USN | | | | | | | | | | |

13MCA545

Fifth Semester MCA Degree Examination, Dec.2016/Jan.2017
Service Oriented Architecture

Time: 3 hrs. Max. Marks:100

Note: ***Answer any FIVE full questions.***

1	a.	List and explain common characteristics of contemporary SOA.	**(12 Marks)**
	b.	Which are the tangible benefits of SOA and explain?	**(08 Marks)**
2	a.	Discuss about the standards organizations and major vendor that contribute to SOA.	**(10 Marks)**
	b.	Which are the different architecture types? Explain distributed internet architecture and compare with SOA.	**(10 Marks)**
3	a.	Explain different types of web service roles.	**(10 Marks)**
	b.	Explain SOAP message frameworks and SOAP nodes.	**(10 Marks)**
4	a.	What is MEP? Explain various types of MEPs.	**(10 Marks)**
	b.	Explain atomic transactions in detail.	**(10 Marks)**
5	a.	What is choreography? Explain in detail.	**(10 Marks)**
	b.	List and explain common principles of service orientation in detail.	**(10 Marks)**
6	a.	Explain application service layer.	**(10 Marks)**
	b.	Explain business service layer and orchestration service layer.	**(10 Marks)**
7	a.	Explain WS-addressing language basics.	**(10 Marks)**
	b.	Explain software platforms for enterprise application.	**(10 Marks)**
8		Write short notes on the following:	
	a.	WSDL.	
	b.	XML.	
	c.	SOAP.	
	d.	Security.	**(20 Marks)**

* * * * *

Service Oriented Architecture – 13MCA545

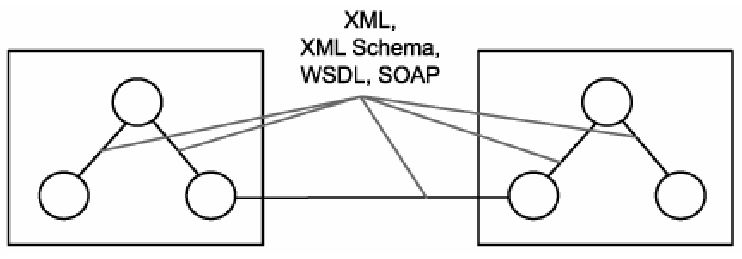
Dec. 2016/Jan 2017 Exam – Answer Key

1.a. List and Explain common characteristics of contemporary SOA

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.

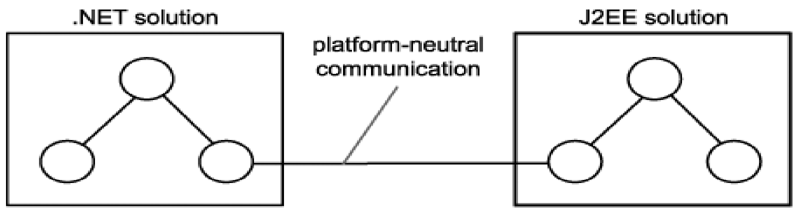
3.Contemporary SOA is fundamentally autonomous.
- The service-orientation principle of autonomy requires that
  - individual services be as independent and
  - self-contained as possible with respect to the control they maintain over their underlying logic.

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.



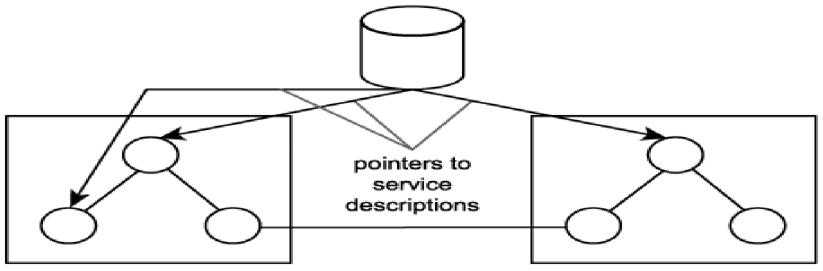
5.Contemporary SOA supports vendor diversity.

- 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.

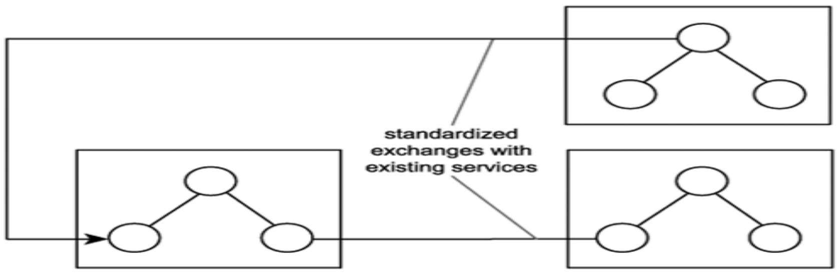
- SOA supports and encourages the advertisement and discovery of services throughout the enterprise and beyond.
- A serious SOA will likely rely on some form of service registry or directory to manage service descriptions



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

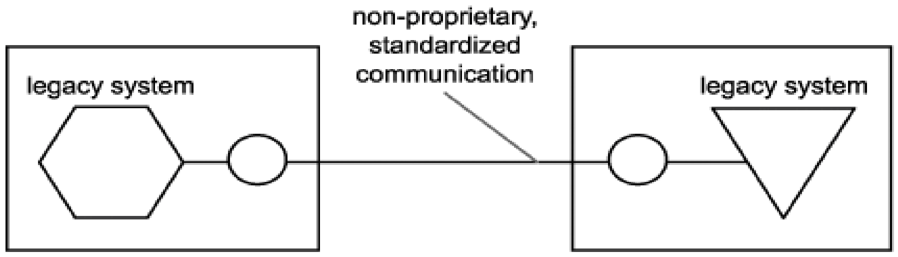
- To leveraging(maximizing) and supporting the
  - required usage of open standards,
  - a vendor diverse environment, and
  - the availability of a discovery mechanism

  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

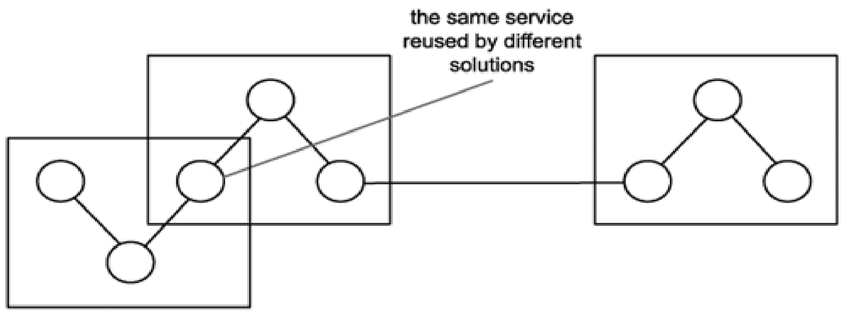


9.Contemporary SOA promotes architectural composability.

- Composability is a deep-rooted characteristic of SOA that can be realized on different levels.
- For example,
- By fostering the development and evolution of composable services, SOA supports the automation of flexible and highly adaptive business processes.

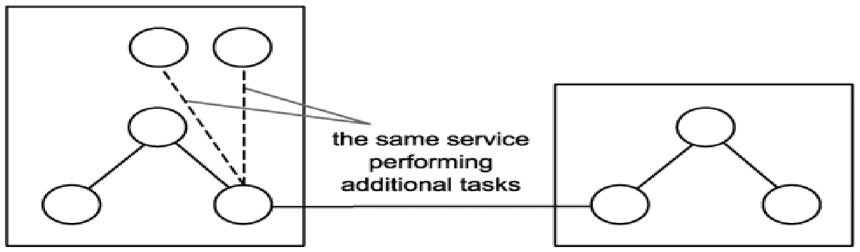
10.Contemporary SOA fosters inherent reusability.

- SOA establishes an environment that promotes reuse on many levels.
- For example, services designed according to service-orientation principles are encouraged to promote reuse, even if no immediate reuse requirements exist.
- Collections of services that form service compositions can themselves be reused by larger compositions.



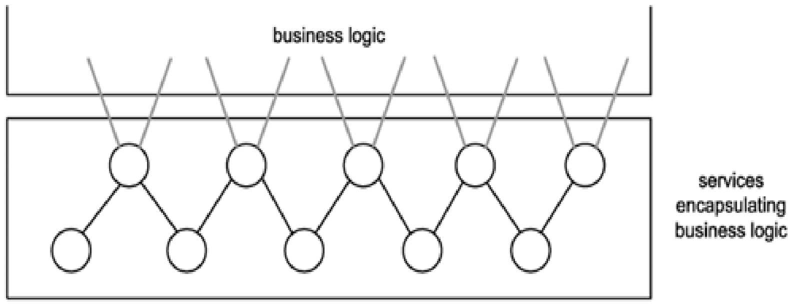
11.Contemporary SOA emphasizes extensibility.

- Extensibility is also a characteristic that is promoted throughout SOA as a whole.
- Extending entire solutions can be accomplished by adding services or by merging with other service-oriented applications (which also, effectively, "adds services").
- Because the loosely coupled relationship fostered among all services minimizes inter-service dependencies, extending logic can be achieved with significantly less impact.



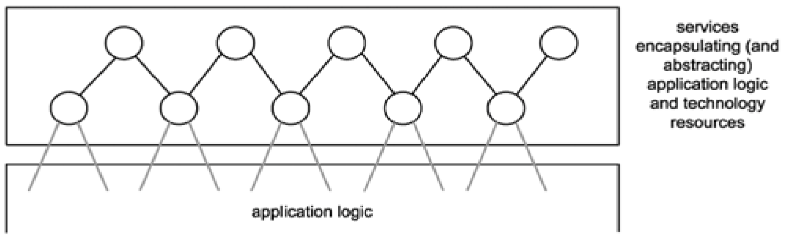
12.Contemporary SOA supports a service-oriented business modeling paradigm.

- Services can be designed to express business logic.
- BPM models, entity models, and other forms of business intelligence can be accurately represented through the coordinated composition of business-centric services.



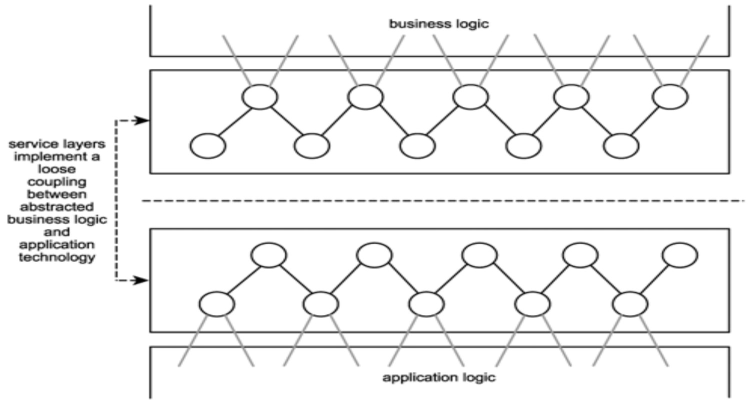
13.Contemporary SOA implements layers of abstraction.

- One of the characteristics that tends to evolve naturally through the application of service-oriented design principles is that of abstraction.
- Typical SOAs can introduce layers of abstraction by positioning services as the sole access points to a variety of resources and processing logic.


services encapsulating (and abstracting) application logic and technology resources
application logic

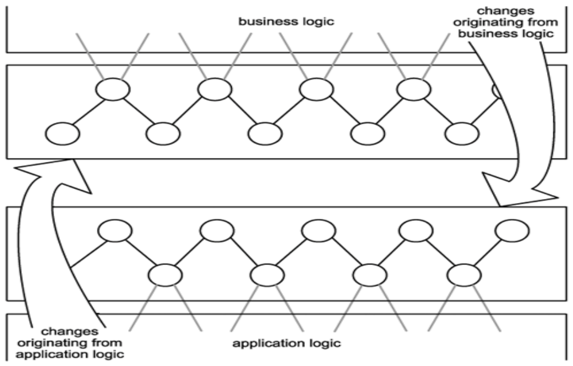
14.Contemporary SOA promotes loose coupling throughout the enterprise.

- core benefit to building a technical architecture with loosely coupled services is the resulting independence of service logic.
- Services only require an awareness of each other, allowing them to evolve independently.


business logic
service layers implement a loose coupling between abstracted business logic and application technology
application logic

15.Contemporary SOA promotes organizational agility(quickness).

- Change in an organization's business logic can impact the application technology that automates it.
- Change in an organization's application technology infrastructure can impact the business logic automated by this technology.
- The more dependencies that exist between these two parts of an enterprise, the greater the extent to which change imposes disruption and expense.


business logic
changes originating from business logic
changes originating from application logic
application logic

16. Contemporary SOA is a building block.

- A service-oriented application architecture will likely be one of several within an organization committed to SOA as the standard architectural platform.

- Organizations standardizing on SOA work toward an ideal known as the service-oriented enterprise (SOE), where all business processes are composed of and exist as services, both logically and physically

17.Contemporary SOA is an evolution(growth).

- SOA defines an architecture that is related to but still distinct from its predecessors.
- It differs from traditional client-server and distributed environments in that it is heavily influenced by the concepts and principles associated with service-orientation and Web services.
- It is similar to previous platforms in that it preserves the successful characteristics of its predecessors and builds upon them with distinct design patterns and a new technology set.

18.Contemporary SOA is still maturing(growing).

- The above characteristics far are fundamental to contemporary SOA, But this point is more relevant to SOA is at this moment.
- SOA is being positioned as the next standard application computing platform, but transition is not yet complete.
- Web services are being used to implement a great deal of application functionality
- But the support for a number of features necessary for enterprise-level computing is not yet fully available.

19.Contemporary SOA is an achievable ideal.

- A standardized enterprise-wide adoption of SOA is a state to which many organizations would like to fast-forward.
- In reality the process of transitioning to this state demands an enormous amount of effort, discipline, and, depending on the size of the organization, a good amount of time.
- Every technical environment will undergo changes during such a migration,
- Various parts of SOA will be phased in at different stages and to varying extents.
- It result in countless hybrid architectures, consisting mostly of distributed environments that are part legacy and part service-oriented.
- As companies adopt SOA during this evolution, many will need to retrofit(add) their environments (and their standards) to accommodate changes and innovations as SOA-related specifications, standards, and products continue to mature.

1.b Tangible benfits of SOA and Explain

Improved integration (and intrinsic interoperability)

 SOA creates solutions that consist of essentially interoperable services.

 A cross-application integration project into less of a custom development effort, and more of a modeling exercise.

 The bottom line: The cost and effort of cross-application integration is significantly lowered when applications being integrated are SOA-compliant.

Inherent reuse

 Service-orientation promotes the design of services that are inherently reusable.

 Designing services to support reuse from the get-go opens the door to increased opportunities for leveraging (force) existing automation logic.

Streamlined architectures and solutions

 The concept of composition is another fundamental part of SOA. .

 These are extensions to the basic Web services framework established by first-generation standards represented by WSDL(Web Service Description Language), SOAP(Small Object Access Protocol), and UDDI (Universal Description, Discovery, and Integration)

Leveraging the legacy investment

 Web services technology set has generated a large adapter market, enabling many legacy environments to participate in service-oriented integration architectures.

 This allows IT departments to work toward a state of federation, where previously isolated environments now can interoperate without requiring the development of expensive and sometimes fragile point-to-point integration channels.

 Still riddled with risks relating mostly

o to how legacy back-ends must cope with increased usage volumes,

o the ability to use what you already have with service-oriented solutions that you are building now and in the future is extremely attractive.

Establishing standardized XML data representation

 SOA is built upon and driven by XML. An adoption of SOA leads to the opportunity to fully leverage the XML data representation platform.

 A standardized data representation format (once fully established) can reduce the complexity of application environments.

Focused investment on communications infrastructure

 Web services establish acommon communications framework,

 SOA can centralize inter-application and intra-application communication as part of standard IT infrastructure.

 This allows organizations to evolve enterprise-wide infrastructure by investing in a single technology set responsible for communication.

"Best-of-breed" alternatives

 A key feature of service-oriented enterprise environments is the support of "best-of-breed" technology.

 Because SOA establishes a vendor-neutral communications framework, it frees IT departments from being chained to a single proprietary development and/or middleware platform.

 For any given piece of automation that can expose an adequate service interface, you now have a choice as to how you want to build the service that implements it.

Organizational agility

 Agility is a quality inherent in just about any aspect of the enterprise.

 All parts contain a measure of agility related to how they are constructed, positioned, and leveraged.

 Regardless of what parts of service-oriented environments are leveraged, the increased agility with which IT can respond to business process or technology-related changes is significant.

 The bottom line: The cost and effort to respond and adapt to business or technology-related change is reduced.

2.a. Discuss about the standards organizations and major ventor that contribute to SOA

Standards organizations that contribute to SOA

- Standards are produced, though, is not always that clear.
- Internet standards organizations have existed for some time now, but their respective agendas are not always distinct and sometimes even overlap.
- Microsoft, IBM, Sun Microsystems, and many others have played ly significant roles in formalizing Web services specifications, and accelerating the implementation of these specifications as industry standards.
- Let's first learn more about the three most prominent standards organizations.
- Collectively, they are responsible for seeing through the evolution of XML and Web services architectures.

The World Wide Web Consortium – (W3C)

- Founded by Tim Berners-Lee in 1994
- W3C has been hugely responsible for furthering the World Wide Web as a global, semantic medium for information sharing.
- First released HTML, one of the most popular technical languages
- Increased in eBusiness, the W3C responded by producing key foundation standards based on XML, such as XML Schema and XSLT.
- Four separate working groups made significant contributions to W3C Web Services Activity projects
- They developed of important base standards for Web services.
- First-most are the SOAP and WSDL standards,
- Recently, the W3C has produced the Web Services Choreography Description Language (WS-CDL) - A specification that governs standardized inter-service exchange patterns.
- Web Services Architecture document - it remains a reference point.
- The W3C is known for its formal and accurate approach to standards development.

- Specifications be subjected to numerous review and revision stages, with each new version being published to their public Web site.
- Standards can take two to three years to be completed.

Organization for the Advancement of Structured Information Standards (OASIS)

- Established in 1993 as the SGML, 5 years later changed to OASIS (SGML to XML-related standards)
- 1000 members from 600 organizations,
- International standards producing organization.
- OASIS a
- WS-BPEL specification
- ebXML (a specification that aims to establish a standardized means of B2B data interchange)
- UDDI specification (Core std for First generation web service)
- XML and Web services security extensions.
- Security Assertion Markup Language (SAML)
- Extensible Access Control Markup Language (XACML) provide important features in the areas of single sign-on and authorization.
- Web Services Security (WSS) technical committee - Security-related project. Further developing and realizing the important WS-Security framework.
- W3C focuses on establishing core, industry-agnostic standards
- OASIS group's primary interests lie in leveraging these standards to produce additional specifications that support various vertical industries.

The Web Services Interoperability Organization (WS-I)

- Object of WS-I is open interoperability
- Established in 2002, 200 organizations, including all major SOA vendors.
- Releasing the Basic Profile, a recommendation-based document that contains available standards collectively used in interoperability architec.
- WSDL, SOAP, UDDI, XML, and XML Schema, the Basic Profile has become an important document within the IT community.
- Developed the Basic Security Profile. (most important collection of Web services and XML security technologies.
- It continue releasing Profiles for each major aspect of Web services-related interoperability, reliable messaging, Web service management, and orchestration.
- Profiles also supplement
- sample implementations and
- best practices on how the standards are to be used together to achieve a quality level of interoperability.
- Provides a series of testing tools that can be used to ensure compliance with Profiles.
- Validity checkers that use Basic Profile conformance as part of the validation criteria.

- Membership includes significant SOA vendors, no one company has more power than another, regardless of its size or market share.
- W3C recently rejected an invitation to become an associate member of the WS-I
- Working group members from the WS-I continue to contribute to W3C and OASIS initiatives by directly participating in their respective working groups.
- The role of these WS-I representatives is to provide continual feedback relating to interoperability issues.

4.2.3. Major vendors that contribute to SOA

- Standards organizations have their own culture and philosophies around how standards should be developed, they are all heavily influenced by the commercial market.
- Vendors supply a significant portion of the contributors that actually end up developing the standards.
- Some of the companies that have participated in the standards development processes include Microsoft, IBM, BEA Systems, Sun Microsystems, Oracle, Tibco, Hewlett-Packard, Canon, Commerce One, Fujitsu, Software AG, Nortel, Verisign, and WebMethods.

The vendor influence

- IBM has laid out a technology path for increasing support of SOA within its WebSphere platform.
- Microsoft increasing SOA features within the .NET technology framework, and building Web services technology for Windows
- Web services non-proprietary, a vendor who can help shape a standard might be motivated to do so with proprietary technology considerations in mind.
- Challenge - getting *all vendors to agree on how one* standard should be designed.

Vendor alliances

- Battle between most established vendor leads distrust.
- Collaborate on specifications (interoperability) between vendor platforms turn into obstacles.
- Forming an alliance allows vendors to join forces in order to attain common goals.
- Lifespan of an alliance is based on d the development cycle of a specification.
- Most noticeable team of repeat-collaborators (IBM, Microsoft,
- and BEA)
- Persisted their working relationship to push forward a series of WS-* extensions.
- One of the more talked about examples of alliances playing a significant role in standards development is the creation of the

Choosing a standards organization

- Choice of standards organization can have implications.
- Standards development arena is directly related to market demand.
- Vendors have market-driven goals fueled by pressures to deliver product releases that meet customer demands and match
- Given that the W3C relies on a longer standards development process, it is tempting for vendors to submit their standards to OASIS instead.
- Organizations develop similar specifications may seem redundant; one always seems to rise to the top.
- And despite the fact that opposing motives may seem counter-productive to fostering a collection of platform-neutral technology standards, the quality of what's been delivered so far has been adequate for furthering the cause of SOA.

2.b Which are the different architecture types? Explain distributed internet architecture and compare with SOA

Application Architecture

- With the rise of multi-tier applications, the variations with which applications could be delivered began to increase
- A definition of a baseline definition application becomes important
- The definition is
    - abstract in nature,
    - but specifically explained the technology,
    - boundaries,
    - rules,
    - limitations, and
    - design characteristics that apply to all solutions - *application architecture.*
- An application architecture is a blueprint
- Different levels can be specified, depending on the organization Some keep it
    - high-level,
    - providing abstract physical and logical representations of the technical blueprint.
- Some more detail, such as
    - common data models,
    - communication flow diagrams,
    - application-wide security requirements, and
    - aspects of infrastructure.
- An organization that houses both .NET and J2EE solutions
- Having separate application architecture specifications for each.
- Key part - it should reflect immediate solution requirements, as well as long-term, strategic IT goals.

Enterprise Architecture

- In larger IT, Different application architectures co-exist and even integrate, the demands on the underlying hosting platforms can be complex.

- Master specification to be created, providing a high-level overview of all forms of heterogeneity that exist within an enterprise, as well as a definition of the supporting infrastructure.
- Enterprise architecture specification - what an urban plan is to a city.
- Relationship between an urban plan and the blueprint of a building are comparable to that of enterprise and application architecture specifications.
- Changes to enterprise architectures directly affect application architectures
- Architecture specifications often are maintained by the same group of individuals.
- EA contains a long-term vision of how the organization plans to evolve its technology and environments.
- For example, the goal of phasing out an outdated technology platform may be established in this specification.

Service Oriented Architecture

- Service-oriented architecture extents both enterprise and application architecture domains.
- The benefit offered by SOA can be realized when applied across multiple solution environments.
- Building reusable and interoperable services based on a vendor-neutral communications platform can fully be leveraged.
- This does not mean that the entire enterprise must become service-oriented. SOA belongs in those areas that have the most to gain from the features and characteristics it introduces.
- "SOA" does not imply a particular architectural scope.
- SOA refer to an application architecture or the approach used to standardize technical architecture across the enterprise.
- Because of the composable nature of SOA, it is absolutely possible for an organization to have more than one SOA.
- Web services platform offers one of a number of available forms of implementation for SOA.

Distributed Internet architecture

 Distributing application logic among multiple components (some residing on the client, others on the server) reduced deployment headaches by centralizing a greater amount of the logic on servers.

 Server-side components, here located on dedicated application servers, would then share and manage pools of database connections, alleviating the burden of concurrent usage on the database server .A single connection could easily facilitate multiple users

 These benefits came at the cost of increased complexity and ended up shifting expense and effort from deployment issues to development and administration processes.

 Building components capable of processing multiple, concurrent requests was more difficult and problem-ridden than developing a straight-forward executable intended for a single user.

 It replaces the client-server database connections with the client-server remote procedure call (RPC) connection.

 RPC technologies such as CORBA and DCOM allowed for remote communication between components residing on client workstations and servers.

- This increases maintenance effort resulting from the introduction of the middleware layer. For example, application servers and transaction monitors required significant attention in larger environments.

Application logic

- Distributed Internet applications place all of their application logic on the server side. Even client-side scripts intended to execute in response to events on a Web page are downloaded from the Web server upon the initial HTTP request. With none of the logic existing on the client workstation, the entire solution is centralized.
- The emphasis is therefore on how application logic should be partitioned where the partitioned units of processing logic should reside
- Service-oriented architecture is very similar to distributed Internet architecture. Provider logic resides on the server end where it is broken down into separate units. The differences lie in the principles used to determine the three primary design considerations just listed.
- Traditional distributed applications consist of a series of components that reside on one or more application servers. Components are designed with varying degrees of functional granularity, depending on the tasks they execute

❖ Components residing on the same server communicate via proprietary APIs, as per the public interfaces they expose. RPC protocols are used to accomplish the same communication across server boundaries.

- This is made possible through the use of local proxy stubs that represent components in remote locations
- Contemporary SOAs still employ and rely on components. However, the entire modeling approach now takes into consideration the creation of services that encapsulate some or all of these components.
- These services are designed according to service-orientation principles and are strategically positioned to expose specific sets of functionality.
- . The standardized interface supports the open communications framework that sits at the core of SOA. When properly designed, loosely coupled services support a composition model, allowing individual services to participate in aggregate assemblies. This introduces continual opportunities for reuse and extensibility.
- Another significant shift related to the design and behavior of distributed application logic is in how services exchange information.
- While traditional components provide methods that, once invoked, send and receive parameter data, Web services communicate with SOAP messages.
- Reuse is also commonly emphasized in traditional distributed design approaches, SOA fosters reuse and cross-application interoperability on a deep level by promoting the creation of solution-agnostic services.

Application processing

- Distributed Internet architecture promotes the use of proprietary communication protocols, such as DCOM and vendor implementations of CORBA for remote data exchange.
- These technologies historically have had challenges, they are considered relatively efficient and reliable, especially once an active connection is made. They can support the creation of stateful and stateless components that primarily interact with synchronous data exchanges (asynchronous communication is supported by some platforms but not commonly used).

- SOA, on the other hand, relies on message-based communication. This involves the serialization, transmission, and deserialization of SOAP messages containing XML document payloads.
- Processing steps can involve the conversion of relational data into an XML-compliant structure, the validation of the XML document prior and subsequent to transmission, and the parsing of the document and extraction of the data by the recipient.
- Network of SOAP servers can effectively replace RPC-style communication channels within service-oriented application environments, the incurred processing overhead becomes a significant design issue.
- Document and message modeling conventions and the strategic placement of validation logic are important factors that shape the transport layer of service-oriented architecture.

. ***Technology***

- The technology behind distributed Internet architecture consisted of components, server-side scripts, and raw Web technologies, such as HTML and HTTP.
- The subsequent availability of Web services allowed distributed Internet applications to cross proprietary platform boundaries.
- Distributed applications use XML and Web services, there may be little difference between the technology behind these solutions and those based on SOA. One clear distinction, though, is that a contemporary SOA will most likely be built upon XML data representation and the Web services technology platform.
- Thus XML and Web services are optional for distributed Internet architecture but not for contemporary SOA.

Security

- When application logic is strewn across multiple physical boundaries, implementing fundamental security measures such as authentication and authorization becomes more difficult.

In a two-tiered client-server environment, an exclusive server-side connection easily facilitates the identification of users and the safe transportation of corporate data.

- Connection is removed, and when data is required to travel across different physical layers, new approaches to security are needed.
- To ensure the safe transportation of information and the recognition of user credentials, while preserving the original security context, traditional security architectures incorporate approaches such as delegation and impersonation.
- Encryption also is added to the otherwise wide open HTTP protocol to allow data to be protected during transmission beyond the Web server.
- **In SOA security** is incorporated and applied by the WS-Security framework, the security models used within SOA emphasize the placement of security logic onto the messaging level.
- SOAP messages provide header blocks in which security logic can be stored. That way, wherever the message goes, so does its security information. This approach is required to preserve individual autonomy and loose coupling between services, as well as the extent to which a service can remain fully stateless
- In a two-tiered client-server environment, an exclusive server-side connection easily facilitates the identification of users and the safe transportation of corporate data.
- Connection is removed, and when data is required to travel across different physical layers, new approaches to security are needed.

□ □To ensure the safe transportation of information and the recognition of user credentials, while preserving the original security context, traditional security architectures incorporate approaches such as delegation and impersonation.
□ □Encryption also is added to the otherwise wide open HTTP protocol to allow data to be protected during transmission beyond the Web server.
□ □**In SOA security** is incorporated and applied by the WS-Security framework, the security models used within SOA emphasize the placement of security logic onto the messaging level.
□ □SOAP messages provide header blocks in which security logic can be stored. That way, wherever the message goes, so does its security information. This approach is required to preserve individual autonomy and loose coupling between services, as well as the extent to which a service can remain fully stateless.

Administration

□ □Maintaining component-based applications involves keeping track of individual component instances, tracing local and remote communication problems, monitoring server resource demands, and, of course, the standard database administration tasks
□ □Distributed Internet architecture further introduces the Web server and with it an additional physical environment that requires attention while solutions are in operation. Because clients, whether local or external to an organization, connect to these solutions using HTTP, the Web server becomes the official first point of contact.
□ □It must therefore be designed for scalability requirement that has led to the creation of Web server farms that pool resources.
□ □SOAs typically require additional runtime administration. Problems with messaging frameworks can more easily go undetected than with RPC-based data exchanges.
□ □This is because so many variations exist as to how messages can be interchanged. RPC communication generally requires a response from the initiating component, indicating success or failure.
□ □Upon encountering a failure condition, an exception handling routine kicks in. Exception handling with messaging frameworks can be more complex and less robust.
□ □Other maintenance tasks, such as resource management are also required. However, to best foster reuse and composability, a useful part of an administration infrastructure for enterprises building large amounts of Web services is a private registry.

3.a. Explain different types of web services roles

5.2.1. Service roles

- A Web service is capable of assuming different roles, depending on the context within which it is used.
- For example, a service can act as the initiator, relayer, or the recipient of a message.
- A service is therefore not labeled exclusively as a client or server, but instead as a unit of software capable of altering its role, depending on its processing responsibility in a given scenario.
- Provided here are descriptions of the fundamental service roles.

Service provider

The *service provider* role is assumed by a Web service under the following conditions:

- The Web service is invoked via an external source, such as a service requestor
- The Web service provides a published service description offering information about its features and behavior.
- The service provider takes the role of server.
-
- Depending on the type of message exchange used when invoking a service provider,
- The service provider may reply to a request message with a response message.

Classified into

- *service provider entity* (the organization or individual providing the Web service)
- *service provider agent* (the Web service itself, acting as an agent on behalf of its owner)

Service requestor

- Any unit of processing logic capable of issuing a request message that can be understood by the service provider is classified as a *service requestor*.
- A Web service is always a service provider but also can act as a service requestor.
- A Web service takes on the service requestor role under the following circumstances:
- The Web service invokes a service provider by sending it a message
- The Web service searches for and assesses the most suitable service provider by studying available service descriptions.
- The service requestor takes the role of client.
- It is important to note that a service provider is not acting as a service requestor when it sends a message in response to a request message.

A service requestor can represent both the Web service itself as well as the Web service owner

- *service requestor entity* (the organization or individual requesting the Web service)
- *service requestor agent* (the Web service itself, acting as an agent on behalf of its owner)

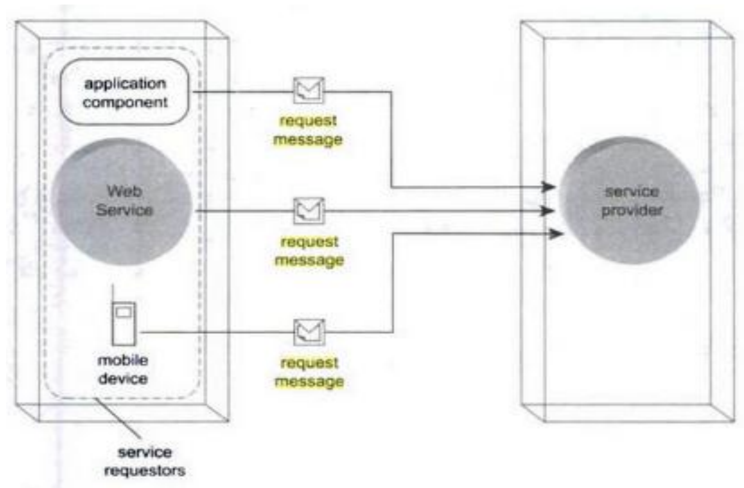


Figure 5.3
The sender of the request message is classified as a service requestor.

Intermediaries

- The communications framework established by Web services different from traditional point-to-point communications channels.
- Point-to-point communication was a great deal easier to design since less flexible and less scalable
- Web services communication is based on the use of messaging paths, which can best be described as point-to-* paths.
- Once a service provider submits a message, it can be processed by multiple intermediate routing and processing agents before it arrives at its ultimate destination.

Web services and service agents that route and process a message after it is initially sent and before it arrives at its ultimate destination are referred to as *intermediaries* or *intermediary services*

There are two types of intermediaries.

1. Passive intermediary
2. Active Intermediary

P*assive intermediary* (Does not modify the message)

- Responsible for routing messages to a subsequent location.
- It may use information in the SOAP message header to determine the routing path, or it may employ native routing logic to achieve some level of load balancing.

Active intermediaries

- It is used route messages to a forwarding destination.
- Alter the message contents
- It will look for particular SOAP header blocks and perform some action in response to the information they find there.
- Alter data in header blocks and may insert or even delete header blocks entirely.

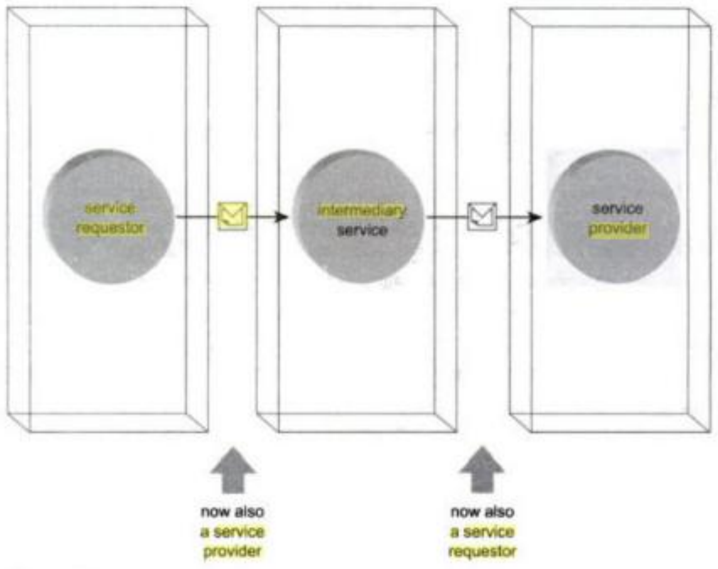


Figure 5.5

The intermediary service transitions through service provider and service requestor roles while processing a message.

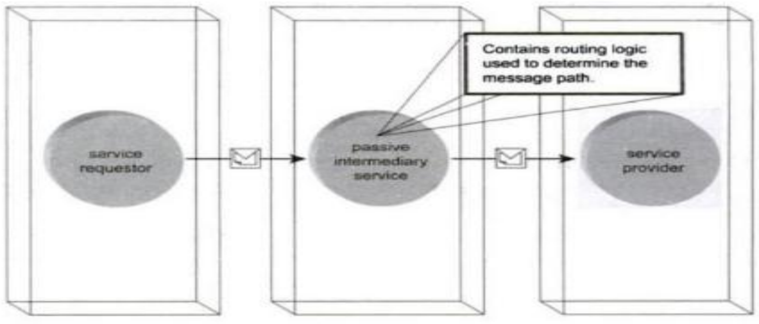

Contains routing logic used to determine the message path.
service requestor
passive intermediary service
service provider

Figure 5.6

A passive intermediary service processing a message without altering its contents.

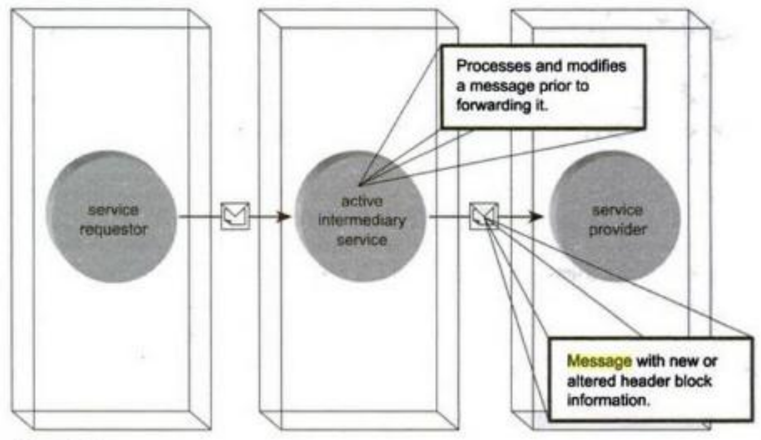

Processes and modifies a message prior to forwarding it.
service requestor
active intermediary service
service provider
Message with new or altered header block information.

Figure 5.7
An active intermediary service.

Initial sender and ultimate receiver

- *Initial senders* are service requestors that initiate the transmission of a message.
- Always the first Web service in a message path.
- The counterpart to this role is the *ultimate receiver*.
- Service providers that exist as the last Web service along a message's path

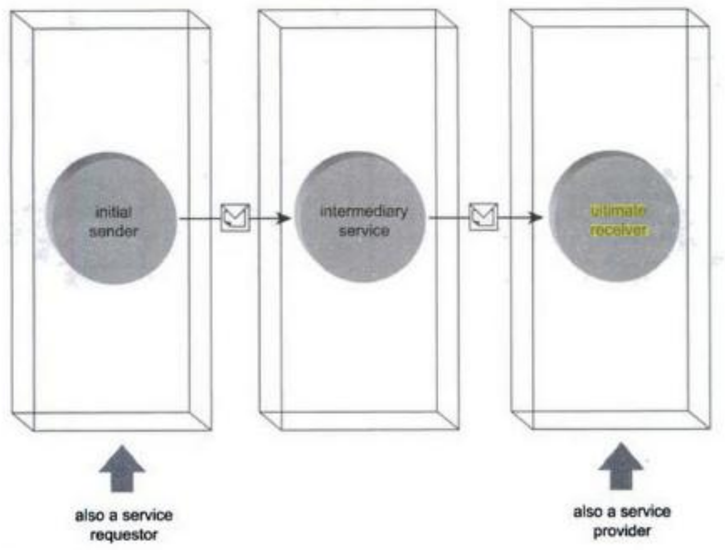


Figure 5.8
Web services acting as initial sender and ultimate receiver.

Service compositions

- Composite relationship between a collection of services.
- Any service can enlist one or more additional services to complete a given task.
- Any of the enlisted services can call other services to complete a given sub-task.

- Service that participates in a composition assumes an individual role of *service composition member*
- Some Web services to be designed in such a manner that they can be pulled into future service compositions without a foreknowledge of how they will be utilized.

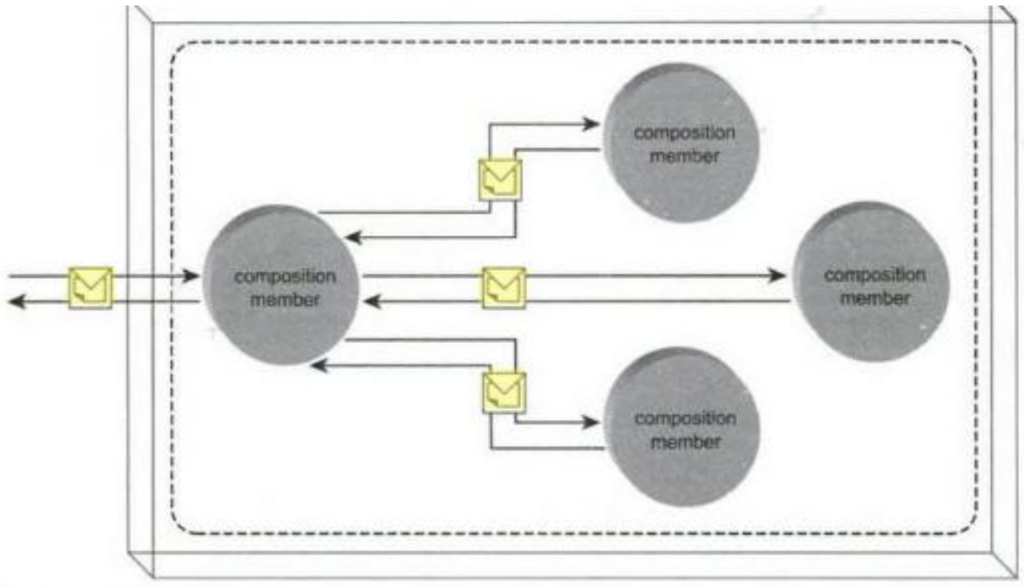


Figure 5.10
A service composition consisting of four members.

3.b.Explain SOAP message frameworks and SOAP nodes

5.4. Messaging (with SOAP)

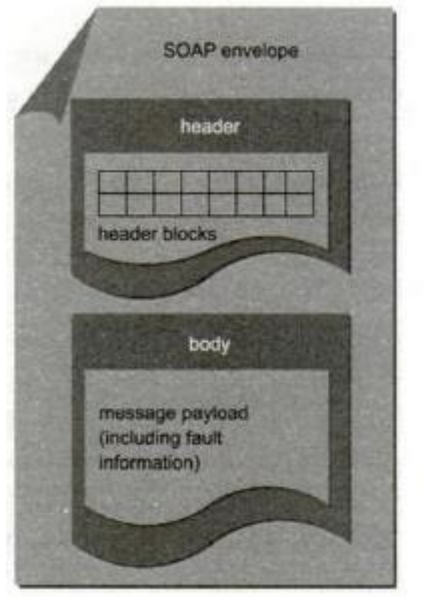
- All communication between services is message-based,
- The messaging framework chosen must be standardized so that all services, regardless of origin, use the same format and transport protocol.
- Message-centric application design that an increasing amount of business and application logic is embedded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

5.4.2. Nodes

The programs that services use to transmit and receive SOAP messages are referred to as *SOAP nodes.*

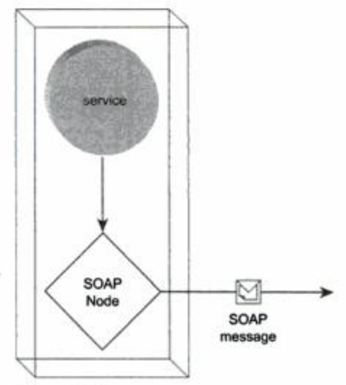


Figure 5.23
A SOAP node transmitting a SOAP message received by the service logic.

- Regardless of how they are implemented, SOAP nodes must conform to the processing standard set forth in the versions of the SOAP specification they support.
- Vendor-neutral communications framework upon which SOA is based on the SOAP node.
- It is what guarantees that a SOAP message sent by the SOAP node from service A can be received and processed by a SOAP node from any other service.

Node types

- *SOAP nodes* are given labels that identify their type, depending on what form of processing they are involved with in a given message processing scenario.
- Below is a list of type labels associated with SOAP nodes
- The SOAP specification has a different use for the term "role" and instead refers to these SOAP types or labels as *concepts*.
    - *SOAP sender*a SOAP node that transmits a message
    - *SOAP receiver*a SOAP node that receives a message
    - *SOAP intermediary*a SOAP node that receives and transmits a message, and optionally processes the message prior to transmission
    - *initial SOAP sender*the first SOAP node to transmit a message
    - *ultimate SOAP receiver*the last SOAP node to receive a message

SOAP intermediaries

Service intermediaries transition through service provider and service requestor roles, *SOAP intermediary* nodes move through SOAP receiver and SOAP sender types when processing a message

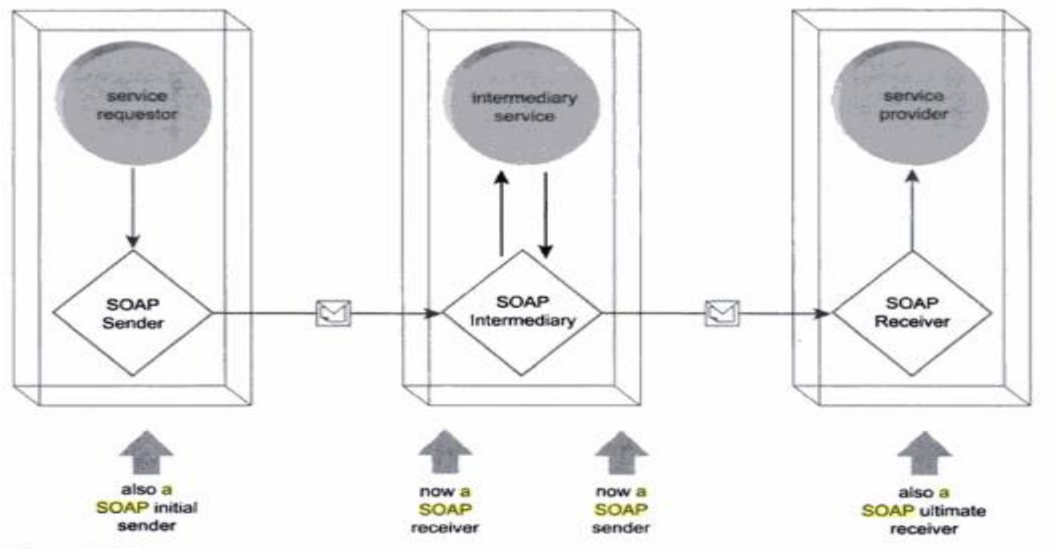


Figure 5.25
Different types of SOAP nodes involved with processing a message.

SOAP nodes acting as intermediaries can be classified as forwarding or active.

When a SOAP node acts as a *forwarding intermediary*, it is responsible for relaying the contents of a message to a subsequent SOAP node. In doing so, the intermediary will often process and alter header block information relating to the forwarding logic it is executing.

Active intermediary nodes are distinguished by the type of processing they perform above and beyond forwarding-related functions. An active intermediary is not required to limit its processing logic to the rules and instructions provided in the header blocks of a message it receives. It can alter existing header blocks, insert new ones, and execute a variety of supporting actions.

5.4.3. Message paths

A *message path* refers to the route taken by a message from when it is first sent until it arrives at its ultimate destination. Therefore, a message path consists of at least one initial sender, one ultimate receiver, and zero or more intermediaries (Figure 5.26). Mapping and modeling message paths becomes an increasingly important exercise in SOAs, as the amount of intermediary services tends to grow along with the expansion of a service-oriented solution. Design considerations relating to the path a message is required to travel often center around performance, security, context management, and reliable messaging concerns.

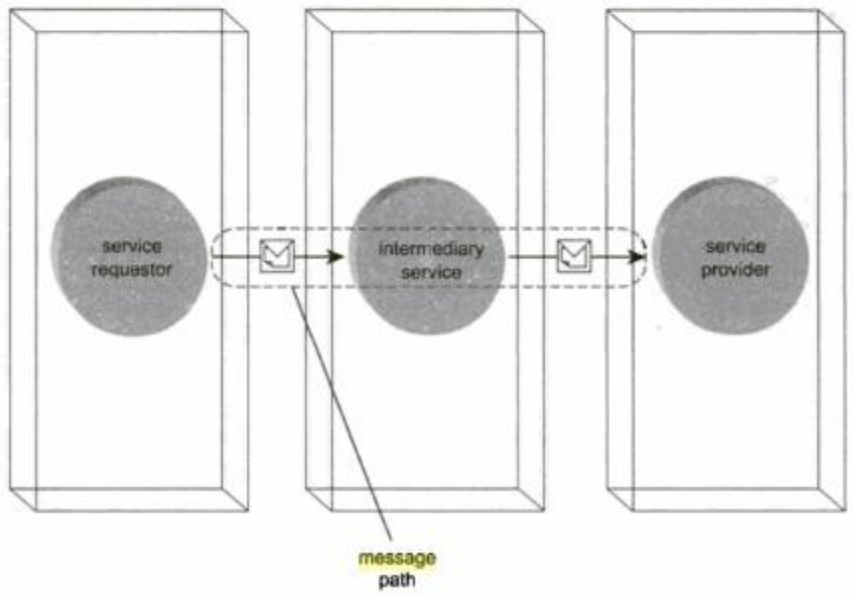


Figure 5.26
A message path consisting of three Web services.

Note also that a message path is sometimes not predetermined. The use of header blocks processed by intermediaries can dynamically determine the path of a message. This may be the result of routing logic, workflow logic, or environmental conditions

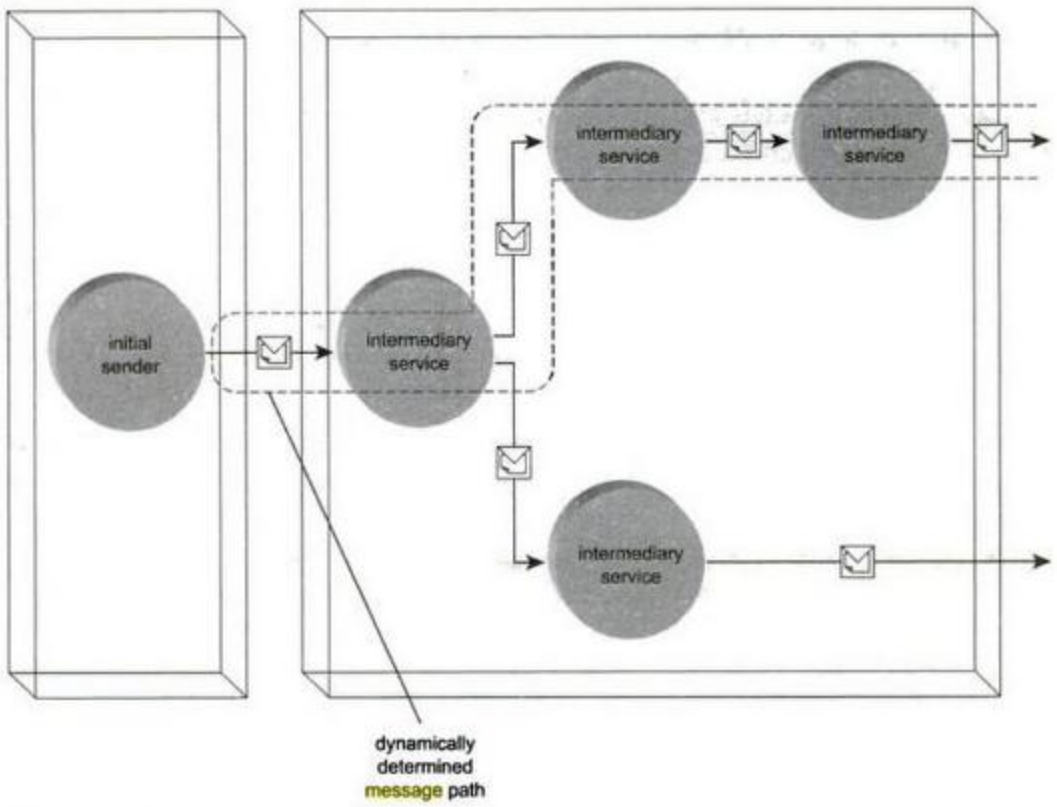
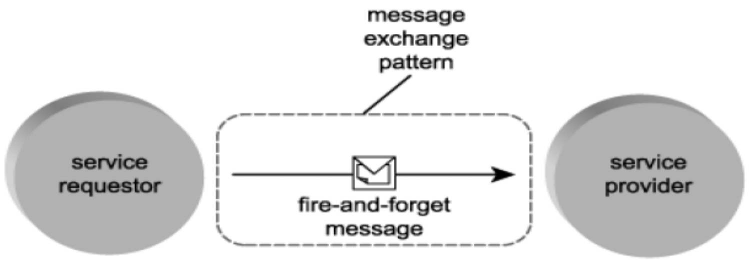


Figure 5.27
A message path determined at runtime.

When used within the context of SOAP nodes, this term is qualified and therefore referred to as a *SOAP message path*. While a message path in abstract can be purely logical, the SOAP node perspective is always focused on the actual physical transport route. A SOAP message path is comprised of a series of SOAP nodes, beginning with the initial SOAP sender and ending with the ultimate SOAP receiver. Every node refers to a physical installation of SOAP software, each with its own physical address.

4.a. What is MEP? Explain various types of MPS

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.

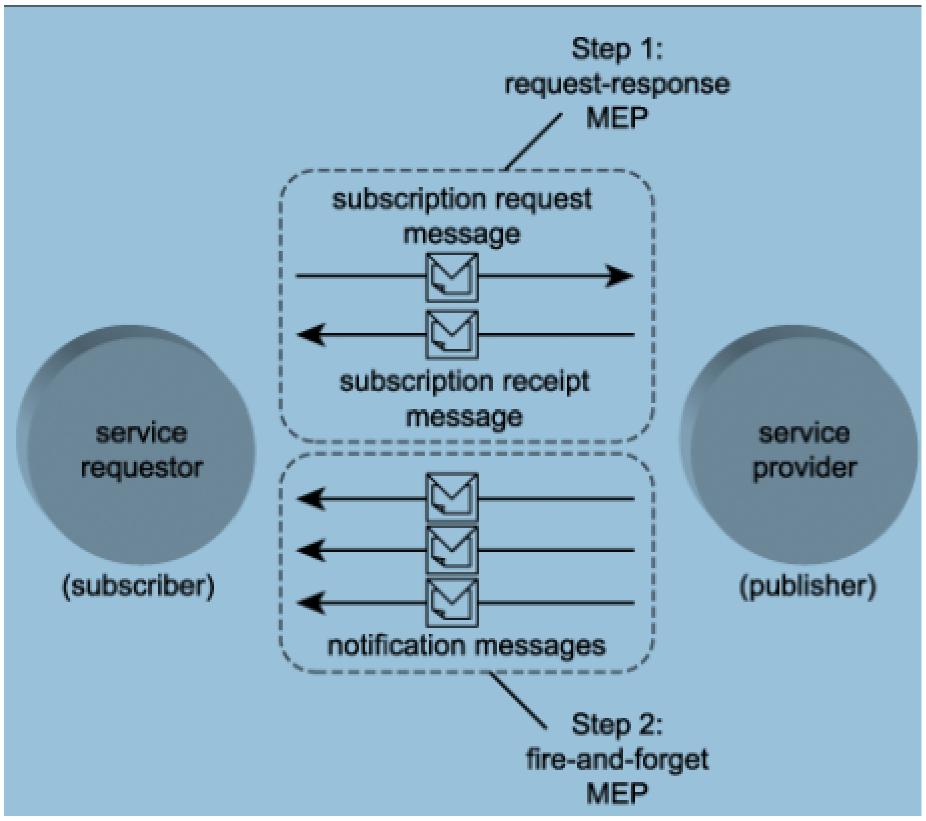


Complex MEPs

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



4.b Explain atomic transactions in detail

Atomic transactions : Transactions have been around for almost as long as automated computer solutions have existed. When managing certain types of corporate data, the need to wrap a series of changes into a single action is fundamental to many business process requirements. Atomic transactions implement the familiar commit and rollback features to enable cross-service transaction support

.

Atomic transactions apply an all-or-nothing requirement to work performed as part of an activity

ACID transactions

The protocols provided by the WS-AtomicTransaction specification enable cross-service transaction functionality comparable to the ACID-compliant transaction features found in most distributed application platforms.

For those of you who haven't yet worked with ACID transactions, let's quickly recap this important standard. The term "ACID" is an acronym representing the following four required characteristics of a traditional transaction:

Atomic Either all of the changes within the scope of the transaction succeed, or none of them succeed. This characteristic introduces the need for the rollback feature that is responsible for restoring any changes completed as part of a failed transaction to their original state.

Consistent None of the data changes made as a result of the transaction can violate the validity of any associated data models. Any violations result in a rollback of the transaction.

Isolated If multiple transactions occur concurrently, they may not interfere with each other. Each transaction must be guaranteed an isolated execution environment.

Durable Upon the completion of a successful transaction, changes made as a result of the transaction can survive subsequent failures.

Atomic transaction protocols :WS-AtomicTransaction is a coordination type, meaning that it is an extension created for use with the WS-Coordination context management framework we covered in the previous section. To participate in an atomic transaction, a service first receives a coordination context from the activation service. It can subsequently register for available atomic transaction protocols.

The following primary transaction protocols are provided:

A Completion protocol, which is typically used to initiate the commit or abort states of the transaction.

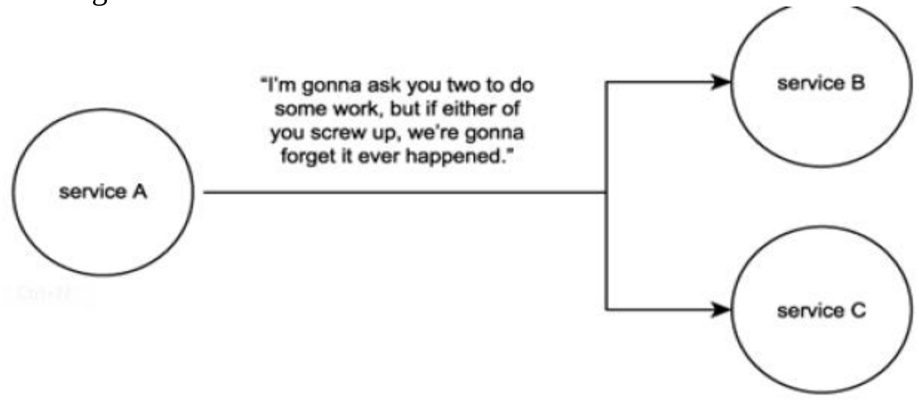
The Durable 2PC protocol for which services representing permanent data repositories should register.

The Volatile 2PC protocol to be used by services managing non-persistent (temporary) data.

Most often these protocols are used to enable a two-phase commit (2PC) that manages an atomic transaction across multiple service participants.

The atomic transaction coordinator

When WS-AtomicTransaction protocols are used, the coordinator controller service can be referred to as an atomic transaction coordinator. This particular implementation of the WS-Coordination coordinator service represents a specific service model. The atomic transaction coordinator plays a key role in managing the participants of the transaction process and in deciding the transaction's ultimate outcome.


"I'm gonna ask you two to do some work, but if either of you screw up, we're gonna forget it ever happened."
service A
service B
service C

5. a Explain in detail about Choreography in SOA
   (10)

Choreography

The Web Services Choreography Description Language (WS-CDL) is one of several specifications that attempts to organize information exchange between multiple organizations (or even multiple applications within organizations), with an emphasis on public collaboration

Figure 6.37. A choreography enables collaboration between its participants.

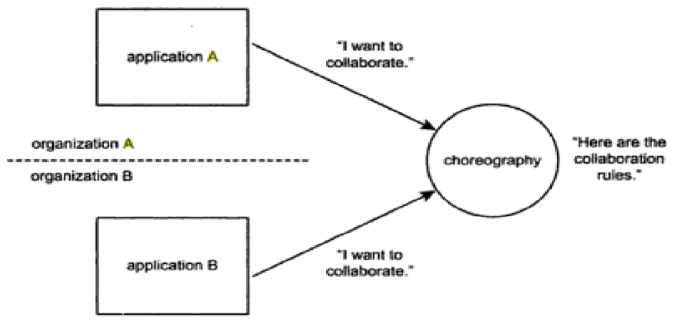


Figure 6.37
A choreography enables collaboration between its participants.

6.7.1. Collaboration

- An important characteristic of choreographies is that they are intended for public message exchanges.
- The goal is to establish a kind of organized collaboration between services representing different service entities, only no one entity (organization) necessarily controls the collaboration logic.
- Choreographies therefore provide the potential for establishing universal interoperability patterns for common inter-organization business tasks.

6.7.2. Roles and participants

- Within any given choreography, a Web service assumes one of a number of predefined *roles*.
- This establishes what the service does and what the service can do within the context of a particular business task.
- Roles can be bound to WSDL definitions, and those related are grouped accordingly, categorized as *participants* (services).

6.7.3. Relationships and channels

- Every action that is mapped out within a choreography can be broken down into a series of message exchanges between two services.
- Each potential exchange between two roles in a choreography is therefore defined individually as a *relationship*.
- Every relationship consequently consists of exactly two roles.

- *Channels* provides the means of establishing the nature of the conversation, by defining the characteristics of the message exchange between two specific roles.
- WS-CDL specification, as it fosters dynamic discovery and increases the number of potential participants within large-scale collaborative tasks.

6.7.4. Interactions and work units

- The actual logic behind a message exchange is encapsulated within an *interaction*.
- Interactions are the fundamental building blocks of choreographies because the completion of an interaction represents actual progress within a choreography.
- Related to interactions are *work units*.
- These impose rules and constraints that must be adhered to for an interaction to successfully complete

6.7.5. Reusability, composability, and modularity

- Each choreography can be designed in a reusable manner, allowing it to be applied to different business tasks comprised of the same fundamental actions.
- A choreography can be assembled from independent *modules*.
- These modules can represent distinct sub-tasks and can be reused by numerous different parent choreographies

Figure 6.38. A choreography composed of two smaller choreographies.

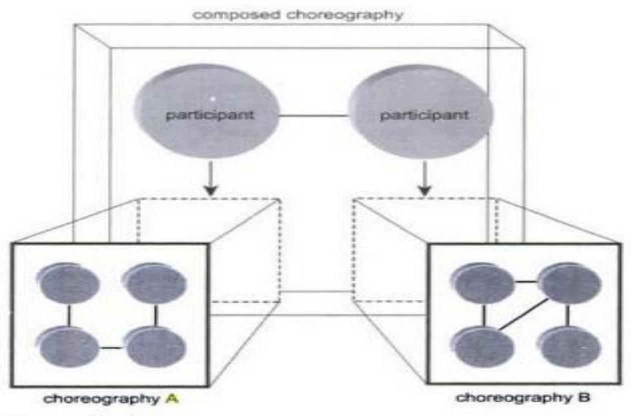


Figure 6.38
A choreography composed of two smaller choreographies.

6.7.6. Orchestrations and choreographies

- 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

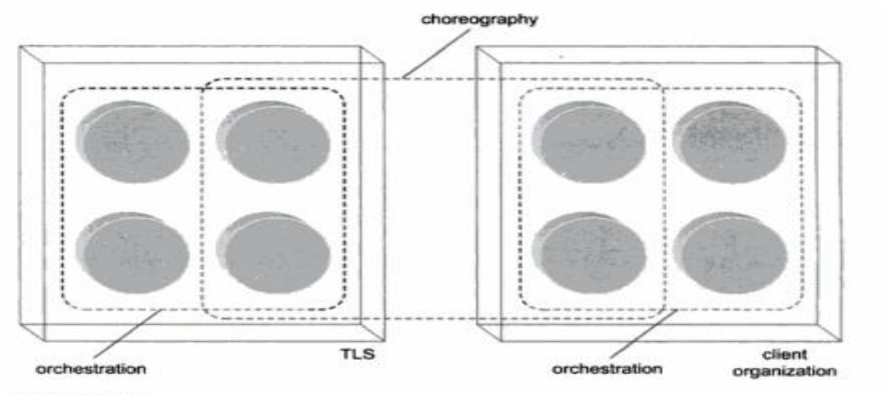


Figure 6.39
A choreography enabling collaboration between two different orchestrations.

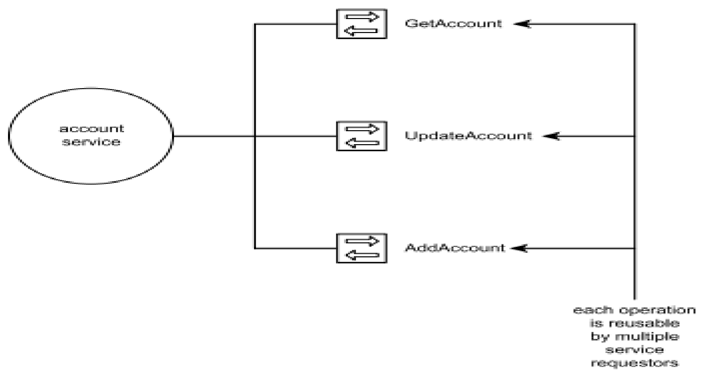
5. b Discuss in detail about the common principles of service orientation (10)

- Services are reusable— Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse.
- Services share a formal contract— For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange.
- Services are loosely coupled— Services must be designed to interact without the need for tight, cross-service dependencies.
- Services abstract underling logic— The only part of a service that is visible to the outside world is what is exposed via the service contract. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requestors.
- Services are composable— Services may compose other services. This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.

- Services are autonomous— The logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance.
- Services are stateless— Services should not be required to manage state information, as that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
- Services are discoverable—Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.

8.3.1 Services are reusable

Service-orientation encourages reuse in all services, regardless if immediate requirements for reuse exist. By applying design standards that make each service potentially reusable, the chances of being able to accommodate future requirements with less development effort are increased. Inherently reusable services also reduce the need for creating wrapper services that expose a generic interface over top of less reusable services.



8.3.2 Services share a formal contract

Service contracts provide a formal definition of:
• the service endpoint
• each service operation
• every input and output message supported by each operation • rules and characteristics of the service and its operations

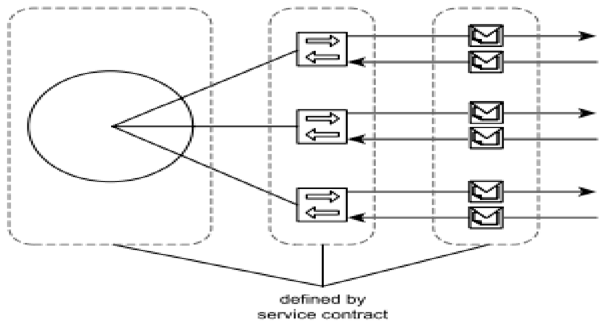


Figure 8.15
Service contracts formally define the service, operation, and message components of a service-oriented architecture.

Services are loosely coupled :

No one can predict how an IT environment will evolve. How automation solutions grow, integrate, or are replaced over time can never be accurately planned out because the requirements that drive these changes are almost always external to the IT environment. Being able to ultimately respond to unforeseen changes in an efficient manner is
a key goal of applying service-orientation

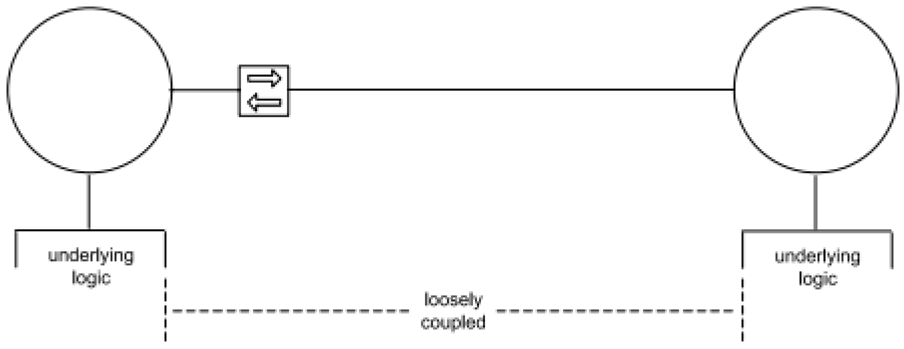


Figure 8.16
Services limit dependencies to the service contract, allowing underlying provider and requestor logic to remain loosely coupled.

Services abstract underlying logic

Also referred to as service interface-level abstraction, it is this principle that allows services
to act as black boxes, hiding their details from the outside world. The scope of logic represented by a service significantly influences the design of its operations and its position
within a process.

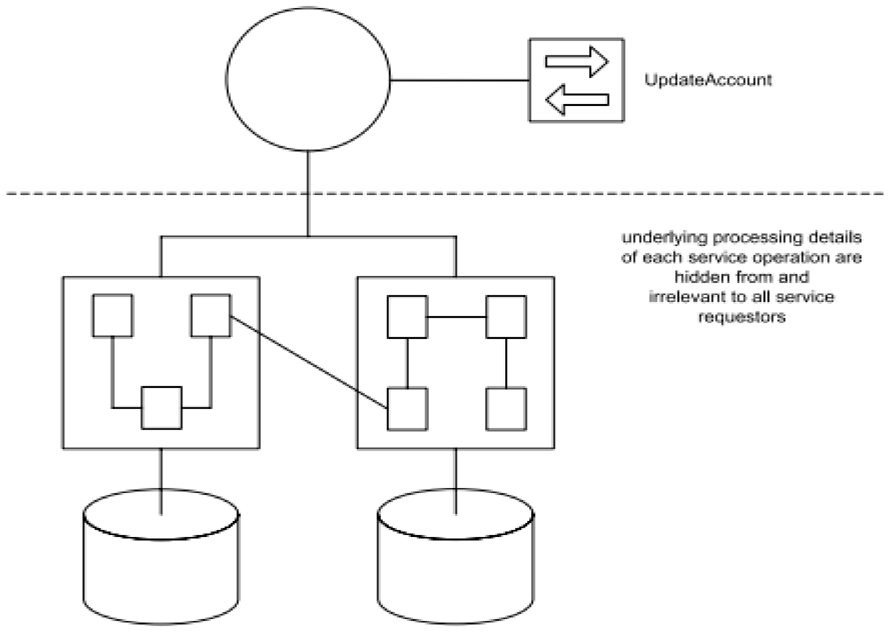

UpdateAccount
underlying processing details of each service operation are hidden from and irrelevant to all service requestors

Figure 8.17
Service operations abstract the underlying details of the functionality they expose.

Services are composable

A service can represent any range of logic from any types of sources, including other services. The main reason to implement this principle is to ensure that services are designed so that they can participate as effective members of other service compositions if ever required. This requirement is irrespective of whether the service itself composes others to accomplish its work

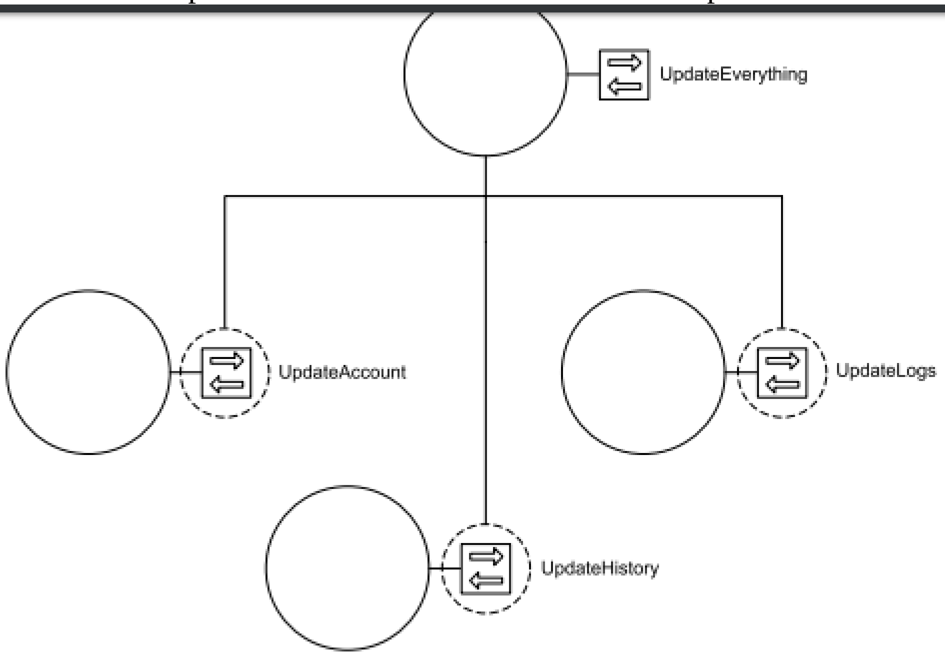


Figure 8.19
The UpdateEverything operation encapsulating a service composition.

Services are autonomous

Autonomy requires that the range of logic exposed by a service exist within an explicit boundary. This allows the service to execute self-governance of all its processing. It also eliminates dependencies on other services, which frees a service from ties that could inhibit its deployment and evolution (Figure 8.22). Service autonomy is a primary consideration when deciding how application logic should be divided up into services and which operations should be grouped together within a service context.

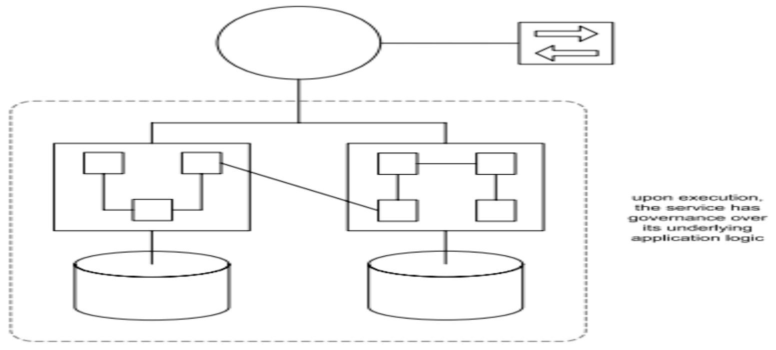

upon execution, the service has governance over its underlying application logic

Figure 8.22
Autonomous services have control over underlying resources.

Services are stateless

Services should minimize the amount of state information they manage and the duration for which they hold it. State information is data-specific to a current activity. While a service is processing a message, for example, it is temporarily stateful (Figure 8.24). If a service is responsible for retaining state for longer periods of time, its ability to remain available to other requestors will be impeded

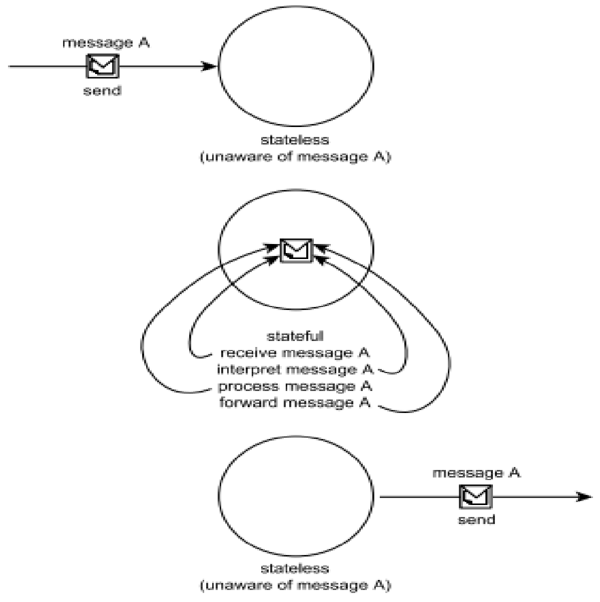


Figure 8.24
Stateless and stateful stages a service passes through while processing a message.

Services are discoverable

Discovery helps avoid the accidental creation of redundant services or services that implement redundant logic. Because each operation provides a potentially reusable piece of processing logic, metadata attached to a service needs to sufficiently describe not only the service's overall purpose, but also the functionality offered by its operations.

6.a. Explain the application service Layer

- 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

Typical examples of service models implemented as application services include the following: utility service

When a separate business service layer exists, there is a strong motivation to turn all application services into generic utility services. This way they are implemented in a solution-agnostic manner, providing reusable operations that can be composed by business services to fulfill business-centric processing requirements. Alternatively, if business logic does not reside in a separate layer, application services may be required to implement service models more associated with the business service layer. For example, a single application service also can be classified as a business service if it interacts directly with application logic and contains embedded business rules.

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. It is not a recommended design when building service abstraction layers

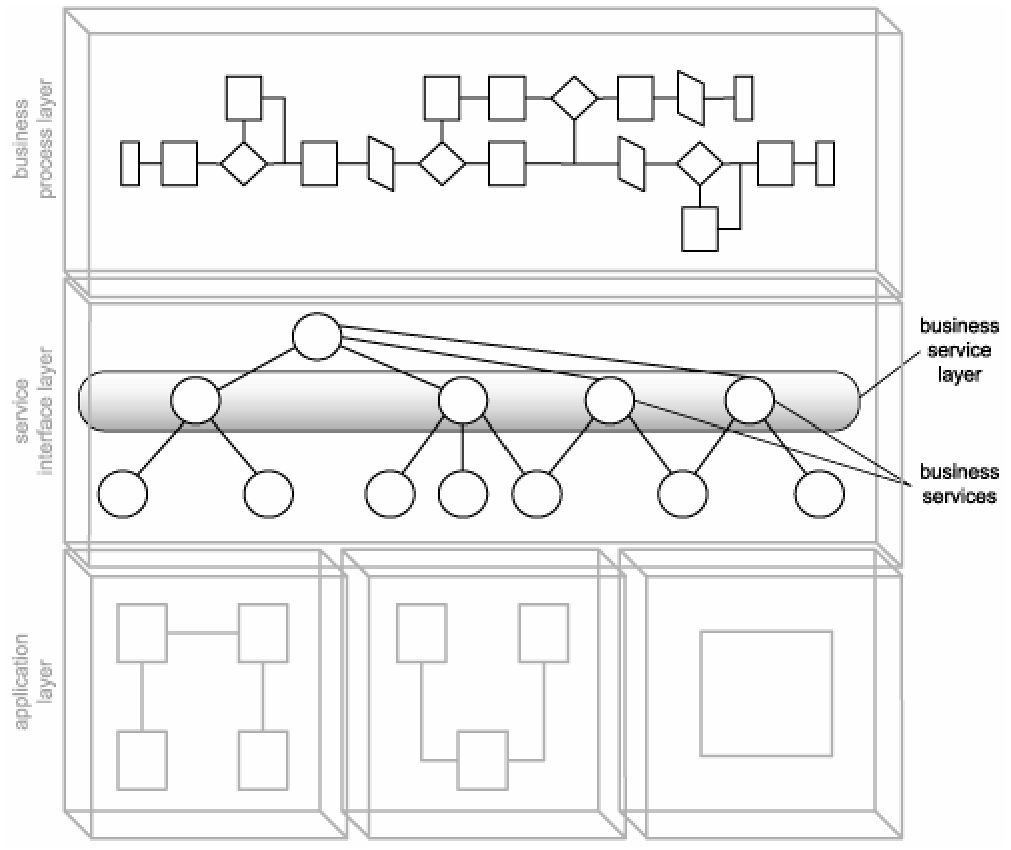
Finally, an application service also can compose other, smaller-grained application services (such as proxy services) into a unit of
coarse-grained application logic. Aggregating application services is frequently done to accommodate integration requirements.

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.
Because they are common residents of the application service layer, now is a good time to introduce the *wrapper service* model. 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. The most frequent form of wrapper service is a service adapter provided by legacy vendors. This type of out-of-the-box Web service simply establishes a vendor-defined service interface that expresses an underlying API to legacy logic.

Another variation of the wrapper service model not discussed in this book is the *proxy service*, also known as an *auto-generated WSDL*. This simply provides a WSDL definition that mirrors an existing component interface. It establishes an endpoint on the component's behalf, essentially allowing it to participate in SOAP communication. The proxy service should not be confused with a *service proxy*, which is used by service requestors to contact service providers

6.b. Explain the business service layer and orchestration 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).

Orchestration is more valuable to us than a standard business process, as it allows us to directly link process logic to service interaction within our workflow logic. This combines business process modeling with service-oriented modeling and design. And, because orchestration languages (such as WS-BPEL) realize workflow management through a process service model, orchestration brings the business process into the service layer, positioning it as a master composition controller.

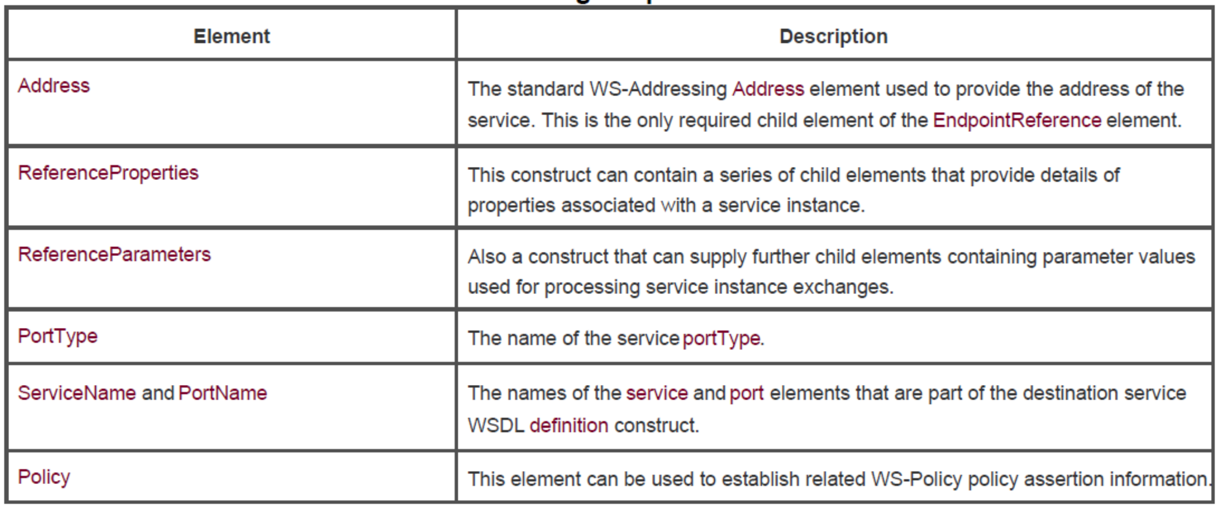
The orchestration service layer introduces a parent level of abstraction that alleviates the need for other services to manage interaction details required to ensure that service operations are executed in a specific sequence (Figure 9.5). Within the orchestration service layer, *process services* compose other services that provide specific sets of functions, independent of the business rules and scenario-specific logic required to execute a process instance.

7.a 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.

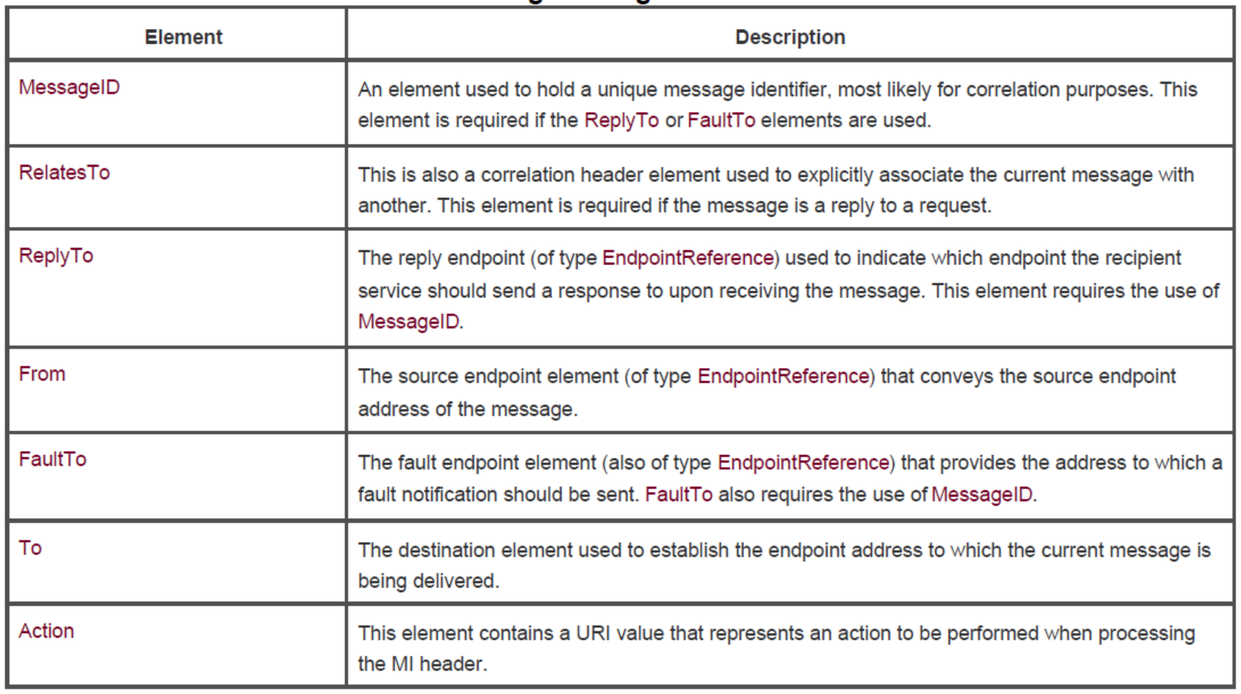


Element	Description
Address	The standard WS-Addressing Address element used to provide the address of the service. This is the only required child element of the EndpointReference element.
ReferenceProperties	This construct can contain a series of child elements that provide details of properties associated 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.

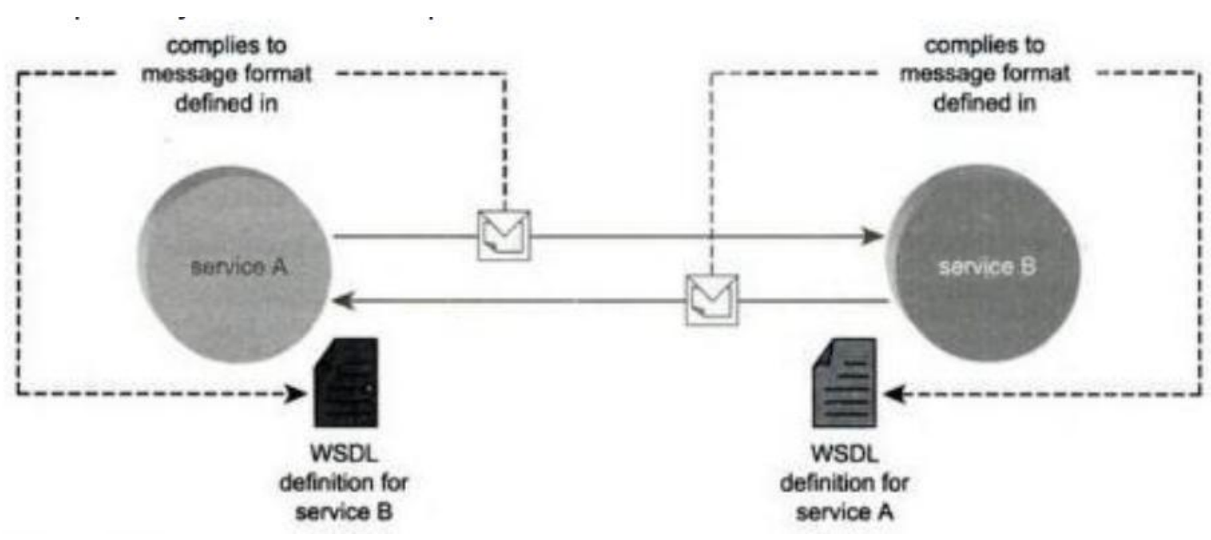
8.Write short notes on the following

a. WSDL

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

b. XML

- 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.
- Within it, XML establishes the format and structure of messages traveling throughout services.
- XSD schemas preserve the integrity and validity of message data,
- XSLT is employed to enable communication between disparate data representations through schema mapping.
- In other words, you cannot make a move within SOA without involving XML.

XML

```xml
<item>
   <title>Empire Burlesque</title>
   <note>Special Edition</note>
   <quantity>1</quantity>
   <price>10.90</price>
 </item>
```

c. 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

d. Security

Security requirements for automation solutions are nothing new to the world of IT. Similarly, service-oriented applications need to be
outfitted to handle many of the traditional security demands of protecting information and ensuring that access to logic is only granted to those permitted.

However, the SOAP messaging communications framework, upon which contemporary SOA is built, emphasizes particular aspects of security that need to be accommodated by a security framework designed specifically for Web services.

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.