leverage the physical in
structure better through greater optimization and utilization w
interoperating with one another.
3. Tiers offer performance, scalability, fault tolerance and secu
when judiciously used and, therefore, application component
enterprise applications are structured as tiers and deployed acc
ingly (Lhotka, 2005).

Principle 10: Wrapping of calls to third-party products and components

1. Third-party products and components used in applications (e.g.
engine/workflow engine) may need to be replaced during the
cycle of application owing to business and technical consideratio
2. A framework-based approach is considered in order to wrap acce
third-party products and components.

Principle 11: Encapsulation of communication with external applicatio

1. Hiding of implementation details of invocation of external ap
tions is essential in order to deal with issues related to communic
encoding and security.
2. A *broker pattern* is used to encapsulate access to a layer other than
ness layer.
3. In the interest of performance, the number of calls to the legac
tems and other applications are kept at a minimum.

plication platforms are discussed in th
d other for

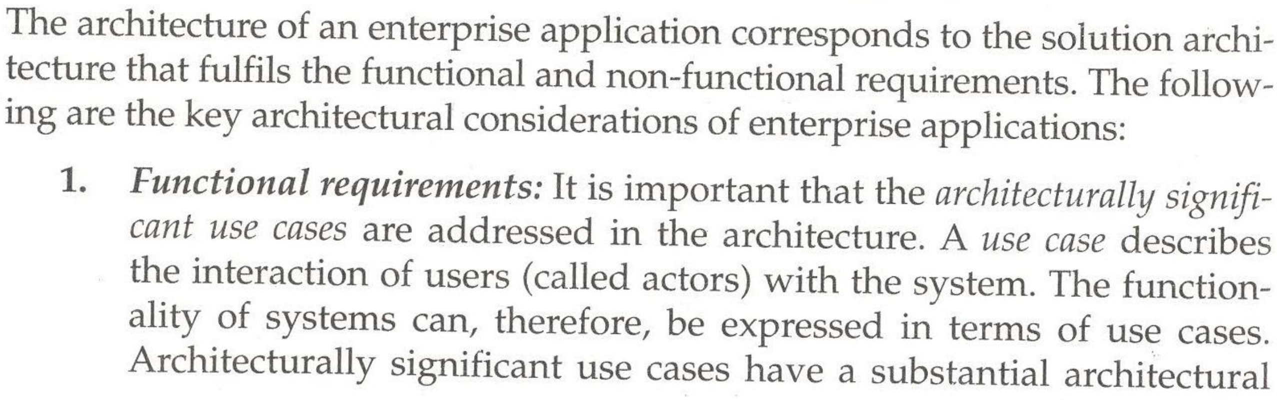
different vendors
tions. The Java E
three fundamenta
and connectors –

Container pro
of enterprise app
pooling. It provi
through its servic
f the componen
or of componen
container is ess
va classes (cor
ntainer shown
that containe
tandard Edition

Components
quired betwee
specification
cle managem
duces housek
ed to build
nts. The app
erprise Java
nents.

Connectors
geneous En
model all
nmunicate
itecture in

9.b List down the architectural consideration of enterprise application

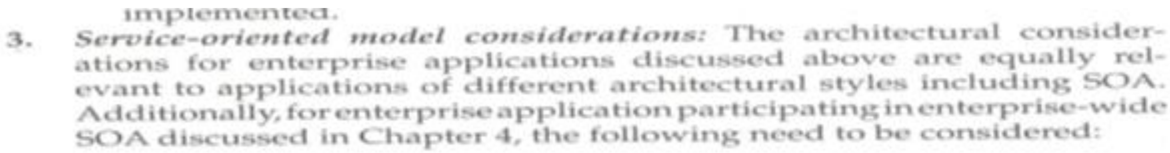


The architecture of an enterprise application corresponds to the solution architecture that fulfils the functional and non-functional requirements. The following are the key architectural considerations of enterprise applications:

1. ***Functional requirements:*** It is important that the *architecturally significant use cases* are addressed in the architecture. A *use case* describes the interaction of users (called actors) with the system. The functionality of systems can, therefore, be expressed in terms of use cases. Architecturally significant use cases have a substantial architectural

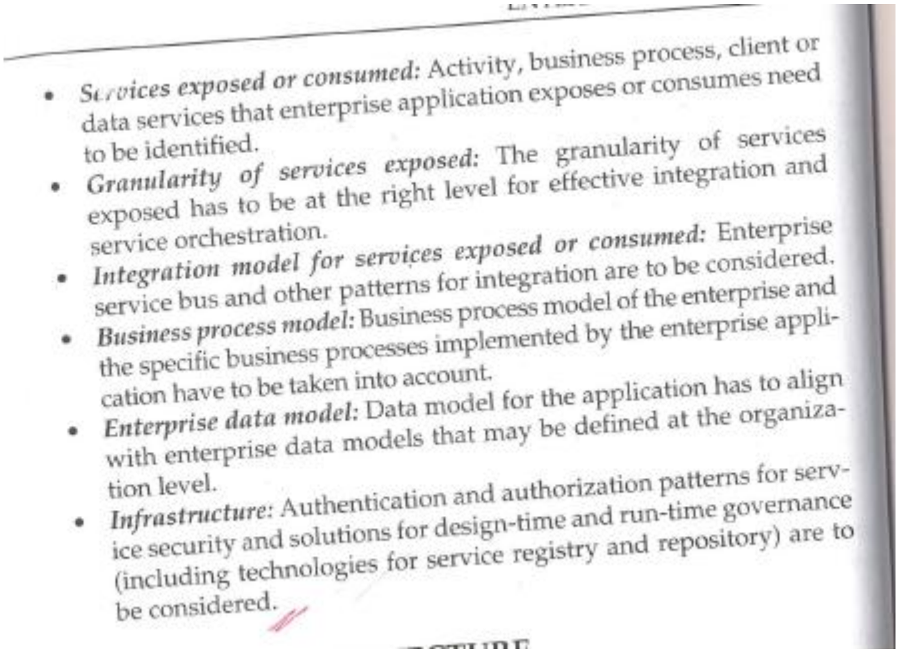
2. Non functional requirements

- Performance
- Scalability
- Availability
- Reliability
- Security



implemented.

3. ***Service-oriented model considerations:*** The architectural considerations for enterprise applications discussed above are equally relevant to applications of different architectural styles including SOA. Additionally, for enterprise application participating in enterprise-wide SOA discussed in Chapter 4, the following need to be considered:



- *Services exposed or consumed:* Activity, business process, client or data services that enterprise application exposes or consumes need to be identified.
- *Granularity of services exposed:* The granularity of services exposed has to be at the right level for effective integration and service orchestration.
- *Integration model for services exposed or consumed:* Enterprise service bus and other patterns for integration are to be considered.
- *Business process model:* Business process model of the enterprise and the specific business processes implemented by the enterprise application have to be taken into account.
- *Enterprise data model:* Data model for the application has to align with enterprise data models that may be defined at the organization level.
- *Infrastructure:* Authentication and authorization patterns for service security and solutions for design-time and run-time governance (including technologies for service registry and repository) are to be considered.

10. Explain the key patterns for J2EE Reference. model and Technical architecture

Reference Model

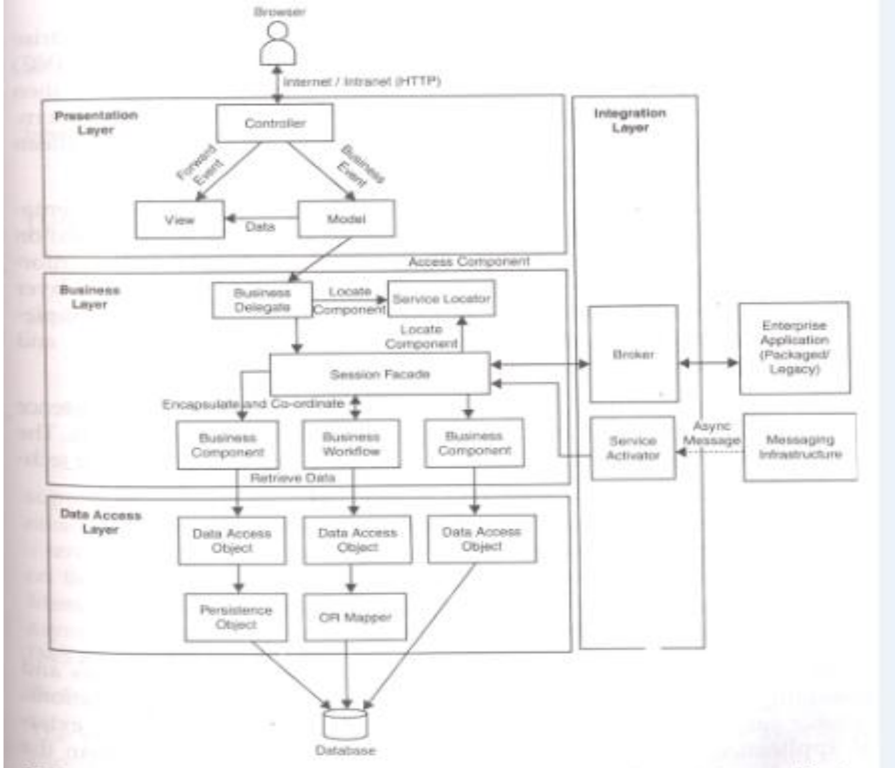
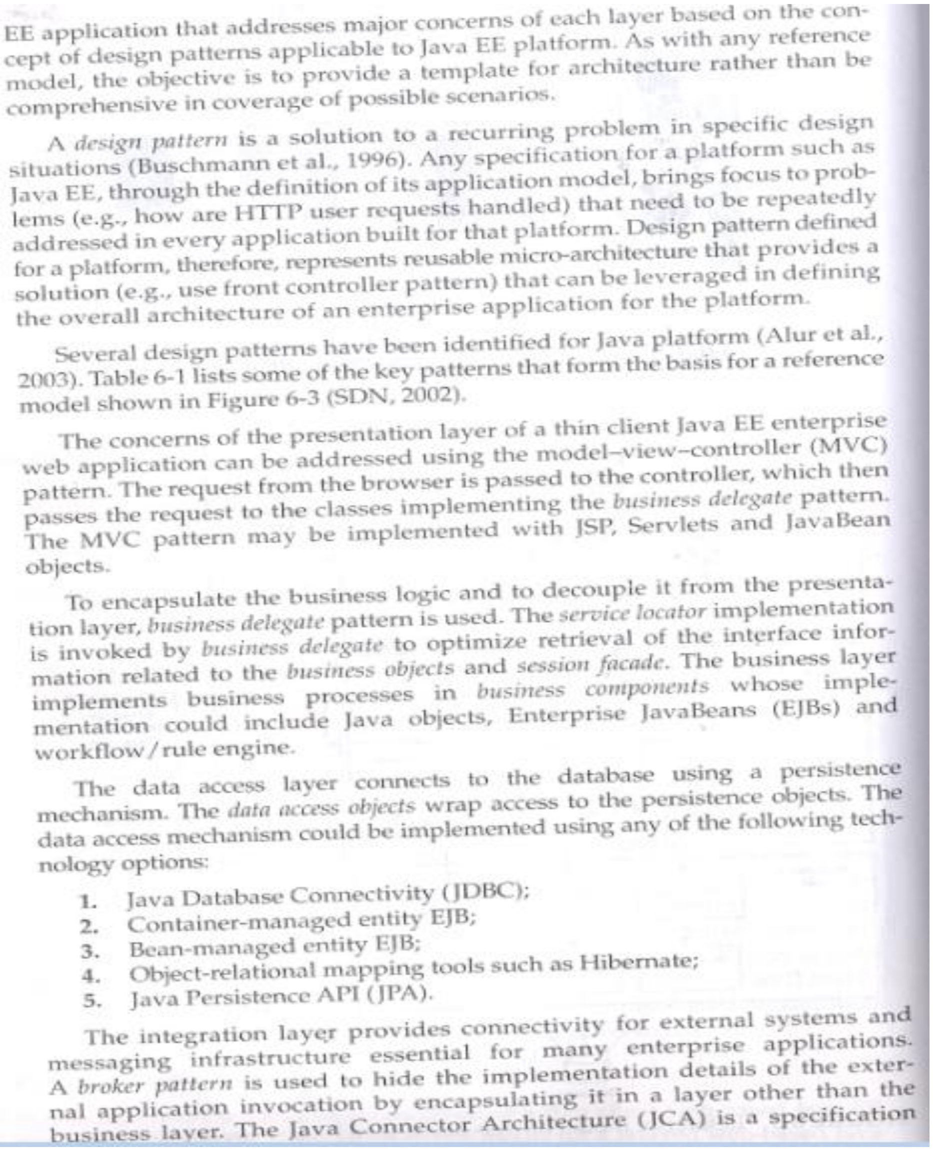


FIGURE 6-3 Reference model of web-based Java EE enterprise application

The below picture shows logical organization of a thin client web-based Java EE application as layers. It depicts reference model for a layered web-based java



EE application that addresses major concerns of each layer based on the concept of design patterns applicable to Java EE platform. As with any reference model, the objective is to provide a template for architecture rather than be comprehensive in coverage of possible scenarios.

A *design pattern* is a solution to a recurring problem in specific design situations (Buschmann et al., 1996). Any specification for a platform such as Java EE, through the definition of its application model, brings focus to problems (e.g., how are HTTP user requests handled) that need to be repeatedly addressed in every application built for that platform. Design pattern defined for a platform, therefore, represents reusable micro-architecture that provides a solution (e.g., use front controller pattern) that can be leveraged in defining the overall architecture of an enterprise application for the platform.

Several design patterns have been identified for Java platform (Alur et al., 2003). Table 6-1 lists some of the key patterns that form the basis for a reference model shown in Figure 6-3 (SDN, 2002).

The concerns of the presentation layer of a thin client Java EE enterprise web application can be addressed using the model–view–controller (MVC) pattern. The request from the browser is passed to the controller, which then passes the request to the classes implementing the *business delegate* pattern. The MVC pattern may be implemented with JSP, Servlets and JavaBean objects.

To encapsulate the business logic and to decouple it from the presentation layer, *business delegate* pattern is used. The *service locator* implementation is invoked by *business delegate* to optimize retrieval of the interface information related to the *business objects* and *session facade*. The business layer implements business processes in *business components* whose implementation could include Java objects, Enterprise JavaBeans (EJBs) and workflow/rule engine.

The data access layer connects to the database using a persistence mechanism. The *data access objects* wrap access to the persistence objects. The data access mechanism could be implemented using any of the following technology options:

1. Java Database Connectivity (JDBC);
2. Container-managed entity EJB;
3. Bean-managed entity EJB;
4. Object-relational mapping tools such as Hibernate;
5. Java Persistence API (JPA).

The integration layer provides connectivity for external systems and messaging infrastructure essential for many enterprise applications. A *broker pattern* is used to hide the implementation details of the external application invocation by encapsulating it in a layer other than the business layer. The Java Connector Architecture (JCA) is a specification

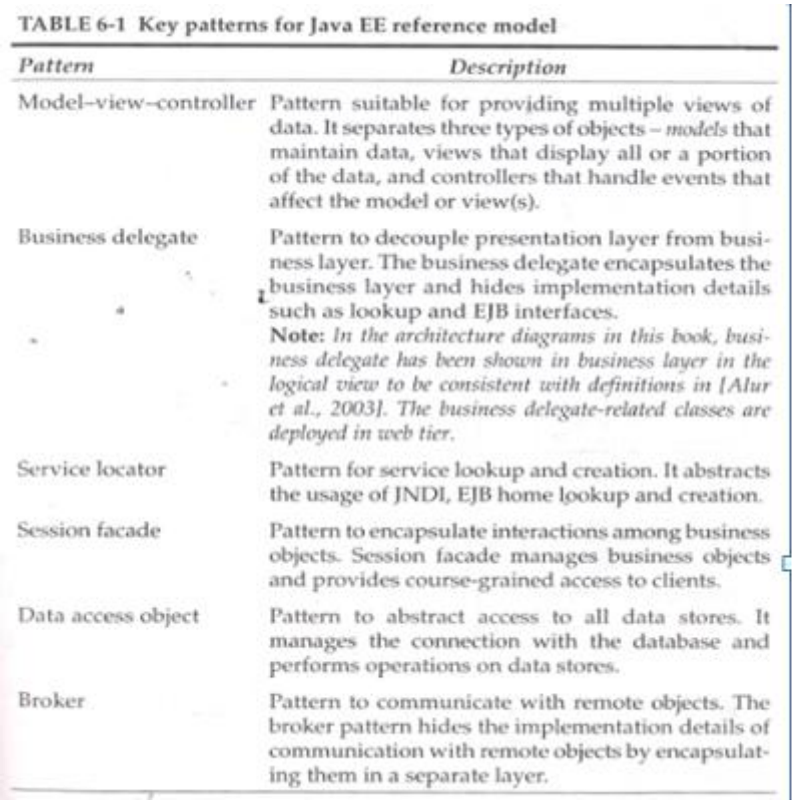
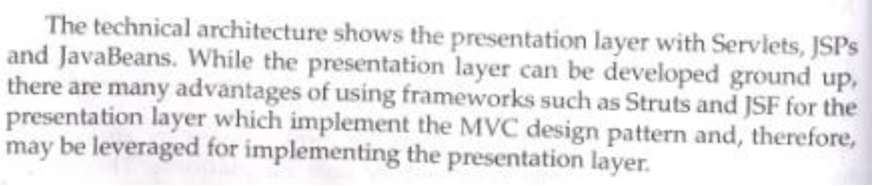


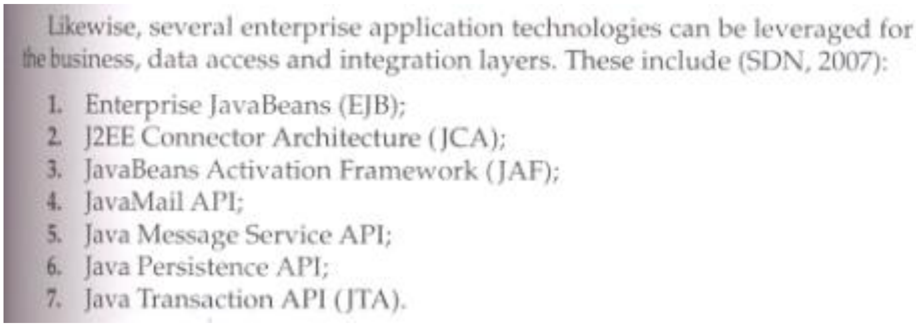
TABLE 6-1 Key patterns for Java EE reference model

Pattern	*Description*
Model–view–controller	Pattern suitable for providing multiple views of data. It separates three types of objects – *models* that maintain data, views that display all or a portion of the data, and controllers that handle events that affect the model or view(s).
Business delegate	Pattern to decouple presentation layer from business layer. The business delegate encapsulates the business layer and hides implementation details such as lookup and EJB interfaces. **Note:** *In the architecture diagrams in this book, business delegate has been shown in business layer in the logical view to be consistent with definitions in [Alur et al., 2003]. The business delegate-related classes are deployed in web tier.*
Service locator	Pattern for service lookup and creation. It abstracts the usage of JNDI, EJB home lookup and creation.
Session facade	Pattern to encapsulate interactions among business objects. Session facade manages business objects and provides course-grained access to clients.
Data access object	Pattern to abstract access to all data stores. It manages the connection with the database and performs operations on data stores.
Broker	Pattern to communicate with remote objects. The broker pattern hides the implementation details of communication with remote objects by encapsulating them in a separate layer.

Technical architecture



The technical architecture shows the presentation layer with Servlets, JSPs and JavaBeans. While the presentation layer can be developed ground up, there are many advantages of using frameworks such as Struts and JSF for the presentation layer which implement the MVC design pattern and, therefore, may be leveraged for implementing the presentation layer.



Likewise, several enterprise application technologies can be leveraged for the business, data access and integration layers. These include (SDN, 2007):

1. Enterprise JavaBeans (EJB);
2. J2EE Connector Architecture (JCA);
3. JavaBeans Activation Framework (JAF);
4. JavaMail API;
5. Java Message Service API;
6. Java Persistence API;
7. Java Transaction API (JTA).