Thakur Educational Trust's (Regd.)

THAKUR RAMNARAYAN COLLEGE OF ARTS & COMMERCE

ISO 21001:2018 Certified

PROGRAMME: B.Sc (I.T)

CLASS: S.Y.B.Sc (I.T)

SUBJECT NAME: SOFTWARE ENGINEERING

SEMESTER: IV

FACULTY NAME: Ms. SMRITI DUBEY

UNIT V

Chapter 2 – Service Oriented Engineering

Concepts:

Introduction

Web Services

- Service Oriented Architecture
- Web Standards

Services as reusable Component

Service Engineering

Software Development with Services

- Six stages of Service Construction by composition in system development

Introduction

Service oriented software engineering (SOSE) is a software engineering paradigm that aims to support the development of rapid, low - cost and easy composition of distributed applications by composition of reusable services often provided by other service providers.

It utilizes services as fundamental elements for developing applications and solutions. These services may be provided as web services but the essential element is the dynamic nature of the connection between the service users and service providers.

Web Services

A web service is an instance of a more general notion of a service: "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production."

A web service is: A loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols.

Services are platform and implementation-language independent.

Service-oriented architectures

Service-oriented architecture (SOA) is a means of **developing distributed systems** where the **components are stand-alone services**. Services may execute on different computers from different service providers. Standard protocols have been developed to support service communication and information exchange.

Standard XML-based protocols, such as SOAP and WSDL, have been designed to support service communication and information exchange.

Figure encapsulates the idea of a SOA. **Service providers** design and implement services and specify the interface to these services. They also publish information about these services in an accessible registry. **Service requestors** (sometimes called service clients) who wish to make use of a service discover the specification of that service and locate the service provider. They can then bind their application to that specific service and communicate with it, using standard service protocols.

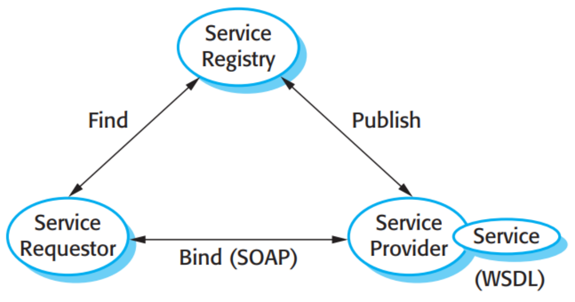


Figure 19.1 Service-oriented architecture

Services are platform and implementation-language independent. Software systems can be constructed by composing local services and external services from different providers, with seamless interaction between the services in the system.

Web Standards

Service-oriented software engineering is as **significant** as object-oriented software engineering. Building applications based on services allows companies and other organizations to cooperate and make use of each other's business functions. Service-based applications may be constructed by linking services from various providers using either a standard programming language or a specialized workflow language.

Figure shows the stack of key standards that have been established to support web services.

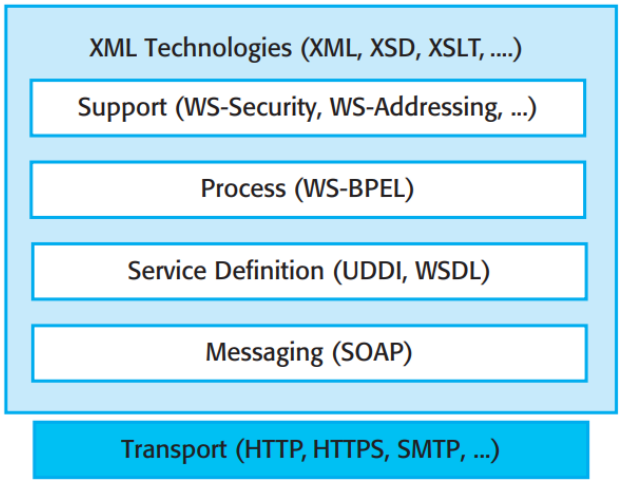

XML Technologies (XML, XSD, XSLT, ...)
Support (WS-Security, WS-Addressing, ...)
Process (WS-BPEL)
Service Definition (UDDI, WSDL)
Messaging (SOAP)
Transport (HTTP, HTTPS, SMTP, ...)

Figure 19.2 Web service standards

Web service protocols cover all aspects of SOAs, from the basic mechanisms for service information exchange (SOAP) to programming language standards (WS-BPEL). These standards are all based on XML, a human and machine-readable notation that allows the definition of structured data where text is tagged with a meaningful identifier.

XML has a range of supporting technologies, such as XSD for schema definition, which are used to extend and manipulate XML descriptions. Erl (2004) provides a good summary of XML technologies and their role in web services. Briefly, the key standards for web SOAs are as follows:

1. SOAP - This is a message interchange standard that supports the communication between services. It defines the essential and optional components of messages passed between services.

2. WSDL - The Web Service Definition Language (WSDL) is a standard for service interface definition. It sets out how the service operations (operation names, parameters, and their types) and service bindings should be defined.

3. WS-BPEL - This is a standard for a workflow language that is used to define process programs involving several different services.

4. UDDI - A service discovery standard, UDDI, was also proposed but this has not been widely adopted. The UDDI (Universal Description, Discovery and Integration) standard defines the components of a service specification, which may be used to discover the existence of a service. These include information about the service provider, the services provided, the location of the WSDL description of the service interface, and information about business relationships. The intention was that this standard would allow companies to set up registries with UDDI descriptions defining the services that they offered.

The principal SOA standards are supported by a range of supporting standards that focus on more specialized aspects of SOA. Some examples of these standards include the following:

1. WS-Reliable Messaging, a standard for message exchange that ensures messages will be delivered once and once only.

2. WS-Security, a set of standards supporting web service security including standards that specify the definition of security policies and standards that cover the use of digital signatures.

3. WS-Addressing, which defines how address information should be represented in a SOAP message.

4. WS-Transactions, which defines how transactions across distributed services should be coordinated.

Services as Reusable Components

Services are a natural development of software components where the component model is, in essence, a set of standards associated with web services. **A service can therefore be defined as the following: A loosely-coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML based protocols.**

When you intend to use a web service, you need to know where the service is located (its URI) and the details of its interface. **These are described in a service description expressed in an XML-based language called WSDL**. The **WSDL specification defines three things about a web service: what the service does, how it communicates, and where to find it:**

1. The **'what'** part of a WSDL document, called an interface, specifies what operations the service supports, and defines the format of the messages that are sent and received by the service.

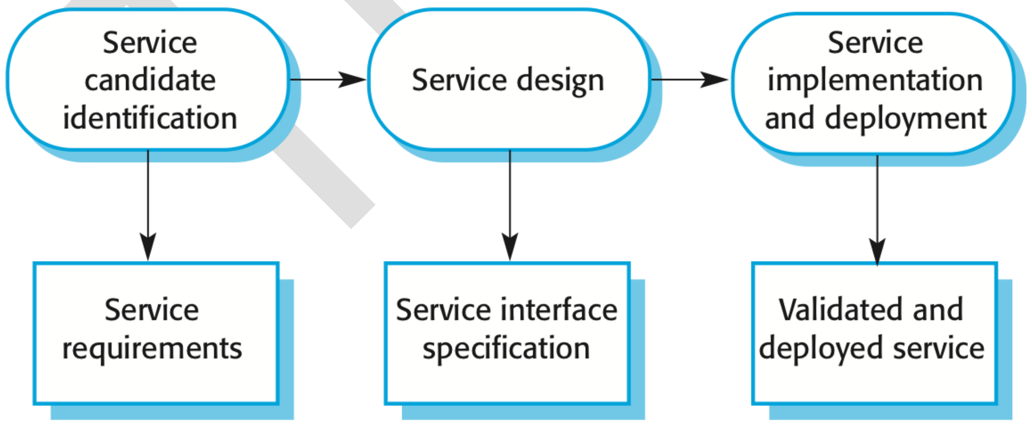
2. The **'how'** part of a WSDL document, called a binding, maps the abstract interface to a concrete set of protocols. The binding specifies the technical details of how to communicate with a web service.

3. The **'where'** part of a WSDL document describes the location of a specific web service implementation (its endpoint).

Service engineering

Service engineering is the process of **developing services for reuse** in service-oriented applications. The service has to be designed as a reusable abstraction that can be used in different systems. Generally useful functionality associated with that abstraction must be designed and the service must be robust and reliable. The service must be documented so that it can be discovered and understood by potential users.

There are three logical stages in the service engineering process, as shown in Figure. These are as follows:


Service candidate identification
Service design
Service implementation and deployment
Service requirements
Service interface specification
Validated and deployed service

1. Service candidate identification, where you identify possible services that might be implemented and define the service requirements. It involves understanding an organization's business processes to decide which reusable services could support these processes.

Three **fundamental types** of service:

- **Utility services** that implement general functionality used by different business processes.
- **Business services** that are associated with a specific business function e.g., in a university, student registration.
- **Coordination services** that support composite processes such as ordering.

2. Service design, where you design the logical and WSDL service interfaces. Involves thinking about the operations associated with the service and the messages exchanged. The number of messages exchanged to complete a service request should normally be minimized. Service state information may have to be included in messages. Interface design stages:

- **Logical interface design.** Starts with the service requirements and defines the operation names and parameters associated with the service. Exceptions should also be defined.
- **Message design (SOAP).** For SOAP-based services, design the structure and organization of the input and output messages. Notations such as the UML are a more abstract representation than XML. The logical specification is converted to a WSDL description.
- **Interface design (REST).** Design how the required operations map onto REST operations and what resources are required.

3. Service implementation and deployment, where you implement and test the service and make it available for use. Programming services using a standard programming language or a workflow language. Services then have to be tested by creating input messages and checking that the output messages produced are as expected. Deployment involves publicizing the service and installing it on a web server. Current servers provide support for service installation.

Software Development with Services

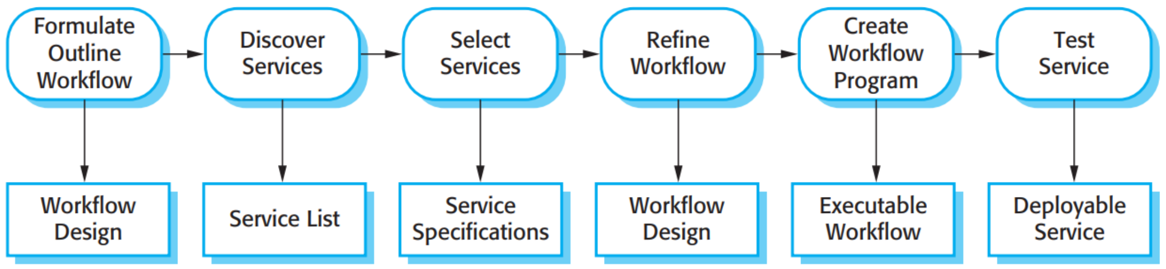
The development of software using services is based around the idea that you compose and configure services to create new, composite services. These may be integrated with a user interface implemented in a browser to create a web application, or may be used as components in some other service composition.

The services involved in the composition may be specially developed for the application, may be business services developed within a company, or may be services from an external provider. Many companies are now converting their enterprise applications into service-oriented systems, where the basic application building block is a service rather than a component.

This opens up the possibility of more widespread reuse within the company. The next stage will be the development of interorganizational applications between trusted suppliers, who will use each other's services. The final realization of the long-term vision of SOAs will rely on the development of a 'services market', where services are bought from external suppliers

The process of designing new services by reusing existing services is essentially a process of software design with reuse. Design with reuse inevitably involves requirements compromises. The 'ideal' requirements for the system have to be modified to reflect the services that are actually available, whose costs fall within budget and whose quality of service is acceptable.

In Figure, there are six key stages in the process of service construction by composition:



1. Formulate outline workflow - In this initial stage of service design, you use the requirements for the composite service as a basis for creating an 'ideal' service design. You should create a fairly abstract design at this stage with the intention of adding details once you know more about available services.

2. Discover services - During this stage of the process, you search service registries or catalogs to discover what services exist, who provides these services, and the details of the service provision

3. Select possible services - From the set of possible service candidates that you have discovered, you then select possible services that can implement workflow activities. Your selection criteria will obviously include the functionality of the services offered. They may also include the cost of the services and the quality of service (responsiveness, availability, etc.) offered.

4. Refine workflow - On the basis of information about the services that you have selected, you then refine the workflow. This involves adding detail to the abstract description and perhaps adding or removing workflow activities. You may then repeat the service discovery and selection stages. Once a stable set of services has been chosen and the final workflow design established, you move on to the next stage in the process.

5. Create workflow program - During this stage, the abstract workflow design is transformed to an executable program and the service interface is defined. You can use a conventional programming language, such as Java or C#, for service implementation or a workflow language, such as WS-BPEL. This stage may also involve the creation of web-based user interfaces to allow the new service to be accessed from a web browser.

6. Test completed service or application - The process of testing the completed, composite service is more complex than component testing in situations where external services are used