CBCS SCHEME

USN 1CY17MCA78

16MCA553

Fifth Semester MCA Degree Examination, Dec.2018/Jan.2019

Service Oriented Architecture

Time: 3 hrs. Max. Marks: 80

Note: *Answer FIVE full questions, choosing ONE full question from each module.*

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and /or equations written eg, 42+8 = 50, will be treated as malpractice.

Module-1

1 a. Explain in detail the main principles adopted during the service design. (08 Marks)
b. Discuss with suitable diagram the primary characteristics of 2-tier client server architecture in comparison with corresponding SOA. (08 Marks)

OR

2 a. What are the characteristics of contemporary SOA? Explain. (08 Marks)
b. Explain in detail the SOA time-line. (08 Marks)

Module-2

3 a. Discuss the primitive and complex message exchange patterns. (08 Marks)
b. With neat diagram explain coordinator service model and also explain service coordinator composition. (08 Marks)

OR

4 a. Explain with neat diagram the structure of SOAP. (08 Marks)
b. Compare and contrast the choreography and orchestration in SOA with reference to web service. (08 Marks)

Module-3

5 a. What are the security requirements in web service design? (08 Marks)
b. How components in SOA inter-relate? Explain. (08 Marks)

OR

6 a. With suitable diagram explain reliable messaging model. (08 Marks)
b. How service-orientation principles relate to object-orientation principles? Discuss. (08 Marks)

Module-4

7 a. Explain business service models in detail. (08 Marks)
b. Discuss service-oriented business process redesign. (08 Marks)

OR

8 a. What are the application services characteristics? Explain. (08 Marks)
b. Explain WS-BPEL languages basics. (08 Marks)

Module-5

9 a. What are architectural considerations? Explain. (08 Marks)
b. Explain packaged application platform. (08 Marks)

OR

10 a. Discuss architectural elaboration process. (08 Marks)
b. Explain the principles in application server platform. (08 Marks)

* * * * *

CBSE Scheme – VTU MCA Degre Exam – Dec 2018/Jan 2019

16MCA553 - Service Oriented Architecture

1.a) Explain in detail the main principles adopted during the service design

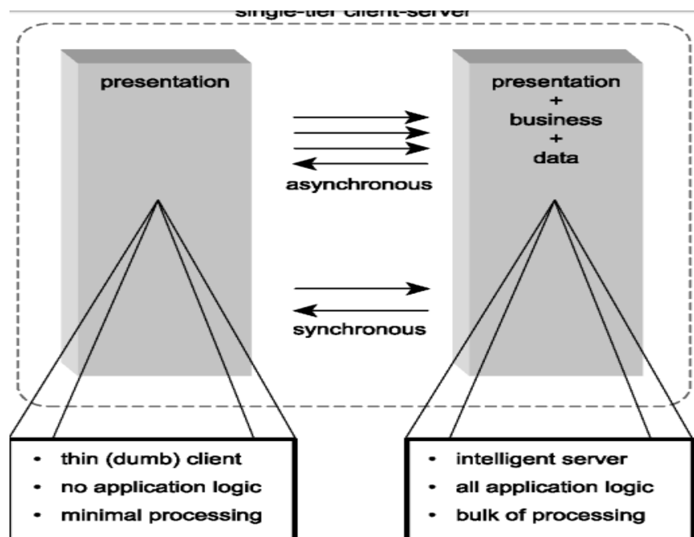
- *Loose coupling* Services maintain a relationship that minimizes dependencies and only requires that they retain an awareness of each other.
- *Service contract* Services adhere to a communications agreement, as defined collectively by one or more service descriptions and related documents.
- *Autonomy* Services have control over the logic they encapsulate.
- *Abstraction* Beyond what is described in the service contract, services hide logic from the outside world.
- *Reusability* Logic is divided into services with the intention of promoting reuse.
- *Composability* Collections of services can be coordinated and assembled to form composite services.
- *Statelessness* Services minimize retaining information specific to an activity.
- *Discoverability* Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

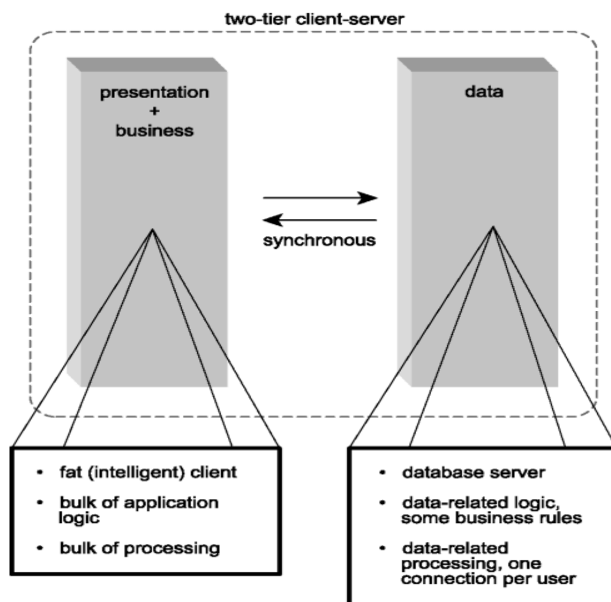
1.b) **Discuss with suitable diagram the primary characteristics of 2 tier client server architecture in comparison with corresponding SOA**

Architecture is an abstract which explains the technology, boundaries, rules, limitations, and design
characteristics that apply to all solutions based on this template.

Client-server architecture: a brief history
□□The original monolithic mainframe systems that empowered organizations to get seriously computerized often are considered the first inception of client-server architecture.
□□These environments, in which bulky mainframe back-ends served thin clients, are considered an implementation of the single-tier client-server architecture
□□Mainframe systems natively supported both synchronous and asynchronous communication.
Example: The latter approach was used primarily to allow the server to continuously receive characters from the terminal in response to individual key-strokes. Only upon certain conditions would the server actually respond.
□□This new approach introduced the concept of delegating logic and processing duties onto individual workstations, resulting in the birth of the fat client.
□□Further supported by the innovation of the graphical user-interface (GUI), two-tier client-server was considered a huge step forward.
□□The common configuration of this architecture consisted of multiple fat clients, each with its own connection to a database on a central server.

□□Client-side software performed the bulk of the processing, including all presentation-related and most data access logic One or more servers facilitated these clients by hosting scalable RDBMSs.





The primary characteristics of the two-tier client-server architecture individually and comparing them to the corresponding parts of SOA.

Application logic

□□**Client-server environments** place the majority of application logic into the client software. This results in a monolithic executable that controls the user experience, as well as the back-end resources. One exception is the distribution of business rules.

□□A popular trend was to embed and maintain business rules relating to data within stored procedures and triggers on the database.

This somewhat abstracted a set of business logic from the client and simplified data access programming.
The presentation layer within contemporary service-oriented solutions can vary. Any piece of software capable of exchanging SOAP messages according to required service contracts can be classified as a service requestor.
While it is commonly expected for requestors to be services as well, presentation layer designs are completely open and specific to a solution's requirements.
Within the server environment, options exist as to where application logic can reside and how it can be distributed. These options do not preclude the use of database triggers or stored procedures.
Service-oriented design principles come into play, often dictating the partitioning of processing logic into autonomous units.
This facilitates specific design qualities, such as service statelessness and interoperability, as well as future composability and reusability.
It common with an SOA for these units of processing logic to be solution-agnostic. This supports the ultimate goal of promoting reuse and loose coupling across application boundaries.

Application processing

Client-server application logic resides in the client component; the client workstation is responsible for the bulk of the processing.
The 80/20 ratio often is used as a rule of thumb, with the database server typically performing twenty percent of the work.
A two-tier client-server solution with a large user-base generally requires that each client establish its own database connection.
Communication is predictably synchronous, and these connections are often persistent (meaning that they are generated upon user login and kept active until the user exits the application).
Proprietary database connections are expensive, and the resource demands sometimes overwhelm database servers, imposing processing latency on all users.

In SOA is highly distributed. Each service has an explicit functional boundary and related resource requirements. In modeling a technical service-oriented architecture, many choices are there to position and deploy services.
Enterprise solutions consist of multiple servers, each hosting sets of Web services and supporting middleware.
There is, therefore, no fixed processing ratio for SOAs. Services can be distributed as required, and performance demands are one of several factors in determining the physical deployment configuration.
Communication between service and requestor can be synchronous or asynchronous.
This flexibility allows processing to be further streamlined, especially when asynchronous message patterns are utilized.
Additionally, by placing a large amount of intelligence into the messages, options for achieving message-level context management are provided.
This promotes the stateless and autonomous nature of services and further alleviates processing by reducing the need for runtime caching of state information.

Technology

Client-server applications use the 4GL programming languages, such as Visual Basic and PowerBuilder.

These development environments took better advantage of the Windows operating system by providing the ability to create aesthetically rich and more interactive user-interfaces.
3GL languages, such as C++, were also still used, especially for solutions that had more rigid performance requirements.
On the back-end, major database vendors, such as Oracle, Informix, IBM, Sybase, and Microsoft, provided robust RDBMSs that could manage multiple connections, while providing flexible data storage and data management features.
The technology set used by SOA actually has not changed as much as it has expanded
Newer versions of older programming languages, such as Visual Basic, still can be used to create Web services, and the use of relational databases still is commonplace.
The technology landscape of SOA, though, has become increasingly diverse.
In addition to the standard set of Web technologies (HTML, CSS, HTTP, etc.) contemporary SOA brings with it the absolute requirement that an XML data representation architecture be established, along with a SOAP messaging framework, and a service architecture comprised of the ever-expanding Web services platform.

Security

Client-server architecture security is centralized at the server level
Databases are sufficiently sophisticated to manage user accounts and groups and to assign these to individual parts of the physical data model.
Security also can be controlled within the client executable, especially when it relates to specific business rules that dictate the execution of application logic (such as limiting access to a part of a user-interface to select users).
Operating system-level security can be incorporated to achieve a single sign-on, where application clearance is derived from the user's operating system login account information.
In the distributed world of SOA, this is not possible. Security becomes a significant complexity directly relational to the degree of security measures required.
Multiple technologies are typically involved, many of which comprise the WS-Security framework.

Administration

One of the main reasons the client-server era ended was the increasingly large maintenance costs associated with the distribution and maintenance of application logic across user workstations.
Each client housed the application code each update to the application required a redistribution of the client software to all workstations.
In larger environments, this resulted in a highly burdensome administration process.
Maintenance issues spanned both client and server ends. Client workstations were subject to environment-specific problems because different workstations could have different software programs installed or may have been purchased from different hardware vendors.
There were increased server-side demands on databases, especially when a client-server application expanded to a larger user base.
Service-oriented solutions can have a variety of requestors; they are not necessarily immune to client-side maintenance challenges. While their distributed back-end does accommodate scalability for application and database servers, new administration demands can be introduced.
For example, once SOAs evolve to a state where services are reused and become part of multiple service compositions, the management of server resources and service interfaces **can require powerful administration tools,** including the use of a private registry

2.a) What are the characteristics of contemporary SOA? Explain

1. Contemporary SOA is at the core of the service-oriented computing platform.
2. Contemporary SOA increases quality of service.
3. Contemporary SOA is fundamentally autonomous.
4. Contemporary SOA is based on open standards.
5. Contemporary SOA supports vendor diversity.
6. Contemporary SOA fosters intrinsic interoperability.
7. Contemporary SOA promotes discovery.
8. Contemporary SOA promotes federation.
9. Contemporary SOA promotes architectural composability.
10. Contemporary SOA fosters inherent reusability.
11. Contemporary SOA emphasizes extensibility.
12. Contemporary SOA supports a service-oriented business modeling paradigm.
13. Contemporary SOA implements layers of abstraction.
14. Contemporary SOA promotes loose coupling throughout the enterprise.
15. Contemporary SOA promotes organizational agility.
16. Contemporary SOA is a building block.
17. Contemporary SOA is an evolution.
18. Contemporary SOA is still maturing.
19. Contemporary SOA is an achievable ideal.

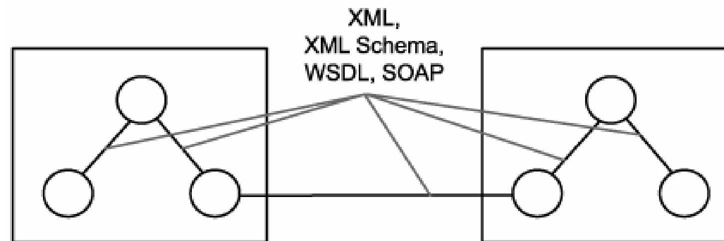
1. **Contemporary SOA is at the core of the service-oriented computing platform.**
   - SOA is used to qualify products, designs, and technologies an application computing platform consisting of Web services technology and service-orientation principles
   - *Contemporary SOA represents an architecture that promotes service-orientation through the use of Web services*.
2. **Contemporary SOA increases quality of service.**
   - The ability for tasks to be carried out in a secure manner, protecting the contents of a message, as well as access to individual services.
   - Allowing tasks to be carried out reliably so that message delivery or notification of failed delivery can be guaranteed.
   - Performance requirements to ensure that the overhead imposed by SOAP message and XML content processing does not inhibit the execution of a task.
   - Transactional capabilities to protect the integrity of specific business tasks with a guarantee that should the task fail, exception logic is executed.

3.Contemporary SOA is fundamentally autonomous.

- The service-orientation principle of autonomy requires that
    - individual services be as independent and
    - self-contained as possible with respect to the control they maintain over their underlying logic.

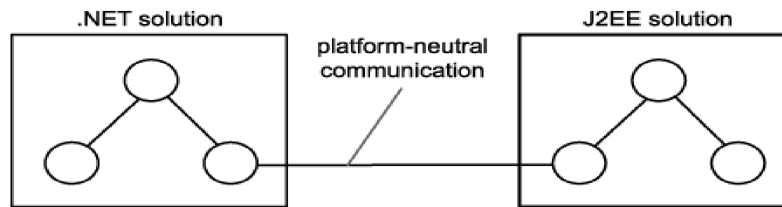
4.Contemporary SOA is based on open standards.

- Significant characteristic of Web services is the fact that
    - data exchange is governed by open standards.
    - After a message is sent from one Web service to another it travels via a set of protocols that is globally standardized and accepted.



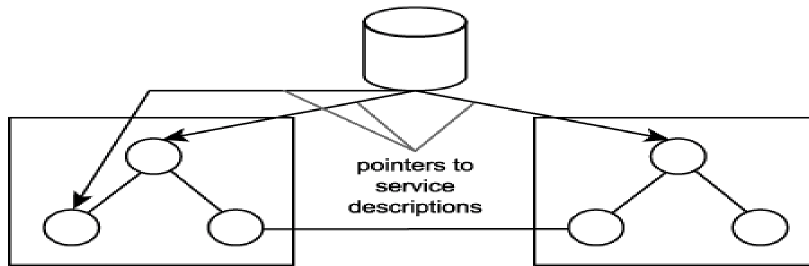
5.Contemporary SOA supports vendor diversity.

- Organizations continue itsbuilding solutions with existing development tools and server products.
- It is continue to leveraging(maximizing) the skill sets of in-house resources.
- Choice to explore the offerings of new vendors is always possible.
- This option is made possible by the
    - open technology provided by the Web services framework
    - the standardization and principles introduced by SOA.



6.Contemporary SOA promotes discovery.

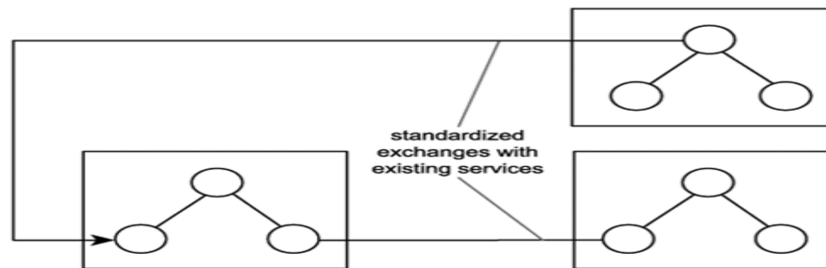
- SOA supports and encourages the advertisement and discovery of services throughout the enterprise and beyond.
- A serious SOA will likely rely on some form of service registry or directory to manage service descriptions



7.Contemporary SOA foster(advance) intrinsic(essential) interoperability.

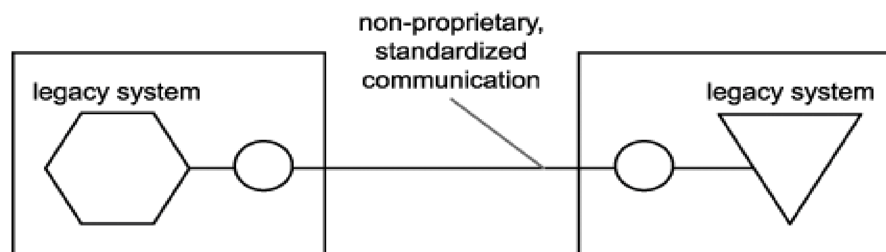
- To leveraging(maximizing) and supporting the
  - required usage of open standards,
  - a vendor diverse environment, and
  - the availability of a discovery mechanism

  is called intrinsic interoperability
- Whether an application actually has immediate integration requirements or not design principles can be applied to outfit services with characteristics that naturally promote interoperability.



8.Contemporary SOA promotes federation.

- Establishing SOA within an enterprise does not necessarily require that you replace what you already have.
- SOA has the ability to introduce unity across previously non-federated environments.
- Web services enable federation
- SOA promotes by establishing and standardizing the ability to encapsulate legacy and non-legacy application logic and by exposing it via a common, open, and standardized communications framework



2.b) Explain in detail the SOA timeline

An SOA timeline (from XML to Web services to SOA)

- XML was a W3C creation derived from the popular Standard Generalized Markup Language (SGML) that has existed since the late 60s.
- **World Wide Web Consortium** (**W3C**) is the main international standards organization for WWW
- purpose : Working together in the development of standards for the World Wide Web
- Widely used meta language
- It allowed Organizations to add intelligence to raw document data.
- XML gained popularity during the eBusiness movement of the late 90s
- Server-side scripting languages made conducting business via the Internet viable.
- By XML, developers were able to attach meaning and context to any piece of information transmitted across Internet protocols.
- Not only was XML used to represent data in a standardized manner, but the language itself was used as the basis for a series of additional specifications.
- The XML Schema Definition Language (XSD) and the XSL Transformation Language (XSLT) were both authored using XML.
- These specifications, in fact, have become key parts of the core XML technology set.
- The XML data representation architecture represents the foundation layer of SOA.
- XSD schemas preserve the integrity and validity of message data,
- XSLT is employed to enable communication between disparate data representations through schema mapping.

4.1.2. Web services: a brief history

- In 2000, the W3C received a submission for the **Simple Object Access Protocol (SOAP)** specification. It is designed to unify the RPC communication.
- The idea was for parameter data transmitted between components to be serialized into XML, transported, and then deserialized back into its native format.
- Increase in the eBusiness technology, vendor expecting proprietary-free Internet communications framework.
- This ultimately led to the idea of creating a pure, Web-based, distributed technology one that could leverage the concept of a standardized communications framework to bridge the enormous disparity that existed between and within organizations. This concept was called Web services.
- The most important part of a Web service is its public interface.

- It is a central piece of information that assigns the service an identity and enables its invocation.
- So, one of the first initiatives in support of Web services was the **Web Service Description Language(WSDL).**
- To open interoperability, Web services required an Internet-friendly and XML-compliant communications format that could establish a standardized messaging framework.
- Alternatives, such as XML-RPC, were considered, SOAP won out as the industry favorite and remains the foremost messaging standard for use with Web services.
- In support of SOAP's new role, the specification to allow for both RPC-style and document-style message types.
- The document style message used more frequently within SOAs.
- Eventually, the word "SOAP" was no longer considered an acronym for "Simple Object Access Protocol."
- As of version 1.2 of the specification, it became a standalone term.
- UDDI(Universal Description, Discovery, and Integration) specification added in first generation web services.. Originally developed by UDDI.org, it was submitted to OASIS, which continued its development in collaboration with UDDI.org.
- This specification allows for the creation of standardized service description registries both within and outside of organization boundaries. UDDI provides the potential for Web services to be registered in a central location, from where they can be discovered by service requestors.
- Unlike WSDL and SOAP, UDDI has not yet attained industry-wide acceptance, and remains an optional extension to SOA.
- Existing messaging platforms, such as messaging-oriented middleware (MOM) products, incorporated Web services to support SOAP in addition to other message protocols.
- Some organizations were also able to immediately incorporate Web services to facilitate B2B data exchange requirements often as an alternative to EDI (Electronic Data Interchange).

4.1.3. SOA: a brief history

- First-generation Web services standards fulfilled this model as follows:
- WSDL described the service.
- SOAP provided the messaging format used by the service and its requestor.
- UDDI provided the standardized service registry format.
- Numerous of the contemporary SOA characteristics
- Aggressive development and collaborative initiatives which have produced a series of extensions to the first-generation Web services platform. Known as the "second-generation" or "WS-*" specifications
- These extensions address specific areas of functionality with the overall goal of elevating the Web services technology platform to an enterprise level
- Through service-orientation, business logic can be cleanly encapsulated and abstracted from the underlying automation technology.

- This vision has been further supported by the rise of business process definition languages, most notably WS-BPEL.

How SOA is re-shaping XML and Web services

- SOA introduces boundaries and rules.
- Contemporary SOA is made possible by the XML and Web services technology platforms.
- These platforms are required to undergo a number of changes in order for their respective technologies to be properly positioned and utilized within the confines of service-oriented architectures.
- Traditional distributed application environments that use XML or Web services are therefore in for some rewiring as service-oriented design principles require a change in both technology and mindset.
- Following are some examples of potential issues you may be faced with when having to retrofit existing implementations.
- SOA requires that data representation and service modeling standards now be kept in alignment. Fundamental to fostering intrinsic interoperability.
- SOA relies on SOAP messaging for all inter-service communication.
- SOA standardizes the use of a document-style messaging. The shift from RPC-style to document-style messages imposes change on the design of service descriptions.
- Due to this emphasis on document-style SOAP messages, SOA promotes a content and intelligence-heavy messaging model. This supports service statelessness and autonomy, and minimizes the frequency of message transmissions
- Until the advanced messaging capabilities of WS-* extensions become commonplace, many applications will need to be outfitted with custom SOAP headers to implement interim solutions to manage complex message exchanges

Module -2

3.a) Discuss the primitive and complex message exchange pattern

Message exchange patterns

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

The fundamental characteristic of the fire-and-forget pattern is that a response to a transmitted message is not expected.

Message exchange patterns (MEPs) represent a set of templates that provide a group of already mapped out sequences for the exchange of messages. The most common example is a request and response pattern. Here the MEP states that upon successful delivery of a message from one service to another, the receiving service responds with a message back to the initial requestor.
Many MEPs have been developed, each addressing a common message exchange requirement. It is useful to have a basic understanding of some of the more important MEPs, as you will no doubt be finding yourself applying MEPs to specific communication requirements when designing service-oriented solutions.

Primitive MEPs

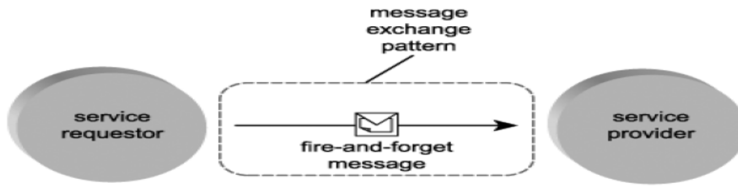
Before the arrival of contemporary SOA, messaging frameworks were already well used by various messaging-oriented middleware products. As a result, a common set of primitive MEPs has been in existence for some time.

Request-response

This is the most popular MEP in use among distributed application environments and the one pattern that defines synchronous communication (although this pattern also can be applied asynchronously).
The request-response MEP establishes a simple exchange in which a message is first transmitted from a source (service requestor) to a destination (service provider). Upon receiving the message, the destination (service provider) then responds with a message back to the source (service requestor).

Fire-and-forget

This simple asynchronous pattern is based on the unidirectional transmission of messages from a source to one or more destinations .
A number of variations of the fire-and-forget MEP exist, including:
The single-destination pattern, where a source sends a message to one destination only.
The multi-cast pattern, where a source sends messages to a predefined set of destinations.
The broadcast pattern, which is similar to the multi-cast pattern, except that the message is sent out to a broader range of recipient destinations.



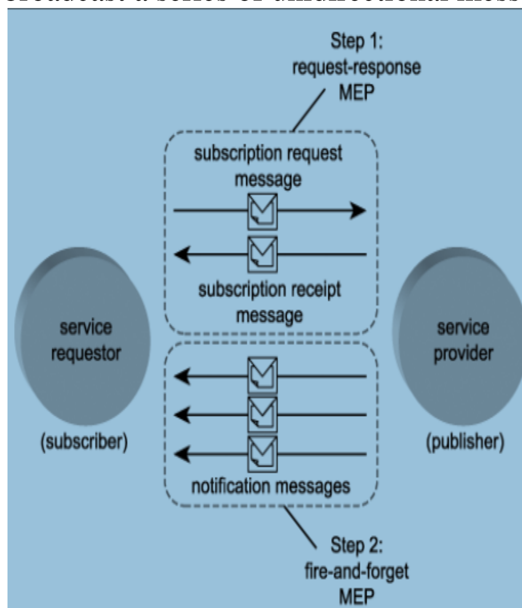
Complex MEPs

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

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

Step 1 in the publish-and-subscribe MEP could be implemented by a request-response MEP, where the subscriber's request message, indicating that it wants to subscribe to a topic, is responded to by a message from the publisher, confirming that the subscription succeeded or failed.

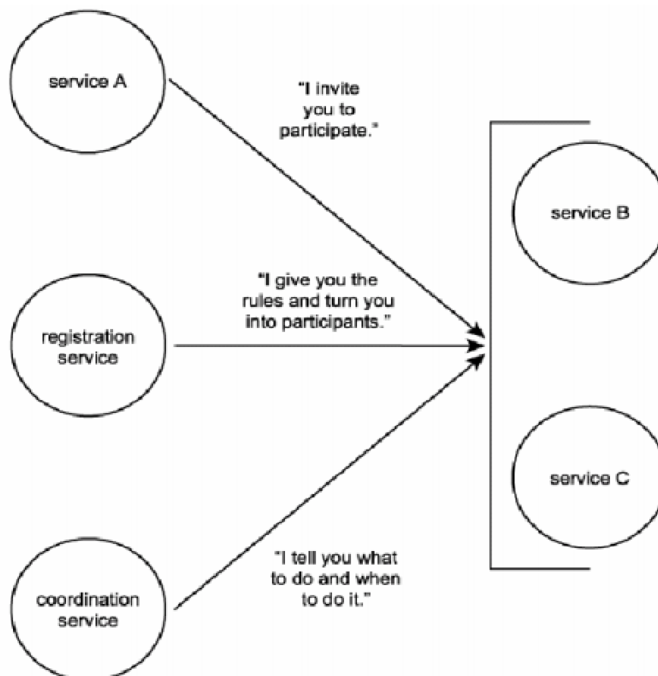
Step 2 then could be supported by one of the fire-and-forget patterns, allowing the publisher to broadcast a series of unidirectional messages to subscribers



3.b) With neat diagram explain coordinator service model and also explain service coordinator composition

- Every activity introduces a level of context into an application runtime environment. Something that is happening or executing has meaning during its lifetime, and the description of its meaning (and other characteristics that relate to its existence) can be classified as context information.
- The more complex an activity, the more context information it tends to bring with it. The complexity of an activity can relate to a number of factors, including:
  - the amount of services that participate in the activity
  - the duration of the activity
  - the frequency with which the nature of the activity changes – whether or not multiple instances of the activity can concurrently exist
- A framework is required to provide a means for context information in complex activities to be managed, preserved and/or updated, and distributed to activity participants. Coordination establishes such a framework.

Coordination provides services that introduce controlled structure into activities



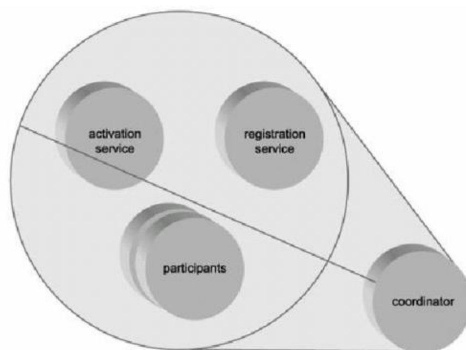
Coordinator composition

WS-Coordination establishes a framework that introduces a generic service based on the coordinator service model. This service controls a composition of three other services that each play a specific part in the management of context data

The coordinator composition consists of the following services:

- **Activation service** Responsible for the creation of a new context and for associating this context to a particular activity.
- **Registration service** Allows participating services to use context information received from the activation service to register for a supported context protocol.

The coordinator service composition



- **Protocol-specific services** These services represent the protocols supported by the coordinator's coordination type.
- **Coordinator** The controller service of this composition, also known as the coordination service.

4.a) Explain with neat diagram the structure of SOAP

5.4. Messaging (with SOAP)

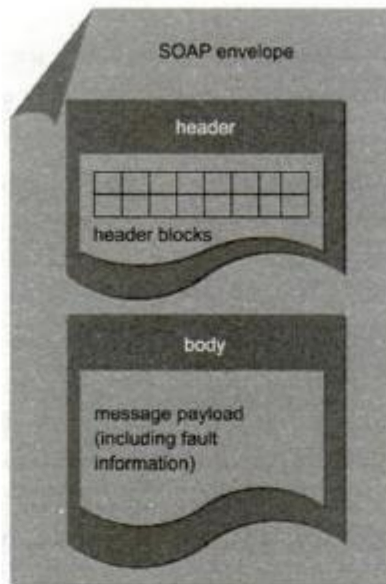
- All communication between services is message-based,
- The messaging framework chosen must be standardized so that all services, regardless of origin, use the same format and transport protocol.
- Message-centric application design that an increasing amount of business and application logic is embedded 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



gure 5.21
e basic structure of a SOAP message.

- Each message can contain a *header*, an area dedicated to hosting meta information.
- Service-oriented solutions, this header section important
- The actual message contents consists of XML formatted data.
- The contents of a message body are often referred to as the message *payload*.

Header blocks

- SOAP communications framework used by SOAs, the creating messages that are intelligence-heavy and self-sufficient
- Independence that increases the robustness and extensibility
- Message independence is implemented through the use of *header blocks*
- packets of supplementary meta information stored in the envelope's header area.
- It further reinforces the characteristics of contemporary SOA related to fostering reuse, interoperability, and composability.
- Examples of the types of features a message can be outfitted with using header blocks include:
    - processing instructions that may be executed by service intermediaries or the ultimate receiver
    - routing or workflow information associated with the message
    - security measures implemented in the message

  - reliability rules related to the delivery of the message
  - context and transaction management information
  - correlation information

Message styles

- The SOAP specification was originally designed to replace proprietary RPC protocols
- Distributed components to be serialized into XML documents, transported, and then deserialized into the native component format upon arrival.

Two types of Message styles

1. *RPC-style* message runs contrary to the emphasis SOA places on independent, intelligence-heavy messages.

2. SOA relies on *document-style* messages to enable larger payloads, coarser interface operations, and reduced message transmission volumes between services.

Attachments

- To facilitate requirements for the delivery of data not so easily formatted into an XML document, the use o*S*f *OAP attachment* technologies exist.
- Each provides a different encoding mechanism used to bundle data in its native format with a SOAP message.
- SOAP attachments are commonly employed to transport binary files, such as images.

Faults

- SOAP messages offer the ability to add exception handling logic by providing an optiona*fla ult* section
- It resides within the body area.
- The typical use for this section is to store a simple message used to deliver error condition information when an exception occurs.

4.b) Compare and contrast the choreography and orchestration in SOA with reference to web service.

- Both Orchestrations and choreographies represent complex message interchange patterns
- Both include multi-organization participants.
- An orchestration expresses organization-specific business workflow. This means that an organization owns and controls the logic behind an orchestration, even if that logic involves interaction with external business partners.
- A choreography, on the other hand, is not necessarily owned by a single entity. It acts as a community interchange pattern used for collaborative purposes by services from different provider entities

Figure 6.39. A choreography enabling collaboration between two different orchestrations

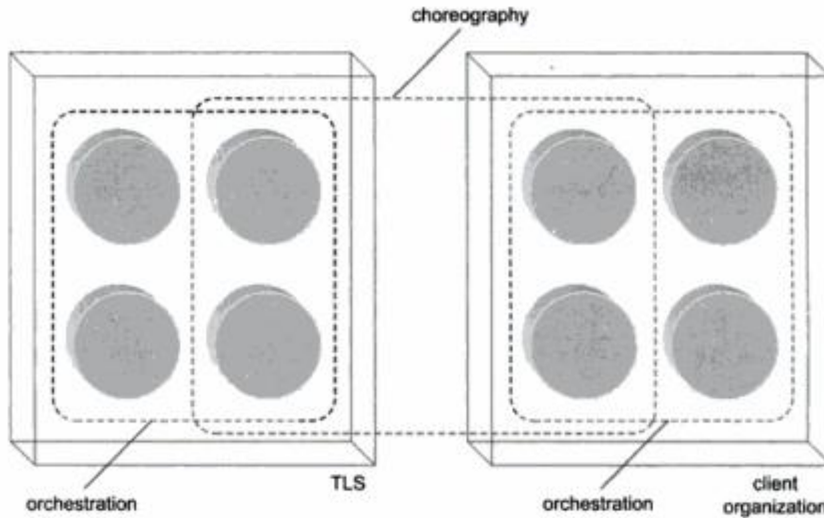


Figure 6.39
A choreography enabling collaboration between two different orchestrations.

Module – 3

5.a) What are the security requirement in web service design

WS-Security specification comprise such a framework, further broadened by a series Of supplementary specifications with specialized feature sets. The basic functions performed by the following three core specifications:

- WS-Security
- XML-Signature
- XML-Encryption

A family of security extensions parented by the WS-Security specification comprise such a framework, further broadened by a series Of supplementary specifications with specialized feature sets. The basic functions performed by the following three core specifications:

- WS-Security
- XML-Signature
- XML-Encryption

Additionally, we'll briefly explore the fundamental concepts behind single sign-on, a form of centralized security that complements these WS-Security extensions. Before we begin, it is worth noting that this section organizes security concepts as they pertain to and support the following five common security requirements: identification, authentication, authorization, confidentiality, and integrity.

Identification, authentication, and authorization

For a service requestor to access a secured service provider, it must first provide information that expresses its origin or owner. This is referred to as making a claim Claims are represented by identification information stored in the SOAP header. WS-Security establishes a standardized header block that stores this information, at which point it is referred to as a token.

Authentication requires that a message being delivered to a recipient prove that the message is in fact from the sender that it claims to be. In other words, the service must provide proof that its claimed identity is true.

Single sign-on

A challenge facing the enablement of authentication and authorization within SOA is propagating the authentication and authorization information for a service requestor across multiple services behind the initial service provider. Because services are autonomous and independent from each other, a mechanism is required to persist the security context established after a requestor has been authenticated. Otherwise, the requestor would need to re-authenticate itself with every subsequent request. The concept of single sign-on addresses this issue. The use of a single sign-on technology allows a service requestor to be authenticated once and then have its security context information shared with other services that the requestor may then access without further authentication.

There are three primary extensions that support the implementation of the single signon concept:

• SAML (Security Assertion Markup Language)
• .NET Passport
• XACML (XML Access Control Markup Language)

Confidentiality and integrity

Confidentiality is concerned with protecting the privacy of the message contents. A message is considered to have remained confidential if no service or agent in its message path not authorized to do so viewed its contents.

Transport-level security and message-level security

The type of technology used to protect a message determines the extent to which the message remains protected while making its way through its message path. Secure Sockets Layer (SSL), for example, is a very popular means of securing the HTTP channel upon which requests and responses are transmitted. However, within a Web services-
based communications framework, it can only protect a message during the transmission between service endpoints. Hence, SSL only affords us transport-level security

Integrity means ensuring that a message's contents have not changed during transmission. Transport-level security only protects the message during transit between service endpoints.

Encryption and digital signatures

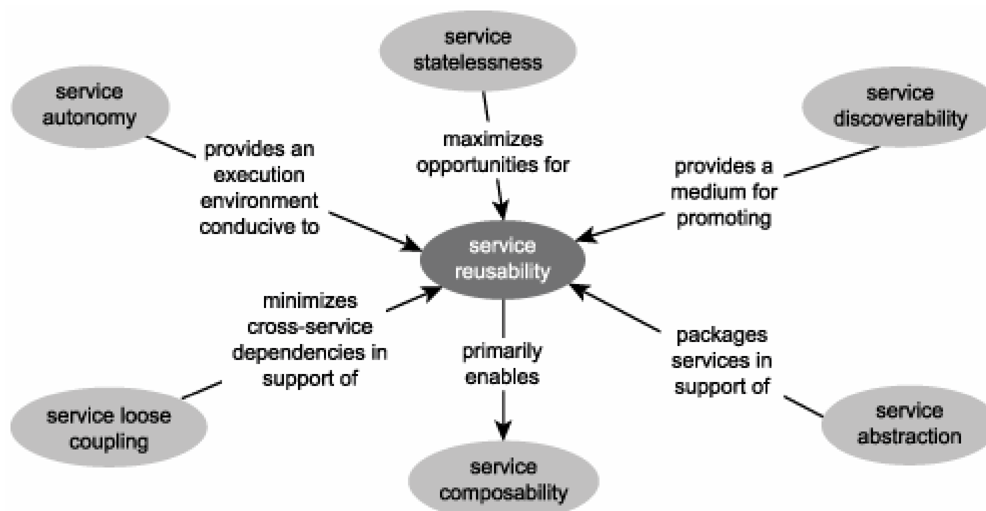
Message-level confidentiality for an XML-based messaging format, such as SOAP, can be realized through the use of specifications that comprise the WS-Security framework. In this section we focus on XML-Encryption and XML-Signature, two of the more important WS-Security extensions that provide security controls that ensure the confidentiality and integrity of a message.

XML-Encryption, an encryption technology designed for use with XML, is a cornerstone part of the WS-Security framework. It provides features with which encryption can be applied to an entire message or only to specific parts of the message (such as the password).

5.b) How compoentes in SOA inter-relate? Explain

1. Service reusability

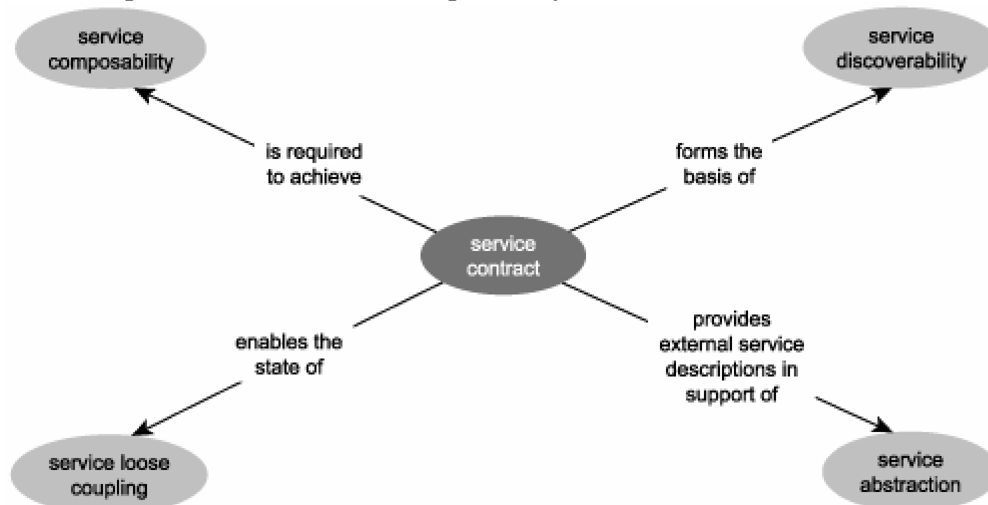
- Service autonomy establishes an execution environment that facilitates reuse because the service has independence and self-governance. The less dependencies a service has, the broader the applicability of its reusable functionality.
- Service statelessness supports reuse because it maximizes the availability of a service and typically promotes a generic service design that defers activity-specific processing outside of service logic boundaries.
- Service abstraction fosters reuse because it establishes the black box concept, where processing details are completely hidden from requestors. This allows a service to simply express a generic public interface.
- Service discoverability promotes reuse, as it allows requestors (and those that build requestors) to search for and discover reusable services.



2.Service contract

- Service abstraction is realized through a service contract, as it is the metadata expressed in the contract that defines the only information made available to service requestors. All additional design, processing, and implementation details are hidden behind this contract.

- Service loose coupling is made possible through the use of service contracts. Processing logic from different services do not need to form tight dependencies; they simply need an awareness of each other's communication requirements, as expressed by the service description documents that comprise the service contract.
- Service composability is indirectly enabled through the use of service contracts. It is via the contract that a controller service enlists and uses services that act as composition members.
- Service discoverability is based on the use of service contracts. While some registries provide information supplemental to that expressed through the contract, it is the service description documents that are primarily searched for in the service discovery process.



3.Service loose coupling

- Service reusability is supported through loose coupling because services are freed from tight dependencies on others. This increases their availability for reuse opportunities.
- Service composability is fostered by the loose coupling of services, especially when services are dynamically composed.
- Service statelessness is directly supported through the loosely coupled communications framework established by this
- principle.
- Service autonomy is made possible through this principle, as it is the nature of loose coupling that minimizes cross-service dependencies.
- Additionally, service loose coupling is directly implemented through the application of a related service-orientation principle: Service contracts are what enable loose coupling between services, as the contract is the only piece of information required for services to interact.

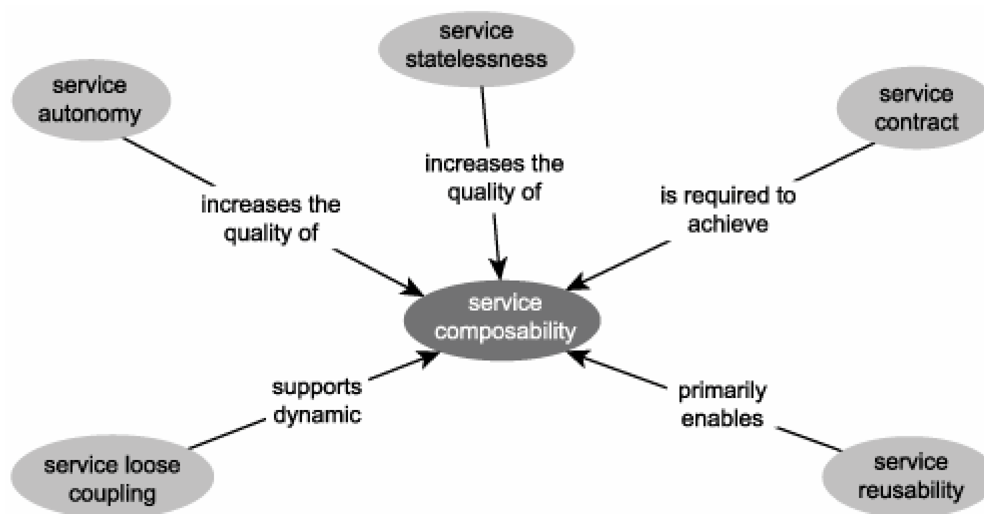
4.Service abstraction

- Service contracts, in a manner, implement service abstraction by providing the official description information that is made public to external service requestors.

- Service reusability is supported by abstraction, as long as what is being abstracted is actually reusable.

Service composability

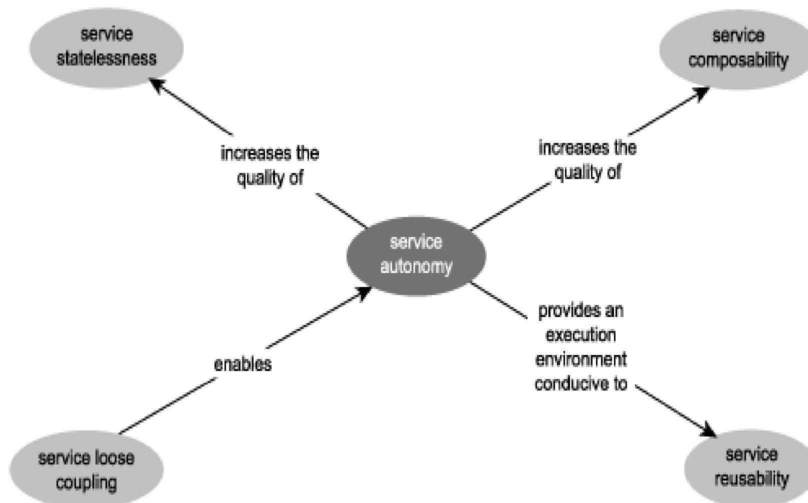
- Service reusability is what enables one service to be composed by numerous others. It is expected that reusable services can be incorporated within different compositions or reused independently by other service requestors.
- Service loose coupling establishes a communications framework that supports the concept of dynamic service composition.
- Because services are freed from many dependencies, they are more available to be reused via composition.
- Service statelessness supports service composability, especially in larger compositions. A service composition is reliant on the design quality and commonality of its collective parts. If all services are stateless (by, for example, deferring activity-specific logic to messages), the overall composition executes more harmoniously.
- Service autonomy held by composition members strengthens the overall composition, but the autonomy of the controller service itself actually is decreased due to the dependencies on its composition members.
- Service contracts enable service composition by formalizing the runtime agreement between composition members.



6. Service autonomy

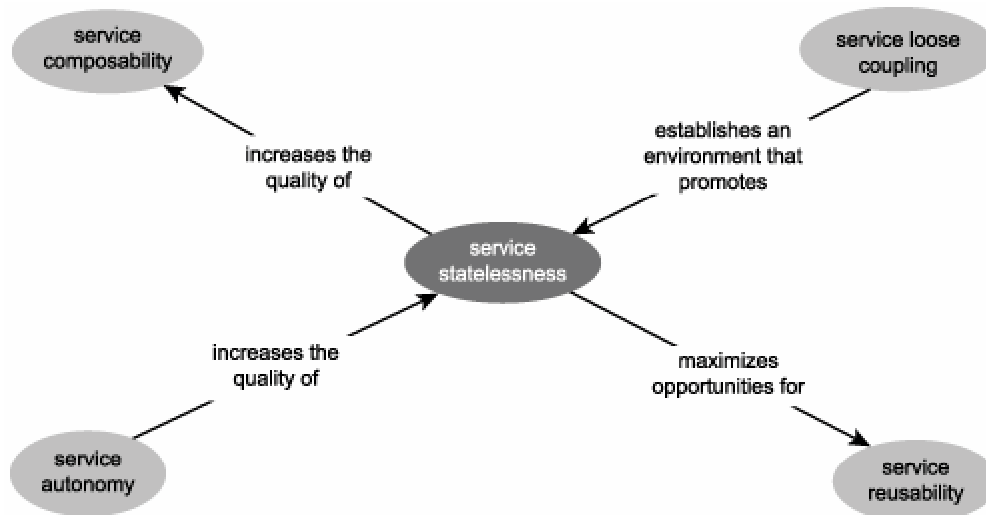
- Service reusability is more easily achieved when the service offering reusable logic has self-governance over its own logic.
- Service Level Agreement (SLA) type requirements that come to the forefront for utility services with multiple requestors, such as availability and scalability, are fulfilled more easily by an autonomous service.

- Service composability is also supported by service autonomyfor much of the same reasons autonomy supports service reusability. A service composition consisting of autonomous services is much more robust and collectively independent.
- Service statelessness is best implemented by a service that can execute independently. Autonomy indirectly supports service statelessness. (However, it is very easy to create a stateful service that is also fully autonomous.)
- Service autonomy is a quality that is realized by leveraging the loosely coupled relationship between services. Therefore service loose coupling is a primary enabler of this principle.



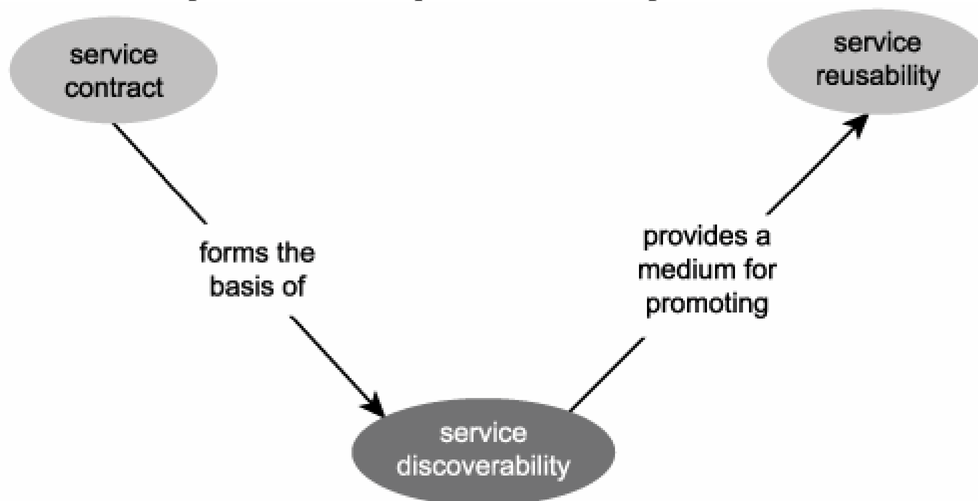
7. Service statelessness

- Service autonomy provides the ability for a service to control its own execution environment. By removing or reducing dependencies it becomes easier to build statelessness into services, primarily because the service logic can be fully customized to defer state management outside of the service logic boundary.
- Service loose coupling and the overall concept of loose coupling establishes a communication paradigm that is fully realized through messaging. This, in turn, supports service statelessness, as state information can be carried and persisted by the messages that pass through the services.
- Service statelessness further supports the following principles:
- Service composability benefits from stateless composition members, as they reduce dependencies and minimize the overhead of the composition as a whole.
- Service reuse becomes more of a reality for stateless services, as availability of the service to multiple requestors is increased and the absence of activity-specific logic promotes a generic service design.



8. Service discoverability

- Service contracts are what service requestors (or those that create them) actually discover and subsequently assess for suitability. Therefore, the extent of a service's discoverability can typically be associated with the quality or descriptiveness of its service contract.
- Service reusability is what requestors are looking for when searching for services and it is what makes a service potentially useful once it has been discovered. A service that isn't reusable would likely never need to be discovered because it would probably have been built for a specific service requestor in the first place.



6.a) With suitable diagram explain reliable messaging model

Reliable messaging addresses these concerns by establishing a measure of quality assurance that can be applied to other activity management frameworks

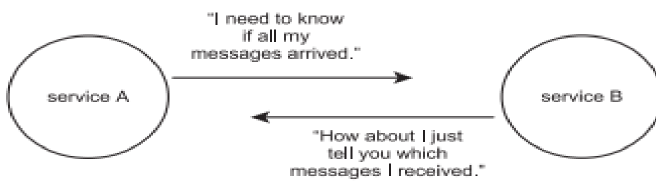


Figure 7.7
Reliable messaging provides a guaranteed notification of delivery success or failure.

WS-ReliableMessaging provides a framework capable of guaranteeing:

• that service providers will be notified of the success or failure of message transmissions

• that messages sent with specific sequence-related rules will arrive as intended (or generate a failure condition)

7.2.1 RM Source, RM Destination, Application Source, and Application Destination

These differentiations are necessary to abstract the reliable messaging framework from the overall SOA. An application source is the service or application logic that sends the message to the RM source, the physical processor or node that performs the actual wire transmission. Similarly, the RM destination represents the target processor or node that receives the message and subsequently delivers it to the application destination

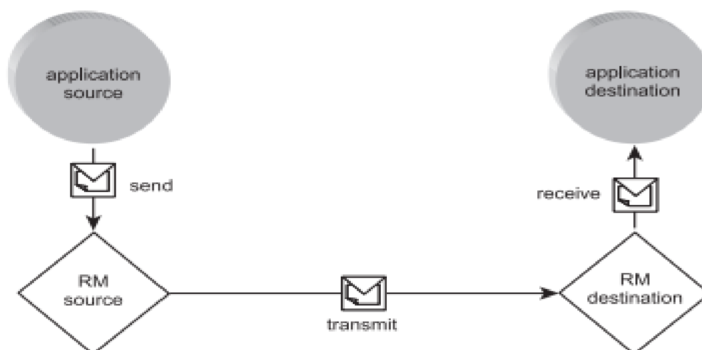


Figure 7.8
An application source, RM source, RM destination, and application destination.

7.2.2 Sequences

A sequence establishes the order in which messages should be delivered. Each message that is part of a sequence is labeled with a message number that identifies the position of the message within the sequence. The final message in a sequence is further tagged with a last message identifier.

7.2.3 Acknowledgements

A core part of the reliable messaging framework is a notification system used to communicate conditions from the RM destination to the RM source. Upon receipt of the message containing the last message identifier, the RM destination issues a sequence acknowledgement (Figure 7.9). The acknowledgement message indicates to the RM source which messages were received. It is up to the RM source to determine if the messages received are equal to the original messages transmitted. The RM source may retransmit any of the missing messages, depending on the delivery assurance used

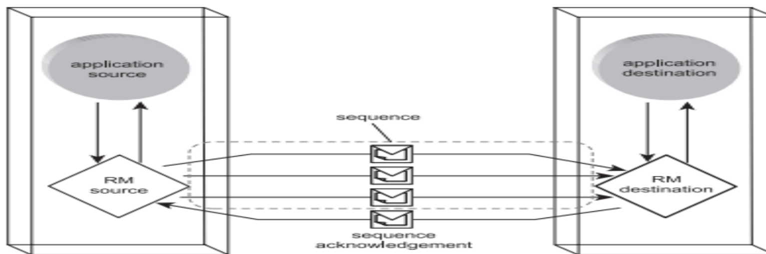


Figure 7.9
A sequence acknowledgement sent by the RM destination after the successful delivery of a sequence of messages.

An RM source does not need to wait until the RM destination receives the last message before receiving an acknowledgement. RM sources can request that additional acknowledgements be transmitted at any time by issuing request acknowledgements to RM destinations (Figure 7.10). Additionally, RM destinations have the option of transmitting negative acknowledgements that immediately indicate to the RM source that a failure condition has occurred

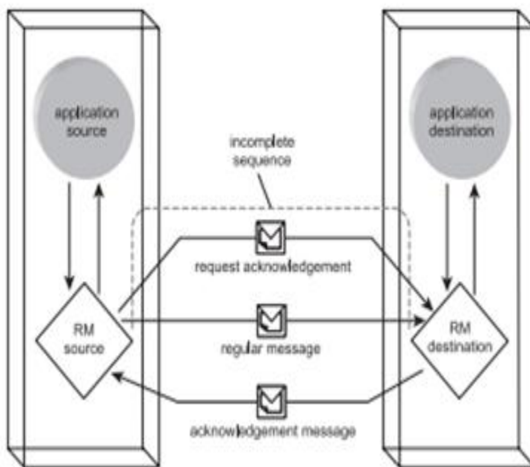


Figure 7.10
A request acknowledgement sent by the RM source to the RM destination, indicating that the RM source would like to receive an acknowledgement message before the sequence completes.

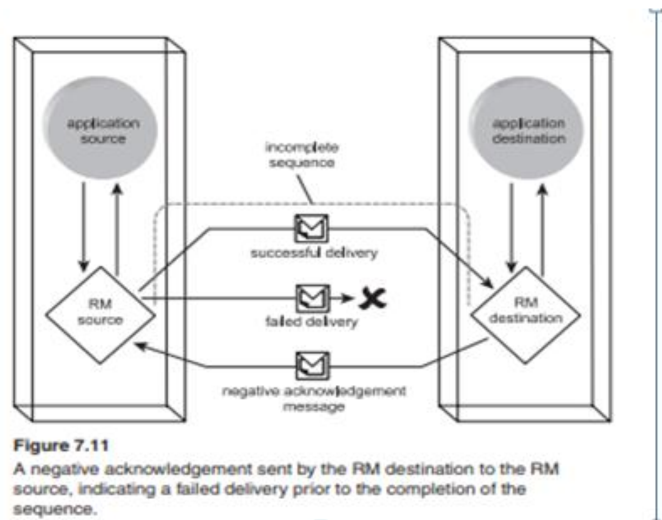


Figure 7.11
A negative acknowledgement sent by the RM destination to the RM source, indicating a failed delivery prior to the completion of the sequence.

7.2.4 Delivery assurances

The nature of a sequence is determined by a set of reliability rules known as delivery assurances. Delivery assurances are predefined message delivery patterns that establish a set of reliability policies. The following delivery assurances are supported:

- The AtMostOnce delivery assurance promises the delivery of one or zero messages. If more than one of the same message is delivered, an error condition occurs
- The AtLeastOnce delivery assurance allows a message to be delivered once or several times. The delivery of zero messages creates an error condition
- The ExactlyOnce delivery assurance guarantees that a message only will be delivered once. An error is raised if zero or duplicate messages are delivered
- The InOrder delivery assurance is used to ensure that messages are delivered in a specific sequence

The delivery of messages out of sequence triggers an error. Note that this delivery assurance can be combined with any of the previously described assurances.

6.b) How service orientation principles relate to object orientation principles. Discuss

Table 8.1. An overview of how service-orientation principles relate to object-orientation principles.

Service-Orientation Principle	Related Object-Orientation Principles
service reusability	Much of object-orientation is geared toward the creation of reusable classes. The object-orientation principle of modularity standardized decomposition as a means of application design. Related principles, such as abstraction and encapsulation, further support reuse by requiring a distinct separation of interface and implementation logic. Service reusability is therefore a continuation of this goal.
service contract	The requirement for a service contract is very comparable to the use of interfaces when building object-oriented applications. Much like WSDL definitions, interfaces provide a means of abstracting the description of a class. And, much like the "WSDL first" approach encouraged within SOA, the "interface first" approach also is considered an object-orientation best practice.
service loose coupling	Although the creation of interfaces somewhat decouples a class from its consumers, coupling in general is one of the primary qualities of service-orientation that deviates from object-orientation. The use of inheritance and other object-orientation principles encourages a much more tightly coupled relationship between units of processing logic when compared to the service-oriented design approach.
service abstraction	The object-orientation principle of abstraction requires that a class provide an interface to the external world and that it be accessible via this interface. Encapsulation supports this by establishing the concept of information hiding, where any logic within the class outside of what is exposed via the interface is not accessible to the external world. Service abstraction accomplishes much of the same as object abstraction and encapsulation. Its purpose is to hide the underlying details of the service so that only the service contract is available and of concern to service requestors.

service composability	Object-orientation supports association concepts, such as aggregation and composition. These, within a loosely coupled context, also are supported by service-orientation. For example, the same way a hierarchy of objects can be composed, a hierarchy of services can be assembled through service composability.
service autonomy	The quality of autonomy is more emphasized in service-oriented design than it has been with object-oriented approaches. Achieving a level of independence between units of processing logic is possible through service-orientation, by leveraging the loosely coupled relationship between services. Cross-object references and inheritance-related dependencies within object-oriented designs support a lower degree of object-level autonomy.
service statelessness	Objects consist of a combination of class and data and are naturally stateful. Promoting statelessness within services therefore tends to deviate from typical object-oriented design. Although it is possible to create stateful services and stateless objects, the principle of statelessness is generally more emphasized with service-orientation.
service discoverability	Designing class interfaces to be consistent and self-descriptive is another object-orientation best practice that improves a means of identifying and distinguishing units of processing logic. These qualities also support reuse by allowing classes to be more easily discovered. Discoverability is another principle more emphasized by the service-orientation paradigm. It is encouraged that service contracts be as communicative as possible to support discoverability at design time and runtime.

7.a) Explain business service models in details

Business services, on the other hand, are always an implementation of the business service model. The sole purpose of business services intended for a separate business service layer is to represent business logic in the purest form possible. This does not, however, prevent them from implementing other service models. For example, a business service also can be classified as a controller service and a utility service.

In fact, when application logic is abstracted into a separate application service layer, it is more than likely that business services will act as controllers to compose available application services to execute their business logic.

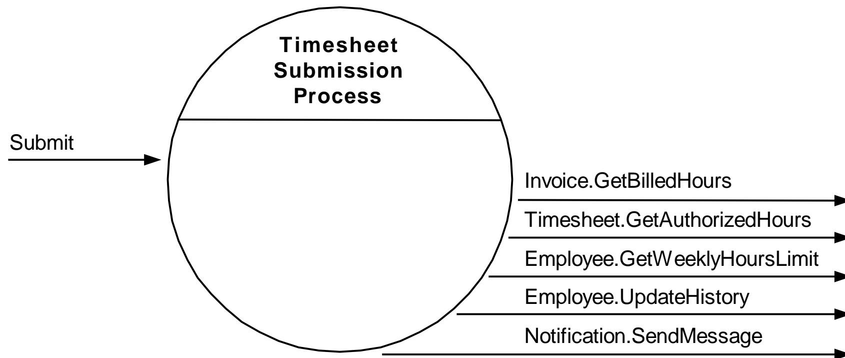
Business service layer abstraction leads to the creation of two further business service models:

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.

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).

b) Discuss service oriented business process redesign

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:



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.

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.

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.

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.

8 a) What are the application services characteristics? Explain

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

8 b) Explain WS_BPEL language basics

WS-BPEL language basics

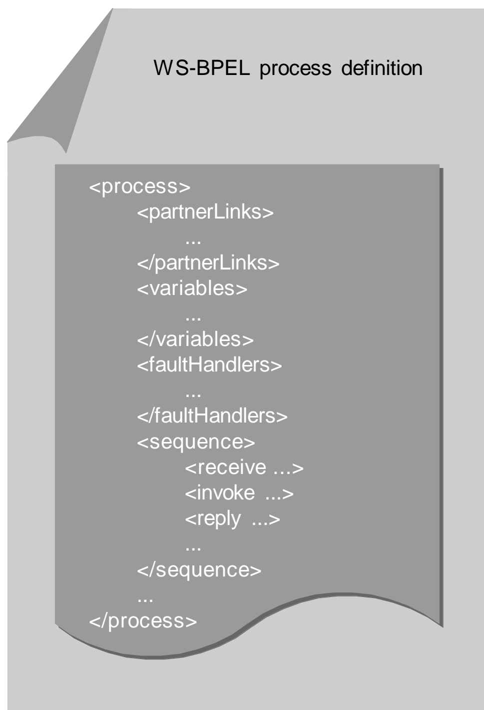


Figure 16-1 ***A common WS-BPEL process definition structure.***

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.

Module – 5

9 a) What are architectural considerations. Explain

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.

9 b) Explain packaged application platform

1. Enterprise Resource Planning : Enterprise resource planning or ERP software is a suite of applications that manages core business processes, such as sales, purchasing, accounting, Human Resource, customer support, CRM and inventory. It's an integrated system as opposed to individual software designed specifically for business process.

2. Supply chain Management : SCM encompasses the integrated planning and execution of processes required to optimize the flow of materials, information and financial capital in the areas that broadly include demand planning, sourcing, production, inventory management and storage, transportation -- or logistics -- and return for excess or defective products. Both business strategy and specialized software are used in these endeavors to create a competitive advantage. Supply chain management is an expansive, complex undertaking that relies on each partner -- from suppliers to manufacturers and

beyond -- to run well. Because of this, effective supply chain management also requires change management, collaboration and risk managementto create alignment and communication between all the entities

3. Customer Relationship management

Customer relationship management (CRM) is a term that refers to practices, strategies and technologies that companies use to manage and analyze customer interactions and data throughout the customer lifecycle, with the goal of improving customer service relationships and assisting in customer retention and driving sales growth.

10 a) Discuss architectural elaboration process

Solution architecture definition is a key part of enterprise application development lifecycle. Project teams in some cases conduct detailed design of enterprise application without taking key aspects of solution architecture into consideration. Consequently, such projects run into issues related to quality attributes such as performance, scalability, etc. on account of non-functional requirements not being addressed adequately. On a different dimension, the current technological advancements have been driving the need for formalization in solution architecture than ever before. This section describes the architecting process that can be employed for solution architecture definition of enterprise applications.

Solution architecture is elaborated in an iterative manner at different stages of the software development cycle on the basis of the perspectives of the stakeholders involved. Figure 4-1 shows three recommended levels of architecture that communicate the solution for a given set of requirements elaborated to the level of detail required by the stakeholders:

1. ***Level 0 – High-level architecture:*** In the initial stages of the project, before requirements are gathered and documented, the architect develops a high-level architecture with three views – *conceptual view,*

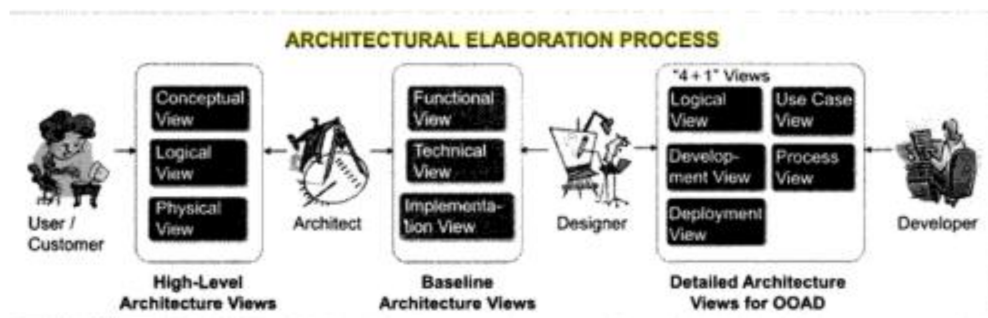


FIGURE 4-1 Architectural elaboration process.

logical view and physical view (Platt, 2002). The objective of this level is to validate the understanding of the scope and the high-level requirements provided by the business users (or customer when the solution development is done by an external organization) and develop a solution to meet the requirements.

2. ***Level 1 – Baseline architecture:*** On completion of requirements analysis phase of a project, the next version of the architecture is developed. This version of the architecture is represented with *functional view, technical view and deployment view*. The objective of this level (MSDN, 2004) is to identify architecturally significant use cases from a functional perspective and provide a solution from a technical perspective based on architectural decisions made after consideration of the various options available for technology platforms, frameworks, products and on analysis of the architecture tradeoffs. This version of the architecture serves as a baseline for a given set of requirements, architecture decisions and tradeoffs.

3. ***Level 2 – Detailed:*** At the start of the design process, baseline architecture is further elaborated and several views are added that can form the basis of the design process. This level provides detailed architectural views based on the methodology to be employed for development of the enterprise application. For Object-Oriented Analysis and Design (OOAD) methodology, the "4+1" views (Krutchen, 1995) consisting of *use case view, logical view, process view, development view* and *deployment view* are one of the most widely used forms of representation of detailed architecture. A tool (such as IBM Rational Software Architect or Borland Together Control Center Edition) is also often employed to model the architectural elements initially and later to continue elaboration into the design process.

10 b) Explain the principles in application server platform

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 application...

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.

...lication platforms are discussed in th... ...d other for...

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

Container pro... enterprise app... pooling. It provi... through its servic... the componen... or of componen... container is ess... Java classes (con... ntainer shown... that containe... Standard Editio...

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

Connectors ...geneous En... model all... municate ... itecture in