T.Y. B.Sc.

(Computer Science)

SEMESTER - V (CBCS)

WEB SERVICES

SUBJECT CODE - USCS506

Prof.(Dr.) D. T. Shirke
Offg. Vice-Chancellor,
University of Mumbai,

Prin. Dr. Ajay Bhamare
Offg. Pro Vice-Chancellor,
University of Mumbai,

Prof. Prakash Mahanwar
Director,
IDOL, University of Mumbai,

Programme Co-ordinator : **Shri. Mandar Bhanushe**
Head, Faculty of Science and Technology
IDOL, University of Mumbai, Mumbai

Course Co-ordinator & Editor : **Ms. Mitali Vijay Shewale**
Doctoral Researcher Scholar,
Veermata Jijabai Technological Institute
Mumbai

Course Writers : **Mr. Abhijeet Pawaskar**
Assistant Professor,
Thakur Educational Trusts Thakur College Of Science And Commerce Thakur Village
W E Highway Kandivli East Mumbai

: **Mr. Milind Thorat**
Lecturer, KJSIEIT,
Mumbai

: **Sujatha Sundar**
Assistant Professor,
Satish Pradhan Dnyanasadhana College

August 2023, Print - I

Published by : Director
Institute of Distance and Open Learning ,
University of Mumbai,Vidyanagari, Mumbai -400 098.

DTP Composed : Mumbai University Press
Printed by Vidyanagari, Santacruz (E), Mumbai - 400 098

CONTENTS

TOPICS (Credits : 03 Lectures/ Work : 03)

WEB SERVICES

Course : USCS506

Objectives :

To understand the details of Web services technologies like SOAP, WSDL, and UDDI. To learn how to implement and deploy web service client and server. To understand the design principles and application of SOAP and REST based web services (JAX-Ws and JAX-RS). To understand WCF service. To design web services and QoS of Web Services.

Expected Learning Outcomes :

Emphasis on SOAP based web services and associated standards such as WSDL. Design SOAP based/ RESTful / WCF services Deal with QoS issues of Web Services.

Unit I.

Web Services basic :

What are Web Services? Types of Web Services Distributed computing infrastructure, overview of XML, SOAP, building Web Services with JAX-WS, Registering and Discovering Web Services, Service oriented Architecture, Web Services Development Life Cycle, Developing and consuming simple Web Services across platform.

Unit II.

The REST Architectural style :

Introducing HTTP, The core architectural elements of a RESTful system, Description and discovery of RESTful web services, Java tools and frameworks for building RESTful web services, JSON message format and tools and frameworks around JSON, Build RESTful Web Services with JAX-RS APIs, The Description and Discovery of RESTful Web Services. Design guidelines RETful web services, Secure RESTful web services.

Unit III.

Developing Service-Oriented Applications with WCF :

What Is Windows Communication Foundation, Fundamental Windows Communication Foundation Concepts, Windows Communication

Foundation Architecture, WCF and NET Framework Client Profile, Basic WCF Programming, WCF Feature Details. Web Service QoS.

Textbook (s) :

1) Web Services : principles and Technology, Michael P. Papazoglou, Pearson Education Limited, 2008.
2) RESTful Java Services, Jobinesh Purushothaman, PACKT Publishing, 2nd Edition, 2015.
3) Developing Service-Oriented Application with ECF, Microft, 2017 https://docs.miscrosoft.com/en-us/dotnet/framework/wcf/index

Additional References (s) :

1) Leonard Richardson and Sam Ruby, RESTful Web Services, O'Reilly, 2007
2) The java EE 6 Tutorial, Oracle, 2013.

❖❖❖❖

1

WEB SERVICES BASICS

Unit Structure :

1.0 OBJECTIVE

- This chapter explores web services, their principles, technologies, and best practices.
- It covers fundamental concepts like SOAP and XML, enabling seamless integration across platforms.
- Readers learn to design robust web services, handle authentication, security, versioning, and optimize performance.
- Advanced topics include microservices, GraphQL, and event-driven services.
- The book also addresses API documentation, testing, and monitoring. Practical insights from experienced developers offer valuable perspectives, making this book suitable for both novices and experts in web service development.

1.1 INTRODUCTION

In the realm of web services, where the digital world converges to exchange information and deliver seamless experiences, the art of designing robust services emerges as a cornerstone of success. In this pivotal chapter, we embark on a journey to master the principles and practices that breathe life into web services, making them reliable, scalable, and efficient.

The foundation of any web service lies in its Application Programming Interface (API), the interface that defines the rules of engagement between applications. As we delve into the intricacies of API design, we unravel the secrets to crafting clear, intuitive, and consistent interfaces that empower developers to interact with your services effortlessly.

But a great API alone is not enough to ensure a seamless user experience. Data representation plays a crucial role in bridging the gap between diverse systems, and we will explore the best practices of structuring data using technologies such as JSON, XML, and others. You will gain insights into the art of data serialization, ensuring that information flows flawlessly across the digital divide.

As the world becomes increasingly interconnected, the importance of security and authentication cannot be overstated. We will delve into the methods of safeguarding your services, implementing robust authentication mechanisms, and establishing trust between applications. By the end of this chapter, you will be armed with the knowledge to build secure and protected web services.

Moreover, we acknowledge that no system is without flaws, and web services are no exception. Hence, we tackle the crucial topic of error handling with utmost care. Understanding how to gracefully handle errors

and communicate them effectively to clients is paramount in delivering a user experience that instills confidence.

As your web services grow and evolve, versioning becomes an inevitable aspect of their lifecycle. We will equip you with strategies to manage multiple versions of your services without causing disruption or chaos. With the wisdom imparted in this chapter, versioning will become a seamless and organized process.

Finally, we address the performance optimization of your web services, a critical element in ensuring smooth operation and responsiveness. By the end of this chapter, you will have the tools and techniques to fine-tune your services, making them agile and capable of handling an ever-increasing load.

Are you ready to elevate your web services to new heights? Join us on this voyage of API design, data representation, security, error handling, versioning, and performance optimization. Armed with the knowledge gained in this chapter, you will transform your web services into robustand reliable engines that drive the digital landscape with confidence and finesse.

1.1.1 What are Web services?

Web services are a set of technologies and standards used for communication and data exchange between different software applications and systems over the internet. They facilitate seamless integration and interaction among diverse platforms, enabling them to work together and share information efficiently.

At their core, web services are a way for software applications to communicate with each other using standard protocols and data formats, irrespective of the programming languages or operating systems they are built upon. This interoperability and platform independence make web services a fundamental component of modern software development and the foundation of the interconnected digital world we experience today.

The key components that define web services include:

1. **Standard Protocols:** Web services utilize well-established communication protocols like HTTP (Hypertext Transfer Protocol) and HTTPS (HTTP Secure) for data exchange. These protocols ensure that the communication between applications is reliable and secure.

2. **XML (eXtensible Markup Language) or JSON (JavaScript Object Notation):** These are the most commonly used data formats for structuring and representing data in web services. XML has been traditionally popular, while JSON has gained significant traction due to its lightweight and human-readable nature.

3. **APIs (Application Programming Interfaces):** APIs act as interfaces that allow different software applications to interact with each other. Web services expose APIs that define the methods, data structures, and communication rules that applications need to follow to request or provide services.

4. **Service Description:** Web services are described using standard languages like WSDL (Web Services Description Language) or OpenAPI, which provide machine-readable specifications of the services offered, including their methods, input parameters, and response formats.

1.1.2 Types of Web services

Topologically, Web services can come in two flavors, see Figure 1.1. Informational, ortype I, Web services, which support only simple request/response operations and alwayswait for a request; they process it and respond. Complex, or type II Web services implement some form of coordination between inbound and outbound operations. Each of thesetwo models exhibits several important characteristics and is in turn subdivided in morepecialized subcategories.

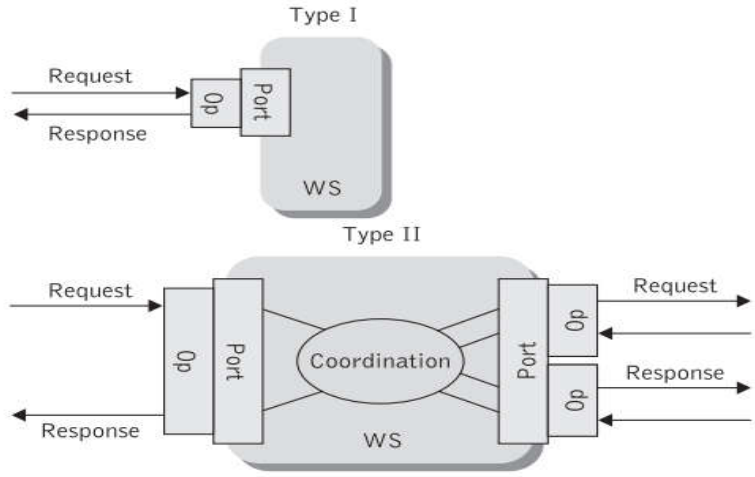


Fig 1.1 High-level view of informational and complex services

1.1.2.1 Simple or informational services

Informational or Type I web services, also known as Document-style web services, are one of the two primary classifications of web services based on their communication style and message format. These web services are designed to exchange information in the form of XML documents and are primarily associated with the Simple Object Access Protocol (SOAP) protocol.

In Type I web services, the focus is on the structure and content of the XML documents that are transmitted between applications. These XML documents are often defined using Web Services Description Language

(WSDL), a standard language used to describe the interface and functionality of the web service.

Key characteristics of Informational or Type I web services include:

1. **SOAP Protocol:** Type I web services predominantly use the SOAP protocol for communication. SOAP provides a standardized way to structure and format messages, ensuring consistency and interoperability across different platforms and programming languages.

2. **XML-Based Messages:** The data exchanged in Type I web services is typically represented as XML documents. XML allows for the structured representation of data, making it suitable for complex and diverse data types.

3. **Document-Centric Approach:** The primary focus of Type I web services is on exchanging XML documents that contain data and information. Unlike Type II web services (RPC-style web services), which emphasize remote method calls and procedural communication, Type I web services prioritize data exchange and document handling.

4. **Platform and Language Independence:** Informational web services promote platform independence, allowing applications written in different programming languages and running on different platforms to communicate seamlessly.

5. **WSDL Description:** Type I web services often provide a WSDL description, which acts as a contract between the service provider and the service consumer. It defines the structure of the XML messages and the operations that the web service supports.

6. **Statelessness:** Like all web services, Type I web services follow the stateless nature of the HTTP protocol, meaning that each request from the client to the server is treated as an independent request without any connection or session maintenance.

Informational or Type I web services find their applications in scenarios where the focus is on exchanging structured data and documents rather than invoking specific remote procedures. They are commonly used in enterprise-level applications, data exchange between different systems, and scenarios where a formal contract between the service provider and consumer is required.

As the world of web services continues to evolve, both Type I and Type II (RPC-style) web services have their unique strengths and use cases, providing developers with flexible options to choose the most suitable communication style based on their specific requirements.

1.1.2.2 Complex services or business processes

Complex, or Type II web services, also known as RPC-style (Remote Procedure Call) web services, are one of the two primary classifications of

web services based on their communication style and message format. Unlike Type I web services (Informational), which focus on exchanging data in the form of XML documents, Type II web services emphasize remote method invocation and procedural communication.

In Type II web services, the primary goal is to execute specific functions or methods on the remote server and receive the results in a structured format. This type of web service is commonly associated with using the SOAP (Simple Object Access Protocol) protocol for communication, similar to Type I web services. However, instead of dealing with XML documents, Type II web services typically use XML messages to define the method calls and parameters.

Key characteristics of Complex or Type II web services include:

1. **RPC-Style Communication:** Type II web services follow a remote procedure call (RPC) style of communication. The client application invokes remote methods on the server-side, and the server processes the requests and returns the results to the client.

2. **SOAP Protocol:** Like Type I web services, Type II web services predominantly use the SOAP protocol for communication. This ensures a standardized and secure way of exchanging messages between applications.

3. **XML-Based Messages:** While Type II web services use XML messages, they are more focused on specifying the remote method calls and their parameters. The XML messages act as wrappers for invoking methods on the server.

4. **Procedural Invocation:** Type II web services emphasize the procedural aspect of remote method invocation, where the client invokes specific functions on the server-side, and the server processes the request accordingly.

5. **Platform and Language Independence:** Like all web services, Type II web services offer platform independence, enabling applications built on different platforms and programming languages to communicate effectively.

6. **WSDL Description:** Type II web services also provide a WSDL description that outlines the methods available for invocation, their input parameters, and expected responses.

Complex or Type II web services are commonly used in scenarios where applications require remote execution of specific functions or tasks. They are well-suited for situations where a client application needs to interact with a server-side service to perform specialized operations or access certain functionalities.

It's important to note that while Type I and Type II web services are two primary classifications, the lines between them can sometimes be blurred, and modern web service implementations often combine aspects of both types to meet specific application requirements.

1.2 DISTRIBUTED COMPUTING INFRASTRUCTURE

A distributed system is characterized as a collection of (probably heterogeneous)networked computers, which communicate and coordinate their actions by passing messages. Distribution is transparent to the user so that the system appears as a single integrated facility. This is in contrast to a network infrastructure, where the user is aware that there are several machines, is also aware of their location, storage replication, and load balancing, and functionality is not transparent.

A distributed system has numerous operational components (computational elements,such as servers and other processors, or applications) which are distributed over variousinterconnected computer systems. Components are autonomous as they posses full controlover their parts at all times. In addition, there is no central control in the sense that a single component assumes control over all the other components in a distributed system.Distributed systems usually use some kind of client–server organization. A computer systemthat hosts some component of a distributed system is referred to as a host. Distributedcomponents are typically heterogeneous in that they are written in different programming languages and may operate under different operating systems and diverse hardware platforms. The sharing of resources is the main motivation for constructing distributed systems. As a consequence of component autonomy, distributed systems execute applications concurrently. Consequently, there are potentially as many processes in a distributed system asthere are components. Furthermore, applications are often multi-threaded. They may create a new thread whenever they start to perform a service for a user or another application. In this way the application is not blocked while it is executing a service and isavailable to respond to further service requests.

1.2.1 Internet protocols

Certainly! In a general sense, internet protocols are a set of rules and conventions that dictate how devices communicate and exchange data over the Internet. They enable devices to understand and interpret each other's messages, ensuring effective and reliable data transmission. Internet protocols cover various aspects of online communication, such as addressing, data formatting, error handling, and security.

Imagine the internet as a vast global network of interconnected devices like computers, servers, smartphones, and other gadgets. These devices need a common language to communicate and share information with each other. Internet protocols provide that common language, ensuring that data packets can be correctly sent, received, and interpreted across the network.

For instance, when you browse the web, your web browser (the client) uses the HTTP protocol to request web pages from a web server. The server then responds with the requested data using the same protocol. In the background, TCP ensures that the data packets arrive in the correct order and without errors.

Similarly, when you send an email, your email client uses SMTP to send the email to your email server, which then uses either POP3 or IMAP to retrieve the email on another device. DNS helps translate human-readable domain names (like google.com) into IP addresses so that your browser can find the correct server to load the website.

These are just a few examples of how internet protocols facilitate communication between devices and services on the internet. They play a fundamental role in enabling the vast array of online activities we engage in daily, from web browsing and email to file transfers, online gaming, video streaming, and more. Without internet protocols, the internet, as we know it, would not function.

1.2.1.1 The Open Systems Interconnection reference model

The Open Systems Interconnection (OSI) reference model is a conceptual framework used to understand and describe how different networking protocols and technologies interact and work together in a networked communication system. It was developed by the International Organization for Standardization (ISO) in the early 1980s. The model is not a specific implementation but rather a guideline for creating network communication standards and products that are interoperable and compatible.

The OSI model consists of seven layers, each representing a specific function of network communication. These layers are stacked in a hierarchical manner, with data passing through each layer as it moves from the application on one device to the application on another device in a network. Each layer performs its designated tasks and communicates with its adjacent layers. The seven layers of the OSI model are as follows:

1. **Physical Layer:** The lowest layer deals with the physical medium over which data is transmitted, such as cables, electrical signals, or wireless transmission. It defines the physical characteristics of the network, such as voltage levels, data rates, and connector types.

2. **Data Link Layer:** This layer is responsible for the reliable transmission of data between two directly connected devices over a specific physical link. It deals with error detection, flow control, and addressing of devices on the local network.

3. **Network Layer:** The network layer is responsible for routing data packets between different networks. It determines the optimal path for

data to travel from the source to the destination based on network topology and addressing.

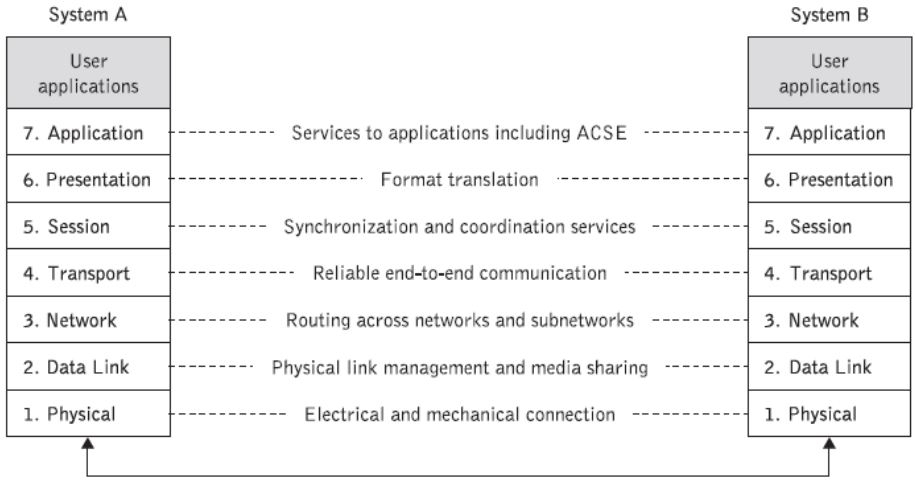
4. **Transport Layer:** The transport layer ensures reliable and error-free data delivery between two devices. It breaks data into smaller segments, manages acknowledgment and retransmission of lost data, and deals with flow control.

5. **Session Layer:** This layer establishes, maintains, and terminates sessions (connections) between applications on different devices. It allows synchronization and coordination between applications.

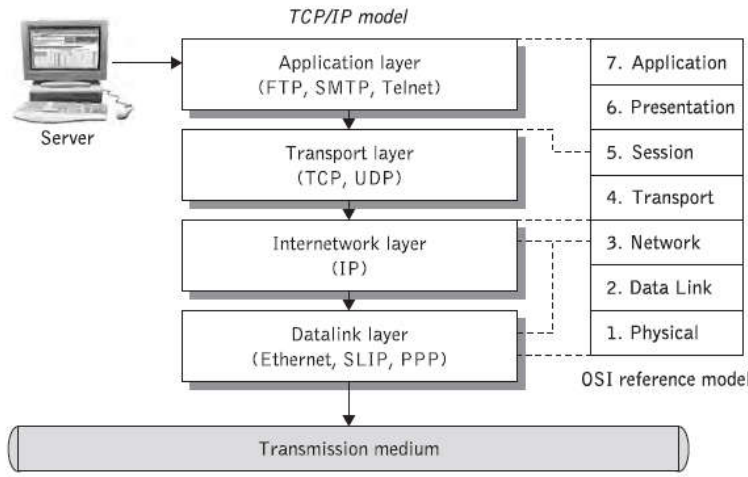
6. **Presentation Layer:** The presentation layer is responsible for data translation, encryption, and compression. It ensures that data from the application layer of one device can be understood by the application layer of another device.

7. **Application Layer:** The top layer of the OSI model represents the user interface and serves as the entry point for network applications. It enables communication between user applications and the network.

The OSI model provides a clear division of network communication tasks into manageable layers, making it easier for developers to design, implement, and troubleshoot network protocols and systems. Though modern networking technologies often do not strictly adhere to the OSI model, it remains a valuable reference for understanding the fundamentals of network communication and interoperability.



1.2.1.2 The TCP/IP network protocol



The TCP/IP (Transmission Control Protocol/Internet Protocol) is a set of communication protocols used for transmitting data over networks, including the Internet. It is the foundational protocol suite for the internet and has become the standard for network communication across the globe.

TCP/IP was developed in the 1970s by the United States Department of Defense (DoD) to connect various computer systems and networks. Over time, it has evolved and become the backbone of the modern internet and intranets.

The TCP/IP protocol suite consists of several protocols, organized into four layers, which are loosely related to the OSI model but do not exactly match its layers. These layers are:

1. **Application Layer:** The top layer is responsible for providing network services directly to applications and end-users. It includes various protocols for different applications, such as HTTP (for web browsing), SMTP (for email), FTP (for file transfer), and DNS (for domain name resolution).

2. **Transport Layer:** This layer is responsible for end-to-end communication between devices. It ensures that data is reliably and accurately delivered between applications running on different devices. The two main protocols in this layer are:

- **Transmission Control Protocol (TCP):** TCP is a connection-oriented protocol that provides reliable, error-checked data transmission. It establishes a connection before data transfer and ensures that data packets are delivered in the correct order and without errors.

- **User Datagram Protocol (UDP):** UDP is a connectionless protocol that offers faster but less reliable communication. It is often used for time-sensitive applications where some data loss is acceptable, such as real-time video streaming and online gaming.

3. **Internet Layer:** The internet layer handles the addressing, routing, and forwarding of data packets between different networks. It uses IP (Internet Protocol) for this purpose. IP provides unique addresses (IPv4 or IPv6) to identify devices on the internet, allowing data to be routed from the source to the destination.

4. **Link Layer (Network Interface Layer):** The lowest layer deals with the physical transmission of data over the local network medium. It includes protocols specific to the type of network technology being used, such as Ethernet, Wi-Fi, or PPP (Point-to-Point Protocol).

TCP/IP is considered an open standard because its specifications are publicly available, allowing anyone to implement the protocols without restrictions. This openness and its scalability have contributed to its widespread adoption and success as the primary network protocol for the internet. It enables seamless communication between a vast array of devices, ranging from computers and smartphones to IoT devices and servers, connecting the world in a global network.

1.2.2 Middleware

Middleware refers to software that acts as an intermediary or bridge between different applications, systems, or components within a distributed computing environment. It plays a crucial role in facilitating communication, data exchange, and integration between various software applications and services.

In a distributed computing environment, where applications and resources are spread across multiple systems and locations, middleware provides a standardized way for these components to interact with each other. It abstracts the complexities of low-level network communication and allows applications to communicate and share data without being concerned about the underlying infrastructure.

Middleware offers several key functionalities, including:

1. **Communication:** Middleware enables communication between different applications and systems, regardless of the underlying hardware, operating systems, or programming languages they use. It provides a common set of communication protocols and APIs, making it easier for applications to exchange data and messages.

2. **Integration:** Middleware allows disparate systems and applications to work together seamlessly by providing mechanisms for data transformation, message routing, and protocol translation. It helps achieve interoperability and smooth data flow across different components.

3. **Distributed Computing:** Middleware is essential for managing and coordinating distributed computing tasks. It supports features like remote procedure calls (RPCs), message queues, and publish-subscribe mechanisms, which enable distributed applications to collaborate and share resources efficiently.

4. **Security:** Middleware often includes security features to ensure secure communication and data exchange between applications. It may provide encryption, authentication, and authorization mechanisms to protect sensitive information and prevent unauthorized access.

5. **Transaction Management:** Middleware supports distributed transactions, allowing multiple operations across different systems to be grouped together as a single unit. If any part of the transaction fails, the middleware ensures that the entire transaction is rolled back, maintaining data consistency.

6. **Scalability and Load Balancing:** Middleware solutions can help distribute workloads across multiple servers, ensuring that resources are used efficiently and that the system can handle increased demand without becoming overwhelmed.

Examples of middleware include:

- **Message-oriented middleware (MOM):** Provides messaging services that allow applications to send and receive messages in a decoupled manner. Examples include Apache Kafka and RabbitMQ.

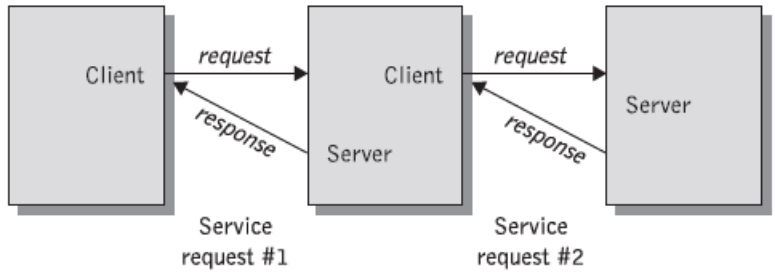
- **Remote Procedure Call (RPC) middleware:** Enables applications to call functions or procedures on remote systems as if they were local. Common implementations include Java RMI and gRPC.

- **Object Request Brokers (ORBs):** Facilitate communication between objects in distributed object-oriented systems. Examples include CORBA (Common Object Request Broker Architecture).

- **Database Middleware:** Allows applications to interact with databases without knowing the underlying database system. Examples include ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity).

Middleware plays a crucial role in modern distributed systems, enabling seamless integration, improved scalability, and efficient communication between various software components, which is especially essential in complex and interconnected environments.

1.2.3 The client–server model



The client-server model is a computing architecture that describes the relationship and interaction between two types of entities: the "client" and the "server." It is a fundamental concept in distributed computing and networking, where multiple devices (clients) communicate with a centralized device or service (server) to access resources, services, or data.

Here's an overview of how the client-server model works:

1. **Client:** The client is an end-user device or application that requests and consumes services or resources from the server. Clients can be desktop computers, laptops, smartphones, tablets, or any other device capable of making requests over a network.

2. **Server:** The server is a powerful and centralized computing device or software application that provides specific services, data, or resources to clients. Servers are designed to be always available and responsive to client requests.

3. **Communication:** In the client-server model, communication between the client and the server is typically based on a request-response paradigm. The client sends a request to the server, specifying the service or resource it needs. The server processes the request, performs the necessary actions, and sends a response back to the client with the requested data or confirmation of the action taken.

4. **Statelessness:** In many cases, the client-server interaction is considered "stateless." It means that each request from the client to the server is independent and does not rely on previous requests. The server treats each request as a separate transaction, and the client includes all the necessary information for the server to process the request completely.

5. **Scalability:** The client-server model allows for scalable systems. Multiple clients can connect to the same server simultaneously, and additional servers can be added to handle increased client demands. This scalability is one of the reasons why the client-server model is widely used in various applications and services.

Examples of the client-server model in practice:

- **Web Browsing:** When you use a web browser (client) to access a website, the browser sends requests to the web server, which then responds with the requested web page, images, or other resources.

- **Email:** When you use an email client (client) to send or receive emails, it communicates with the email server, which stores and manages the email messages.

- **Online Gaming:** In multiplayer online games, players' devices act as clients that connect to a central game server to interact with other players and access game data.

- **File Sharing:** In a file-sharing system, clients request files or data from a file server, which grants access to the requested resources.

The client-server model simplifies the design and implementation of distributed systems, as it provides a clear separation of concerns between clients and servers. The server's centralized management allows for better control, security, and efficient resource utilization, while clients can be lightweight and focus on providing a user-friendly interface to interact with the server's resources.

1.3 OVERVIEW OF XML

XML is an extensible markup language used for the description and delivery ofmarked-upelectronic text over the Web. Two important characteristics of XML distinguish it fromother markup languages: its document type concept and its portability.An important aspect of XML is its notion of a document type. XML documents areregarded as having types. XML’s constituent parts and their structure formally define thetype of a document.Another basic design feature of XML is to ensure that documents are portable betweendifferent computing environments. All XML documents, whatever language or writingsystem they employ, use the same underlying character encoding scheme. This encodingis defined by the international standard Unicode, which is a standard encoding system thatsupports characters of diverse natural languages.

1.3.1 XML declaration

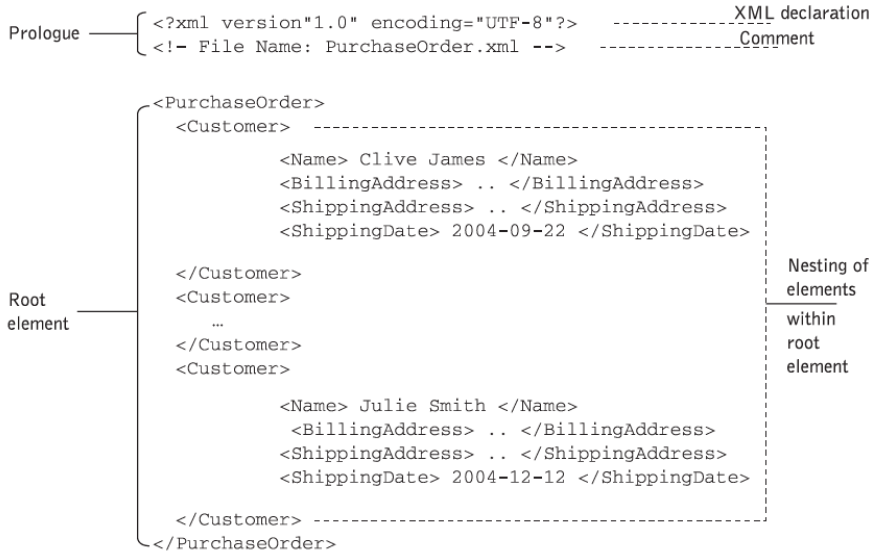


Image courtesy Web.services...principles.and.technology.pdf

The first few characters of an XML document must make up an XMLdeclaration.The XML processing software uses the declaration to determine how to deal with thesubsequent XML content. A typical XML declaration begins with a prologue that typicallycontains a declaration of conformity to version 1.0 of the XML standard and to the UTF-8encoding standard: <?xml version="1.0" encoding="UTF-8"?>. This is shownin Figure

1.3.2 URIs and XML namespaces

URIs (Uniform Resource Identifiers) and XML namespaces are important concepts in XML (eXtensible Markup Language) that help in uniquely identifying and organizing elements and attributes within an XML document. Let's explore them in more detail:

1. URIs (Uniform Resource Identifiers):
A URI is a string of characters used to identify a resource, either on the internet or within an XML document. URIs provide a standardized way to reference resources, making it possible to locate and access them. The most common types of URIs are:

- URL (Uniform Resource Locator): A specific type of URI that provides the address of a resource on the internet. It includes the protocol (e.g., "http://" or "https://") followed by the domain name or IP address and the resource's path.

- URN (Uniform Resource Name): Another type of URI that is intended to be globally unique and persistent even if the resource's location changes. URNs are used to identify resources without specifying their location.

In XML, URIs are commonly used to identify XML namespaces, allowing multiple vocabularies or sets of element and attribute names to coexist within the same XML document without conflicting with each other.

2. XML namespaces:

XML namespaces are a way to avoid naming conflicts when using XML elements and attributes. They provide a method for qualifying element and attribute names with a unique identifier (usually a URI) to indicate which vocabulary or schema the elements and attributes belong to.

In XML documents, namespaces are defined using the `xmlns` attribute. This attribute is added to the root element of the XML document or any element that needs to be associated with a specific namespace. The `xmlns` attribute is followed by the namespace URI.

For example:

```xml
<root xmlns="http://www.example.com/ns1">
<element1>Some content</element1>
<element2>Some other content</element2>
</root>
```

In the above example, the elements `element1` and `element2` are associated with the namespace identified by the URI "http://www.example.com/ns1".

When using XML namespaces, you need to use the namespace prefix to qualify the elements and attributes within the XML document. The prefix is typically defined using another `xmlns` attribute with a unique prefix name and the associated namespace URI.

For example:

```xml
<root xmlns:prefix="http://www.example.com/ns1">
<prefix:element1>Some content</prefix:element1>
<prefix:element2>Some other content</prefix:element2>
</root>
```

In this case, the elements are qualified with the prefix "prefix" to indicate they belong to the namespace identified by "http://www.example.com/ns1".

By using XML namespaces, different XML vocabularies can be mixed together in a single document without causing conflicts, enabling better data organization and sharing.

1.3.3 The XML Schema Definition Language

The XML Schema Definition Language (XSD) is a powerful and widely used language for describing the structure and constraints of XML documents. It allows you to define the rules that XML documents must follow, ensuring data consistency and providing a formal way to validate XML content against those rules.

XSD provides a set of elements and attributes that allow you to specify the structure of XML documents, including:

1. **Complex Types:** Defines the structure of elements that can contain other elements or attributes. Complex types can be made up of sequences, choices, and other complex types.

2. **Simple Types:** Defines the data type of elements that contain text content. XSD supports a range of built-in simple data types such as strings, numbers, dates, booleans, etc. It also allows you to define your custom simple types.

3. **Elements:** Define the building blocks of XML documents. Elements can be either simple elements (contain text content only) or complex elements (can contain other elements and attributes).

4. **Attributes:** Define additional properties or metadata for elements. Attributes must be declared in the context of an element or attribute group.

5. **Groups:** Allow you to group related elements or attributes, making it easier to reuse and maintain schemas.

6. **Namespaces:** XML namespaces can also be defined in XSD to enable the use of qualified names for elements and attributes, as discussed in the previous response.

7. **Restrictions and Constraints:** XSD supports a wide range of constraints that can be applied to elements and attributes, such as minimum and maximum occurrence constraints, length constraints, regular expressions, enumerations, etc.

An XML schema file (usually with the file extension ".xsd") contains the definitions of elements, attributes, and types, along with the rules and

constraints governing their usage. XML documents can then be validated against these schema files to ensure that they conform to the defined rules.

For example, a simple XSD definition for a book element might look like this:

```xml
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="book">
<xs:complexType>
<xs:sequence>
<xs:element name="title" type="xs:string"/>
<xs:element name="author" type="xs:string"/>
<xs:element name="publication_year" type="xs:int"/>
</xs:sequence>
<xs:attribute name="isbn" type="xs:string"/>
</xs:complexType>
</xs:element>
</xs:schema>
```

With the above XSD, you can define XML documents like:

```xml
<book isbn="123456789">
<title>Sample Book</title>
<author>John Doe</author>
<publication_year>2023</publication_year>
</book>
```

The XML document above adheres to the rules defined in the XSD schema for the book element. If the document does not follow the rules, a validation error will be raised.

1.4 SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

SOAP (Simple Object Access Protocol) is a messaging protocol used for exchanging structured information in the form of XML-based messages between web services over computer networks, typically over HTTP or SMTP. It is a protocol that facilitates communication between different systems and platforms, allowing them to interact with each other and perform various operations.

Key features of SOAP include:

1. **XML-Based Messages:** SOAP messages are encoded in XML, making them platform-independent and easy to parse. The XML format ensures that the data can be understood by any system that supports XML.

2. **Standardized Messaging:** SOAP provides a standardized way to structure messages, defining rules for message headers and body elements. This standardization enables interoperability between different systems and programming languages.

3. **Transport-Independent:** SOAP messages can be sent over various transport protocols, although HTTP is the most common choice. This flexibility allows SOAP to work in different network environments.

5. **Platform Independence:** SOAP allows communication between services written in different programming languages and running on different platforms. As long as the systems can understand XML and adhere to the SOAP specifications, they can communicate with each other.

6. **Header and Body:** SOAP messages consist of a header and a body. The header contains information about the message, such as authentication details, while the body contains the actual payload (data) of the message.

7. **WS- Specifications:** SOAP can be extended using a set of Web Services specifications, collectively known as WS-* (Web Services specifications), which define additional functionalities such as security, reliability, and transaction support.

SOAP is commonly used in web services environments where remote procedure calls (RPCs) and service-oriented architectures (SOA) are prevalent. It was widely used in the early 2000s but has since faced competition from other web service protocols like REST (Representational State Transfer), which is more lightweight and simpler to use in many scenarios.

While SOAP can be powerful and suitable for certain enterprise-level applications, it also introduces more overhead due to its XML-based messaging and additional complexities of WS-* specifications. Therefore, when developing new web services, RESTful APIs (using HTTP and JSON) have become more popular due to their simplicity, ease of use, and efficiency.

1.5 BUILDING WEB SERVICES WITH JAX-WS

JAX-WS (Java API for XML Web Services) is a Java API that allows developers to build and deploy web services using XML-based SOAP messaging. It is part of the Java EE (Java Platform, Enterprise Edition) specification and provides a standard way to create and consume web services in Java.

Here's a step-by-step guide on building web services with JAX-WS:

1. Define the Service Endpoint Interface (SEI):
Start by defining the Service Endpoint Interface, which is a Java interface that declares the methods that will be exposed as web service operations. These methods will be annotated with JAX-WS annotations to specify how they should be exposed as web service operations.

```java
import javax.jws.WebMethod;
import javax.jws.WebService;
@WebService
public interface MyWebService {
    @WebMethod
    String sayHello(String name);
}
```

2. Implement the Service Endpoint Interface:
Create a Java class that implements the Service Endpoint Interface. This class will provide the actual implementation for the web service operations.

```java
@WebService(endpointInterface = "com.example.MyWebService")
public class MyWebServiceImpl implements MyWebService {
    public String sayHello(String name) {
        return "Hello, " + name + "!";
    }
}
```

3. Publish the Web Service:
To publish the web service and make it accessible over the network, you need to create an endpoint and publish it using the JAX-WS runtime.

```java
import javax.xml.ws.Endpoint;
public class Main {
    public static void main(String[] args) {
        String url = "http://localhost:8080/myWebService";
Endpoint.publish(url, new MyWebServiceImpl());

System.out.println("Web service is running at: " + url);
    }
}
```

4. Generate the WSDL:
Once the web service is published, you can generate the WSDL (Web Services Description Language) document. The WSDL describes the web service's interface, operations, and data types.

To generate the WSDL, you can access the WSDL file using the "?wsdl" query parameter in the web service URL, like this: http://localhost:8080/myWebService?wsdl

5. Create a Client for the Web Service:
To consume the web service, you need to create a client that interacts with the web service's methods using SOAP messages.

Java provides the `wsimport` tool to generate the client artifacts from the WSDL:

```sh
wsimport -keep http://localhost:8080/myWebService?wsdl
```

This will generate the necessary Java classes for the client to invoke the web service methods.

6. Consume the Web Service:
Finally, you can create a Java client to consume the web service by using the generated client artifacts.

```java
import com.example.MyWebService;
import com.example.MyWebServiceImplService;


public class MyWebServiceClient {
   public static void main(String[] args) {
MyWebServiceImplService service = new MyWebServiceImplService();
MyWebService port = service.getMyWebServiceImplPort();

      String result = port.sayHello("John");
System.out.println(result);
   }
}
```

That's it! You've successfully built a web service using JAX-WS and consumed it with a Java client. JAX-WS provides a straightforward and standardized way to develop and interact with XML-based web services in the Java ecosystem.

1.6 REGISTERING AND DISCOVERING WEB SERVICES

Registering and discovering web services are important steps in building a service-oriented architecture (SOA) and enabling service discovery, which allows clients to find and interact with available web services dynamically. There are various approaches and technologies to achieve this, including:

1. UDDI (Universal Description, Discovery, and Integration):
UDDI is a registry-based approach to service discovery. It provides a standard way for businesses to publish and discover web services. Web service providers publish their service descriptions (WSDL) to a UDDI registry, which acts as a central directory of available services. Clients can query the UDDI registry to find services that match their requirements.

2. Service Registries:
Apart from UDDI, some organizations use custom service registries or repositories to store service metadata. These registries can be implemented using databases or other storage mechanisms. Service providers register their services along with their service details in these repositories, and clients can discover services by querying the registry based on various criteria.

3. Service Discovery via Service Discovery Protocols:
Service discovery protocols like SSDP (Simple Service Discovery Protocol) and mDNS (Multicast DNS) provide a more decentralized approach to service discovery within a local network. These protocols allow devices and services to announce their presence and capabilities on the network, and clients can discover and communicate with these services directly.

4. API Gateways and Service Meshes:
In modern microservices architectures, API gateways and service meshes can play a role in service registration and discovery. API gateways act as entry points for incoming client requests and can handle service discovery and routing to different microservices. Service meshes, on the other hand, provide a dedicated infrastructure layer for service-to-service communication and can manage service discovery, load balancing, and traffic routing within the microservices network.

5. DNS-Based Service Discovery:
DNS-based service discovery is another approach where services are registered in the Domain Name System (DNS) as DNS records. Clients can then use DNS queries to discover services by their registered names.

It's essential to choose the right service discovery approach based on the specific needs and scale of your application. For example, in large-scale distributed systems, a combination of DNS-based discovery and service

mesh might be more suitable, while in smaller environments, a centralized UDDI or custom registry could work well.

As technology evolves, new approaches and tools for service registration and discovery may emerge, so it's always a good idea to stay updated with the latest trends in service-oriented architectures and microservices ecosystems.

1.7 SERVICE-ORIENTED ARCHITECTURE

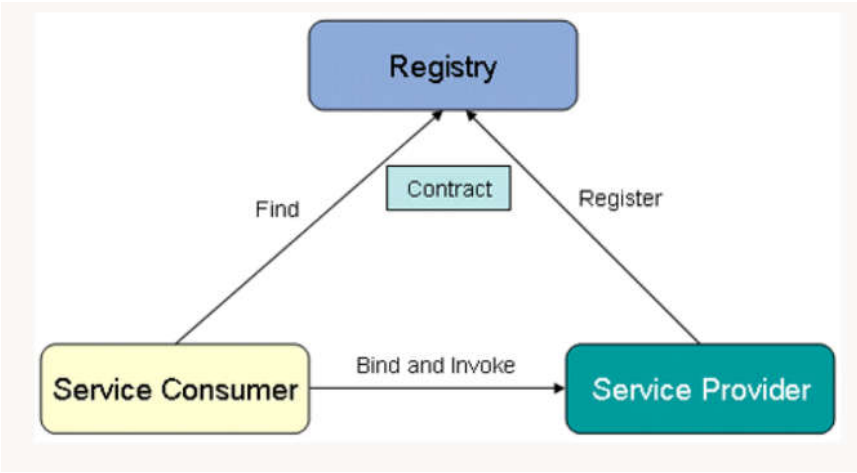
SOA is an architectural style for building software applications that use services available in a network such as the web. It promotes loose coupling between software components so that they can be reused. Applications in SOA are built based on services. A service is an implementation of a well-defined business functionality, and such services can then be consumed by clients in different applications or business processes.

SOA allows for the reuse of existing assets where new services can be created from an existing IT infrastructure of systems. In other words, it enables businesses to leverage existing investments by allowing them to reuse existing applications, and promises interoperability between heterogeneous applications and technologies. SOA provides a level of flexibility that wasn't possible before in the sense that:

Services are software components with well-defined interfaces that are implementation-independent. An important aspect of SOA is the separation of the service interface (the what) from its implementation (the how). Such services are consumed by clients that are not concerned with how these services will execute their requests.

Services are self-contained (perform predetermined tasks) and loosely coupled (for independence)

Services can be dynamically discovered Composite services can be built from aggregates of other services SOA uses the find-bind-execute paradigm as shown in Figure. In this paradigm, service providers register their service in a public registry. This registry is used by consumers to find services that match certain criteria. If the registry has such a service, it provides the consumer with a contract and an endpoint address for that service.



SOA-based applications are distributed multi-tier applications that have presentation, business logic, and persistence layers. Services are the building blocks of SOA applications. While any functionality can be made into a service, the challenge is to define a service interface that is at the right level of abstraction. Services should provide coarse-grained functionality.

1.8 WEB SERVICES DEVELOPMENT LIFE CYCLE

The software and web development life cycle adheres to a specific standard that has to be followed to move in the right direction. There are frameworks, methodologies, modelling tools, and languages involved.

The Web Development Life Cycle is a method that outlines the stages involved in building websites and web applications. It provides a structured approach, ensuring optimal results throughout the development process.

1.8.1 7 Steps

The web development life cycle typically consists of several stages that web developers follow to create, deploy, and maintain a website or web application. While the exact number and names of the stages can vary depending on the development process followed, the seven stages you provided cover the major phases of web development. Let's briefly go through each stage:

1. Gathering Relevant Information:
This stage involves understanding the project's goals, objectives, and requirements. The development team communicates with clients or stakeholders to gather information about the target audience, purpose of the website, desired functionalities, and design preferences. This stage sets the foundation for the entire development process.

2. Planning - Sitemap and Wireframe:
During this phase, the project plan is formulated, including defining the website's structure through a sitemap. A sitemap provides an overview of the site's pages and their hierarchical relationship. Additionally, wireframes are created, which are basic sketches of the website's layout, outlining the placement of elements without incorporating design details.

3. Design & Layout:
In the design stage, graphic designers and UI/UX experts work on creating the visual elements of the website. The website's layout, color schemes, typography, and user interface are designed to create an appealing and user-friendly experience.

4. Content Creation:
Content creation involves producing and organizing the textual, visual, and multimedia elements that will be part of the website. This includes writing and gathering content such as text, images, videos, and other media that align with the website's purpose and target audience.

5. Development:
The actual development of the website takes place in this phase. Web developers use various programming languages, such as HTML, CSS, JavaScript, and backend technologies, to implement the design and functionality of the website. Databases may also be integrated during this stage.

6. Testing, Review, and Launch:
The testing phase is crucial to ensure that the website functions as intended and is free from errors or bugs. Quality Assurance (QA) testers conduct thorough testing, including usability testing, performance testing, security testing, and compatibility testing. After thorough review and any necessary adjustments, the website is ready for launch.

7. Maintenance and Updation:
Once the website is launched, it requires ongoing maintenance and updates. Regular monitoring and updates are performed to address any

issues, ensure security, and keep the website up-to-date with the latest technologies and content changes.

The web development life cycle is iterative, and feedback from users and clients may lead to improvements or changes in the website over time. Following a structured development process ensures a well-planned and successful website that meets the client's requirements and delivers a positive user experience.

1.9 DEVELOPING AND CONSUMING SIMPLE WEB SERVICES ACROSS PLATFORM

Developing and consuming simple web services across different platforms involves creating web services that can communicate and exchange data seamlessly between various programming languages and platforms. This can be achieved by following standards like SOAP (Simple Object Access Protocol) or REST (Representational State Transfer) and ensuring the data is encoded in a universal format like XML or JSON. Here's a high-level guide to achieving this:

1. Choosing a Web Service Protocol:
Decide whether you want to use SOAP or REST as the protocol for your web services. SOAP is more rigid and uses XML for data exchange, while REST is more flexible and commonly uses JSON, though it can also use XML.

2. Designing the Web Service:
Define the service endpoint interface (SEI) for your web service. This is a contract that specifies the operations the service supports, along with their input and output parameters. The SEI is typically defined using WSDL (Web Services Description Language) for SOAP-based web services or a simple URL for RESTful web services.

3. Implementing the Web Service:
In the server-side code, implement the SEI. For SOAP web services, use libraries like JAX-WS (Java API for XML Web Services) for Java or WCF (Windows Communication Foundation) for .NET. For RESTful web services, frameworks like JAX-RS for Java or ASP.NET Web API for .NET can be used.

4. Exposing the Web Service:
Make the web service available on a public URL or endpoint so that clients can access it over the internet. For SOAP web services, this typically involves deploying the service on a web server that supports SOAP (e.g., Apache Axis for Java or IIS for .NET). For RESTful web services, a web server with the necessary framework support will suffice.

5. Consuming the Web Service:
On the client-side, use the appropriate programming language and libraries to consume the web service. Most modern programming languages have built-in support or third-party libraries for making web service calls.

6. Generating Client Code:
In some cases, you can use tools to generate client-side code based on the WSDL (for SOAP) or Swagger/OpenAPI specifications (for REST) of the web service. These tools create client code to interact with the web service, making it easier to consume the service.

7. Making Requests and Handling Responses:
Send HTTP requests to the web service endpoint using libraries like `curl`, `HttpClient` (for Java), or `requests` (for Python). For SOAP, use the appropriate methods from the generated client code. Handle the responses accordingly to retrieve the data or process any errors.

8. Parsing Data:
Once the response is received from the web service, parse the data accordingly. For JSON, use built-in or third-party libraries to deserialize the JSON response into objects or data structures in your programming language. For XML (in SOAP), use XML parsing libraries.

By following these steps and adhering to industry standards, you can successfully develop and consume web services across different platforms, allowing seamless communication and data exchange between various systems and programming languages.

1.10 SUMMARY

- Web services are a set of technologies and standards used for communication and data exchange between different software applications and systems over the internet.
- A distributed system is characterized as a collection of (probably heterogeneous)networked computers, which communicate and coordinate their actions by passing messages.
- Internet protocols are a set of rules and conventions that dictate how devices communicate and exchange data over the Internet.
- The Open Systems Interconnection (OSI) reference model is a conceptual framework used to understand and describe how different networking protocols and technologies interact and work together in a networked communication system.
- Middleware refers to software that acts as an intermediary or bridge between different applications, systems, or components within a distributed computing environment.

- The client-server model is a computing architecture that describes the relationship and interaction between two types of entities: the "client" and the "server."
- XML is an extensible markup language used for the description and delivery ofmarked-upelectronic text over the Web.
- SOAP (Simple Object Access Protocol) is a messaging protocol used for exchanging structured information in the form of XML-based messages between web services over computer networks, typically over HTTP or SMTP.
- JAX-WS (Java API for XML Web Services) is a Java API that allows developers to build and deploy web services using XML-based SOAP messaging.
- Registering and discovering web services are important steps in building a service-oriented architecture (SOA) and enabling service discovery, which allows clients to find and interact with available web services dynamically.
- SOA is an architectural style for building software applications that use services available in a network such as the web.
- The software and web development life cycle adheres to a specific standard that has to be followed to move in the right direction.

Developing and consuming simple web services across different platforms involves creating web services that can communicate and exchange data seamlessly between various programming languages and platforms.

1.11 QUESTIONS

- What are Web Services?
- What are the types of web services?
- Explain Distributed computing infrastructure.
- Write a short note on Internet Protocols.
- Differentiate between The Open Systems Interconnection reference model and Internet protocols
- What is Middleware? Explain.
- Write a short note on XML.
- Explain Web Services Development Life Cycle
- What is SOAP?
- What is Web Services Development Life Cycle?

1.12 REFERENCES

- https://www.signitysolutions.com/blog/web-development-life-cycle
- https://www.oracle.com/technical-resources/articles/javase/soa.html
- http://www.nortonaudio.com/Ficheiros/Web.services...principles.and.technology.pdf

❖❖❖❖

2

THE REST ARCHITECTURAL STYLE

Unit Structure :

2.0 OBJECTIVE

- To understand the concept of REST Architectural style.
- To study the core architectural elements of the RESTful system.
- To learn the Description and discovery of RESTful web services.
- To understand the JSON message format.

2.1 INTRODUCTION

- REST stands for REpresentational State Transfer and API stands for Application Program Interface.
- REST is a software architectural style that enables the set of rules to be used for creating web services.
- Web services which come after the REST architectural style are called RESTful web services.
- It permits requesting systems to access and manipulate web resources by using a uniform and predefined set of rules.
- Interconnection in REST based systems happens through the Internet's Hypertext Transfer Protocol (HTTP).

A Restful system consists of a:

- Client who requests for the resources.
- Server who has the resources.

2.2 INTRODUCING HTTP

- Hypertext Transfer Protocol (HTTP) is the base of data communication for WWW.
- This protocol shows how messages are formatted, transmitted, and processed through the Internet.

A. HTTP versions

- There are three versions of HTTP which have been regularly developed over time.
- HTTP 0.9 was the first version which is documented, which was released in the year 1991. The 0.9 version was very primitive and supported only the GET method.
- HTTP 1.0 was released in 1996 with additional features and corrections for the shortcomings in the prior release. HTTP 1.0 supported extra request methods such as GET, HEAD, and POST.
- The next release was HTTP/1.1 in 1999. This was the revision of HTTP/1.0. This version is in ordinary use today.
- HTTP 2 (HTTP 2.0) is the next designed version. It is mainly focused on how the data is wrapped and transported between the client and the server.

B. HTTP request-response model

Example:

GET /index.html HTTP/1.1

- The general method for the request line is an HTTP command, followed by the resource to retrieve, and the HTTP version adaptable with the client.
- The client means it can be any type of an application that recognizes HTTP, this example refers to a web browser as the client.
- The request and header fields must end with a carriage return character followed by a line feed character.
- In the preceding example, the browser instructs the server to get the index.html file through the HTTP 1.1 protocol.
- The header fields are separated by colon (:) key value pairs in the plain text format, and terminated by a carriage return followed by a line feed character.
- The header fields in the request line, such as the acceptable content types, languages, and connection type, are the operating parameters for an HTTP transaction.
- The server used this information while producing the response for the request.

- An empty blank line is used at the end of the header which indicates the end of the header portion in a request.
- The end part of an HTTP request is the HTTP body. Usually, the body is kept blank unless the client has some data to submit to the server.
- In this example, the body portion is empty as this is a GET request for retrieving a page from the server.

Uniform resource identifier

- The word uniform resource identifier (URI) is used very often. A URI is a text that pinpoints any resource or name on the Internet.
- URI classifies as a Uniform Resource Locator (URL) if the text used for recognizing the resource also holds the means for accessing the resource such as HTTP or FTP.
- The following is one such example:

  https://www.packet.com/application-development

- In general, all URLs are URIs. To learn more about URIs, visit

  http://en.wikipedia.org/wiki/Uniform_resource_identifier.

C. HTTP request methods

- The HTTP GET request method is used for recovering a page from the server. More request methods which are the same as GET are available with HTTP, each accomplish specific actions on the target resource.
- The set of common methods for HTTP/1.1 is shown in the following table:

Method	Description
GET	This method is used for retrieving resources from the server by using the given URI.
HEAD	This method is the same as the GET request, but it only transfers the status line and the header section without the response body.
POST	This method is used for posting data to the server. The server stores the data (entity) as a new subordinate of the resource identified by the URI. If you execute POST multiple times on a resource, it may yield different results.
PUT	This method is used for updating the resource pointed at by the URI. If the URI does not point to an existing resource, the server can create the resource with that URI.
DELETE	This method deletes the resource pointed at by the URI.
TRACE	This method is used for echoing the contents of the received request. This is useful for the debugging purpose with which the client can see what changes (if any) have been made by the intermediate servers.
OPTIONS	This method returns the HTTP methods that the server supports for the specified URI.
CONNECT	This method is used for establishing a connection to the target server over HTTP.
PATCH	This method is used for applying partial modifications to a resource identified by the URI.

Table 1.set of common methods for HTTP/1.1

D. Representation of content types using HTTP header fields

- The HTTP header parameters name-value pairs that define the utilizing parameters of an HTTP transaction.
- The header parameter used for expressing the content types present in the request and the response message body.
- The Content-Type header in an HTTP request or response expresses the content type for the message body.
- The Accept header in the request informs the server the content types that the client is looking for in the response body.
- The content types are characterized using the Internet media type. The Internet media type (MIME type) specifies the type of data that a file contains.
- Example: Content-Type: text/html

  This header shows that the body content is expressed in the html format. The format of the content type values is a primary type or subtype followed by an optional semicolon (:) delimited attribute-value pairs called as parameters. The Internet media types are mainly classified in to the following categories on the foundation of the primary (or initial) Content-Type header:

Text	This type indicates that the content is plain text and no special software is required to read the contents.
Multipart	As the name indicates, this type consists of multiple parts of the independent data types.
Message	This type encapsulates more messages. It allows messages to contain other messages or pointers to other messages.
Image	This type represents the image data. For instance, Content-Type: image/png indicates that the body content is a .png image.
Audio	This type indicates the audio data.
Video	This type indicates the video data.
Application	This type represents the application data or binary data.

Table 2. Content-Type header

E. HTTP status codes

- Each HTTP request, the server returns a status code specifying the processing status of the request..
- A basic of status codes will definitely help for designing RESTful web services:
  - **1xx Informational:** This series of status codes shows informational content. Hence the request is received and processing is proceeding.

Here are the frequently used informational status codes:

100 Continue:	This code indicates that the server has received the request header and the client can now send the body content.
101 Switching Protocols:	This code indicates that the server is OK for a protocol switch request from the client.
102 Processing:	This code is an informational status code used for long running processing to prevent the client from timing out. This tells the client to wait for the future response, which will have the actual response body.

- **2xx Success:** This series of status codes specify the successful processing of requests. Some of the often used status codes in this class are as follows:

200 OK:	This code indicates that the request is successful and the response content is returned to the client as appropriate.
201 Created:	This code indicates that the request is successful and a new resource is created.
204 No Content:	This code indicates that the request is processed successfully, but there's no return value for this request. For instance, you may find such status codes in response to the deletion of a resource.

- **3xx Redirection:** This series of status codes shows that the client needs to perform further actions to logically end the request. A oftenly used status code in this class is as follows:

304 Not Modified:	This status indicates that the resource has not been modified since it was last accessed

- **4xx Client Error:** This series of status codes represent an error in processing the request. Some of the frequently used status codes in this class are as follows:

400 Bad Request:	This code indicates that the server failed to process the request because of the malformed syntax in the request. The client can try again after correcting the request.
401 Unauthorized:	This code indicates that authentication is required for the resource. The client can try again with the appropriate authentication.
403 Forbidden:	This code indicates that the server is refusing to respond to the request even if the request is valid. The reason will be listed in the body content if the request is not a HEAD method.
404 Not Found:	This code indicates that the requested resource is not found at the location specified in the request.
405 Method Not Allowed:	This code indicates that the HTTP method specified in the request is not allowed on the resource identified by the URI.
408 Request Timeout:	This code indicates that the client failed to respond within the time frame set on the server.
409 Conflict:	This code indicates that the request cannot be completed because it conflicts with some rules established on resources, such as validation failure.

- **5xx Server Error:** This series of status codes shows server failures while computing a valid request. Below is one of the frequently used status codes in this class:

500 Internal Server Error:	This code indicates a generic error message, and it tells that an unexpected error occurred on the server and the request cannot be fulfilled.

F. The evolution of RESTful web services

- A web service is one of the very well-liked methods of communication between the client and server applications over the Internet world.
- In simple name, web services are web application components that can be published, found, and used on the web. Consistently, a web service has an interface explaining the web service APIs, which is called Web Services Description Language (WSDL).
- A WSDL file can be easily handled by machines, which blows out the merger complexities. Other systems linked with the web service by using Simple Object Access Protocol (SOAP) messages.
- The contract for communication is operated by the WSDL exposed by the web service. Typically, communication happens over HTTP with XML in cooccurrence with other web-related standards.
- There are two main areas web services are used:
  - Numerous of the companies specialized in Internet-related services and products have opened their doors to developers using publicly available APIs. For example, companies like Google, Yahoo, Amazon, and Facebook are using web services to offer new products that depend on their massive hardware infrastructures. Google and Yahoo recommended their search services; Amazon offers its on-demand hosting storage infrastructure and Facebook offers its platform for targeted marketing and advertising drive. With the help of web services, these companies have opened the door for the formation of products that did not exist some years ago.
  - Web services are being used inside the enterprises to connect previously disjointed departments such as marketing and manufacturing. Each department or line of business (LOB) can uncover its business processes as a web service, which can be ingested by the other departments. By connecting more than one department to pass information by using web services, we begin to enter the territory of the Service-Oriented Architecture (SOA). The SOA is essentially a collection of services, each interacting to one another in a well-defined manner, in order to complete relatively large and logically complete business processes.

2.3 THE CORE ARCHITECTURAL ELEMENTS OF A RESTFUL SYSTEM

- A constant interface is fundamental to the architecture of any RESTful system. In plain words, this term refers to a generic interface to manage all linkedions between a client and a server in a unified way.
- All resources (or business data) involved in the client-server linkedions are dealt with by a fixed set of operations. The below are core elements that form a uniform interface for a RESTful system:
  - Resources and their identifiers
  - Representations of resources

- Generic linkedin semantics for the REST resources
- Self-descriptive messages
- Hypermedia as the engine of an application state

Resources

- A RESTful resource is whatever that is addressable over the Web.
- By addressable, resources that can be accessed and transferred between clients and servers. Subsequently, a resource is a logical, temporal mapping to a concept in the problem domain for which we are implementing a solution.

Here are some examples of the REST resources:

- A news story
- The temperature in IN at 4:00 p.m. IST
- A tax return stored in the IRS database
- A list of code revision history in a repository such as SVN or CVS
- A student in a classroom in a school
- A search result for a particular item in a Web index, such as Google

URI

- A URI is a string of characters used to recognize a resource over the Web.
- In simple words, the URI in a RESTful web service is a hyperlink to a resource, and it is the only for clients and servers to interchange representations.
- The client uses a URI to find the resources over the Web and then, sends a request to the server and looks through the response.
- In a RESTful system, the URI is not meant to swap over time as it may break the contract between a client and a server.
- More essentially, even if the fundamental infrastructure or hardware changes (for example, swapping the database servers) for a server hosting REST APIs, the URIs for resources are expected to remain the same as long as the web service is up and running.

The representation of resources

- The characterization of resources is what is sent back and forth between clients and servers in a RESTful system.
- A characterization is a temporal state of the actual data located in some storage device at the time of a request.
- In general terms, it is a binary stream together with its metadata that shows how the stream is to be consumed by the client.
- The metadata can also consist of extra information about the resource, for example, validation, encryption information, or extra code to be executed during runtime.

- All over the life of a web service, there may be a variety of clients requesting resources.
- Different clients can consume different representations of the same resource. Therefore, a representation can take various forms, such as an image, a text file, an XML, or a JSON format.
- All clients will use the same URI with appropriate Accept header values for accessing the same resource in different representations.
- For the human-generated requests with a web browser, a representation is typically in the form of an HTML page.
- For automated requests from the other web services,clarity is not as important and a more efficient representation, such as JSON or XML, can be used.

Generic linkedin semantics for REST resources

- The generics of linkedin semantics and self-descriptive messages followed for the client-server communication in a RESTful system.
- Developing RESTful web services is similar to what we have been doing up to this point with our web applications.
- In a RESTful web service, resources are interchange between the client and the server, which enables the business entities or data. HTTP specifies methods or actions for the resources.
- The most frequently used HTTP methods or actions are POST, GET, PUT, and DELETE.
- This clearly simplifies the REST API design and makes it more readable.
- On the other hand, in traditional application development, we can have countless actions with no naming or implementation standards.
- This may call for more development effort for both the client and the server, and make the APIs less readable.
- In a RESTful system, It is easy to map our CRUD actions on the resources to the relevant HTTP methods such as POST, GET, PUT, and DELETE. This is shown in the following table:

Data action	**HTTP equivalent**
CREATE	POST or PUT
READ	GET
UPDATE	PUT or PATCH
DELETE	DELETE

- There are more HTTP methods available, but they are less frequently used in the context of RESTful implementations. Among these less frequent methods, OPTIONS and HEAD are used more often than others. these two method types:
  - **OPTIONS**: This method is used by the client to find the options or actions related with the target resource, without causing any action on the resource or retrieval of the resource
  - **HEAD**: This method can be used for recovering information about the entity without having the entity itself in the response
- RESTful web services are networked applications that handle the state of resources. In this context, resource controls means resource creation, retrieval, update, and deletion.
- RESTful web services are not limited to just these four basic data manipulation ideas.
- They can even be used for executing business logic on the server, but remember that every result must be a resource representation of the domain at hand.
- A uniform interface leads all the aforementioned abstractions into focus.

The HTTP GET method

- This method is used to retrieve resources. Before going into details about the actual mechanics of the HTTP GET request, Here first we need to find what a resource is in the context of our web service and what type of representation we are exchanging.
- Example of a RESTful web service handling department details for an organization. For this the JSON representation of a department shown below:

  {"departmentId":10,"departmentName":"IT","manager":"John Chen"}
- The JSON description of the list of departments looks like the following:

```
[{"departmentId":10,"departmentName":"IT","manager":"John
Chen"},
{"departmentId":20,"departmentName":"Marketing","manager":"A
meyaJ"},
{"departmentId":30,"departmentName":"HR","manager":"Pat
Fay"}]
```

- With our representations defined, we now assume URIs of the form http://www. packet.com/resources/departments to access a list of departments, and http://www.packet.com/resources/departments/{name} to access a specific department with a name (unique identifier).

The HTTP POST method

- The POST method is used to generate resources. For creating a department, need to use the HTTP POST method. One more time, the URI to create a new department in our example is http://www.packet.com/resources/departments.
- The method type for the request is placed by the client. Consider that the Sales department does not exist in our list, and we want to add it to the list. The Sales data representation shown below:

  {"departmentName":"Sales","manager":"TonyGreig"}

The HTTP PUT method

- The PUT method is used for updating resources.
- To update a resource,
  - first need its representation in the client;
  - At the client level, update the resource with the new value that we required
  - Finally, update the resource by using a PUT request together with the representation as its payload.
- In this example, add a manager to the Sales department that we created in the previous example. Our primary representation of the Sales department is as follows:

  {"departmentId":40,"departmentName":"Sales","manager":"Tony Greig" }
- Let's update the manager for the Sales department; our representation is as follows:

  {"departmentId":40,"departmentName":"Sales","manager":"Ki Gee"}

The HTTP DELETE method

- The DELETE method is used to delete or remove the resource.
- Example, delete a resource by making use of the same URI that we used earlier. Assume that to delete the Sales department from the data storage. We send a DELETE request to our service with the given URI:

  http://www.packet.com/resources/departments/Sales.

2.4 DESCRIPTION AND DISCOVERY OF RESTFUL WEB SERVICES

- WSDL stands for Web Services Description Language which is used for describing the functionality offered by a SOAP web service. For a

SOAP web service, this is an extensively accepted standard and is supported by numerous enterprises today.

- In distinction, for RESTful web services, different metadata formats are used by various enterprises.
- Below goals in common between all these metadata formats for RESTful APIs, even through they differ in their syntax and semantics:
  - Entry points for the service
  - Resource paths for accessing each resource
  - HTTP methods enables to access these resources, such as GET, POST, PUT, and DELETE
  - Extra parameters that need to be provided with these methods, such as pagination parameters, while reading large collections
  - Format types used for showing the request and response body contents like JSON, XML, and TEXT
  - Status codes and error messages returned by the APIs
  - Human readable documentation for REST APIs, which includes the documentation of the request methods, input and output parameters, response codes (success or error), API security, and business logic Some of the popular metadata formats used for describing REST APIs are Web Application Description Language (WADL), Swagger, RESTful API Modeling Language (RAML), API Blueprint, and WSDL 2.0.

2.5 JAVA TOOLS AND FRAMEWORKS FOR BUILDING RESTFUL WEB SERVICES

- This is a famous Java based framework and tools for building RESTful systems.
- The Java API for RESTful web services (JAX-RS) is the Java API for creating RESTful web services following the REST architectural pattern.
- JAX-RS is a part of the Java Platform Enterprise Edition (Java EE) platform and is designed to be a standard and portable solution.
- There are numerous reference implementations available for JAX-RS today. The most popular implementations are Jersey, Apache CXF, RESTEasy, and Restlet.
- Except JAX-RS-based frameworks, there are some promising nonstandard Java REST frameworks on the market. Some such frameworks are as given below:

- One equivalent framework is RESTX, which is an open source Java REST framework and is primarily focused on the server-side REST API development. This is relatively new on the market and simplifies the REST API development.
- Spark is the second framework that falls into this type. It is a Java web framework with support for building REST APIs. Spark 2.0 is built using Java 8, leveraging all the latest improvements of the Java language.
- Play is the third framework worth mentioning in this category. It is a Java based web application framework with inherent support for building RESTful web services.

2.6 JSON MESSAGE FORMAT AND TOOLS AND FRAMEWORKS AROUND JSON

The JSON data syntax

- The JSON format is very simple by design. It is represented by the following two data structures:
- An unordered collection of name-value pairs (an object):
- The attributes of an object and their values are represented in the name-value pair format; the name and the value in a pair is separated by a colon (:).
- Names in an object are strings, and values may be of any of the valid JSON data types such as number, string, Boolean, array, object, or null.
- Each name:value pair in a JSON object is separated by a comma (,). The entire object is enclosed in curly braces ({ }).
- For instance, the JSON representation of a department object is as follows:{"departmentId":10,"departmentName":"IT", manager":"John Chen"}
- This example shows how you can represent various attributes of a department, such as departmentId, departmentName, and manager, in the JSON format.
- An ordered collection of values (representing an array): Arrays are enclosed in square brackets ([ ]), and their values are separated by a comma (,). Each value in an array may be of a different type, including another array or an object.
- The following example illustrates the use of an array notation to represent employees working in a department. You may also see an array of locations in this example:

```
{"departmentName":"IT",
"employees":[
{"firstName":"John", "lastName":"Chen"},
{"firstName":"Ameya", "lastName":"Job"},
{"firstName":"Pat", "lastName":"Fay"}
],
"location":["New York", "New Delhi"]
}
```

Basic data types available with JSON

- List of the basic data types available with JSON:

○ **Number:** This type is used for storing a signed decimal number that may optionally contain a fractional part. Both integer and floating point numbers are represented by using this data type.

Example : The decimal data type for storing totalWeight: {"totalWeight": 123.456}

○ **String:** This type represents a sequence of zero or more characters. Strings are surrounded with double quotation marks and support a backslash escaping syntax. Example of the string data type:

{"firstName": "Jobinesh"}

○ **Boolean:** This type represents either a true or a false value. The Boolean type is used for representing whether a condition is true or false, or to represent two states of a variable (true or false) in the code.

Example representing a Boolean value:

{"isValidEntry": true}

○ **Array:** This type represents an ordered list of zero or more values, each of which can be of any type. In this representation, comma-separated values are enclosed in square brackets.

Example represents an array of fruits:

{"fruits": ["apple", "banana", "orange"]}

○ **Object:** This type is an unordered collection of comma-separated attribute- value pairs enclosed in curly braces. All attributes must be strings and should be distinct from each other within that object.

Example illustrates an object representation in JSON:

{"departmentId":10, "departmentName":"IT", "manager":"John Chen"}

- **null:** This type indicates an empty value, represented by using the word null.

  Example uses null as the value for the error attribute of an object:

  {"error":null}

A sample JSON file representing employee

- Here is a sample JSON document file called "emp-array.json", which contains the JSON array of the employee objects. The content of the file is as follows:

```
{"employeeId":100,"firstName":"John","lastName":"Chen",
"email":"john.chen@xxxx.com","hireDate":"2008-10-16"},
{"employeeId":101,"firstName":"Ameya","lastName":"Job",
"email":"ameya.job@xxx.com","hireDate":"2013-03-06"},
{"employeeId":102,"firstName":"Pat","lastName":"Fay",
"email":"pat.fey@xxx.com","hireDate":"2001-03-06"} ]
```

Processing JSON data

- While using Java RESTful web service frameworks, like JAX-RS, for building RESTful web APIs, the serialization and deserialization of the request and response messages will be taken care of by the framework.
- However, if our requirements do not meet then the JSON structure and tools for processing JSON will definitely help us for the framework. The following diagram shows the role of the JSON marshaling and unmarshalling

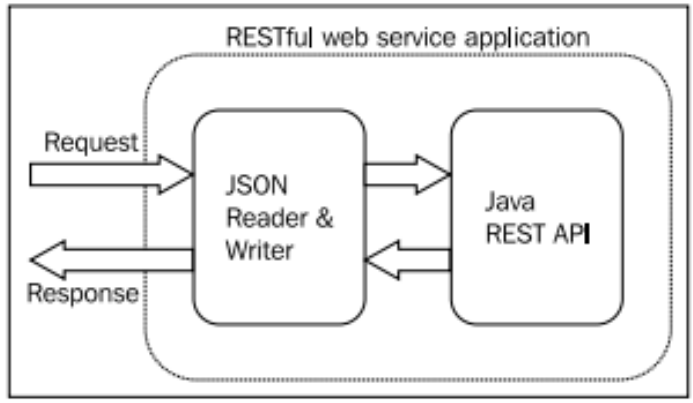


Figure 2. JSON marshaling and unmarshalling

- Two Generally adopted programming models for processing JSON are as follows:

  - **Object model**: In this model, the entire JSON data is read into memory in a tree format. This tree can be traversed, analyzed, or modified with the suitable APIs.

  - **Streaming model:** The term streaming is very generic in meaning and can be used in various aspects.

2.7 SUMMARY

1. This chapter is intended to give an overview of RESTful web services.
2. This is essential for an easy understanding of what we will learn in the rest of the book.
3. We will examine the most popular Java tools and frameworks available for building a RESTful web service along with numerous real-life examples and code samples.
4. Processing models for the JSON content and some of the popular Java-based JSON processing frameworks available today.
5. The JSON-based request and response messages are bound to the Java model while building REST APIs.

2.8 REFERENCE FOR FURTHER READING

1. Web Services: Principles and Technology, Michael P. Papazoglou, Pearson Education Limited, 2008
2. RESTful Java Web Services, JobineshPurushothaman, PACKT Publishing,2nd Edition, 2015

2.9 UNIT END EXERCISES

1. Explain the HTTP request-response model? Explain the HTTP Request methods.
2. Explain the different status code under HTTP request.
3. Explain the JSON message format and tool?
4. Write a short note on Description and discovery of RESTful web services

3

THE RESTFUL WEB SERVICES

Unit Structure :

3.0 OBJECTIVES

- To understand the building RESTful web services with JAX-RS APIs
- To study description and discovery of RESTful Web Services.
- Design guidelines for building RESTful web services.
- To understand the Secure RESTful web services

3.1 INTRODUCTION

- We will learn how to build a simple end-to-end RESTful web service by using JAX-RS.
- Securing web services is the calculated control of resources.
- Example, Google's web service API limited the number of queries a registered user could execute daily. Likewise, many other API vendors restrict the access of their APIs.
- The popular solutions available today for describing, producing, consuming, and visualizing RESTful web services.
- Here we are using API documentation solutions available in the market.

3.2 BUILD RESTFUL WEB SERVICES WITH JAX-RS APIS

- Setting up the environment

This example uses the following software and tools:

- Java SE Development Kit 8 or newer
- NetBeans IDE 8.0.2
- Glassfish Server 4.1 or newer
- Maven 3.2.3 or newer
- Oracle Database Express Edition 11g Release 2 or newer with HR sample database schema
- Oracle Database JDBC Driver

- Make sure that machine has all tools ready before starting installation with this setup. In this setup, we will build a RESTful web service by using the JAX-RS APIs.
- Use Maven as a build tool for sample applications as it does a great job in the dependency management department for the application and provides a standard structure for the source code.
- NetBeans has great support for building Maven-based applications, and this is one of the reasons for us to choose NetBeans as the IDE.
- Once the development environment is set up, we are ready to use NetBeans IDE for application development.

Building a simple RESTful web service application using NetBeans IDE

- To create a JAX-RS application, follow below given steps:

1. Launch NetBeans IDE.
2. In the main toolbar, navigate to File | New Project.
3. New Project dialog screen, navigate to Maven | Web Application for building the RESTful web service. Proceed with the next screen by clicking on the Next button.
4. In the Name and Location screen, enter Project Name, Project Location (storing the source), Group Id, Version (Maven project), and Package (Java source files) as follows:

Project Name: rest-topic3-service

Group Id:com.packtdata

Package: com.packtdata.rest.ch3.service

Note to the following screenshot for the values used for this e.g:

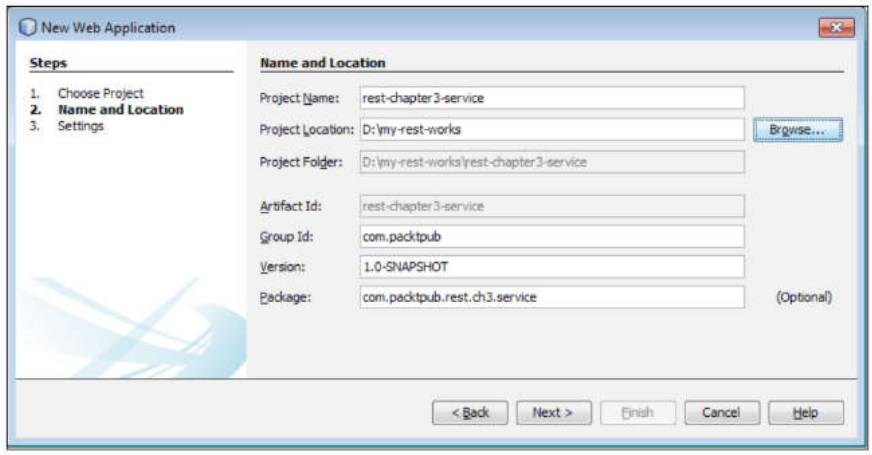


Figure 1.RESTful web service application

- Enter the identical values as in the preceding screenshot.
- After setting all things, click on Next to continue to the Settings screen in the wizard.

5. On the Settings screen, select GlassFish Server that have installed along with NetBeans IDE as Server for running ,r JAX-RS application, and then click on Java EE 7 Web as the Java EE version for the application that , build.

6. The server list on this output screen may appear empty if , have not configured any server for the IDE yet. To add a new server reference, perform the following steps:

   i. Click on the Add button. In the Add Server type wizard, select GlassFish as the server and click on Next to continue the wizard.

   ii. Setting Server Location to the folder where , installed GlassFish. Select Local Domain and click on Next to continue the wizard.

   iii. On the Domain Name screen, enter domain1 in the Domain field (which is the default one) and localhost in the Host field. Click on Finish to complete the server creation.

   iv. Now, IDE will show the Settings screen once again where , can choose GlassFish as the server and Java EE 7 Web as the Java EE version.

7. You can now click on the Finish button to finish the project configuration wizard. NetBeans now setting up a Maven-based web project for , as shown in the following screenshot:

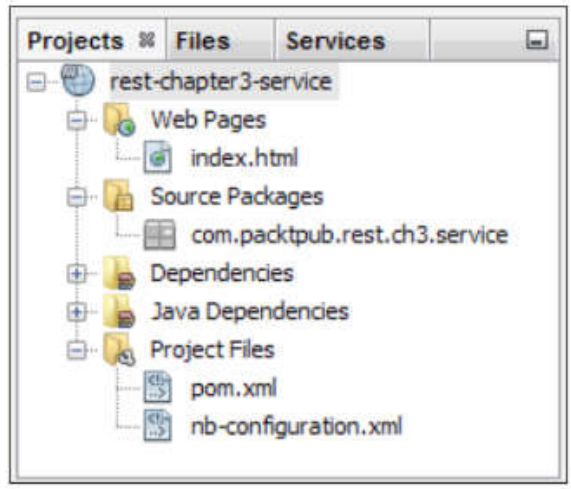


Figure 2. Maven-based web project

8. The next step is to build a simple RESTful web service implementation by using a POJO class to get a feel of the JAX-RS APIs. To build a POJO class, , can right-click on the project and navigate to New | Java Class in the menu. In the New Java Class editor, enter DepartmentService in the Class Name field and enter com.packtdata. rest.ch3.service in the Package field. This class will hold the service implementation for this example.

9. We will add @Path("departments") to this class so that DepartmentService becomes a REST resource class and responds to the REST API calls with the URI path fragment \departments. Let's add a simple helloWorld() method to this class and add @Path("hello") to this method. Addon the @GET annotation to designate this method to respond to the HTTP GET methods. The DepartmentService class now looks like the following: package com.packtdata.rest.ch3.service;

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
importjavax.ws.rs.core.MediaType;
@Path("departments")
public class DepartmentService{
@GET
@Path("hello")
@Produces(MediaType.APPLICATION_JSON)
public String helloWorld(){
return "Hello world";
}
}
```

10. For configuring resources, add a REST configurations class, which extends javax.ws.rs.core.Application. This class characterizes the components of a JAX-RS application and supplies additional metadata. The configuration class shown below:

```
package com.packtdata.rest.ch3.jaxrs.service;

import java.util.Set;

import javax.ws.rs.core.Application;

@javax.ws.rs.ApplicationPath("webresources")

public class RestAppConfig extends Application {

// Get a set of root resource and provider classes.

@Override

public Set<Class<?>>getClasses() {

Set<Class<?>> resources =

new java.util.HashSet<>();

resources.add(com.packtdata.rest.ch3.service.

DepartmentService.class);

return resources;

}

}
```

11. To deploy and run the RESTful web service application, , can right-click on the rest-topic3-service project and click on Run. This is building and deploying the application to the GlassFish server integrated with NetBeans IDE.

12. To test the desired REST API, right-click on the appropriate HTTP methods, as shown in the following screenshot, and select Test Resource Uri. These measures will open up the default browser with the response content returned by the REST call. The URI for accessing the helloWorldRESTful web API will look like http://localhost:8080/rest-topic3-service/ webresources/departments/hello.

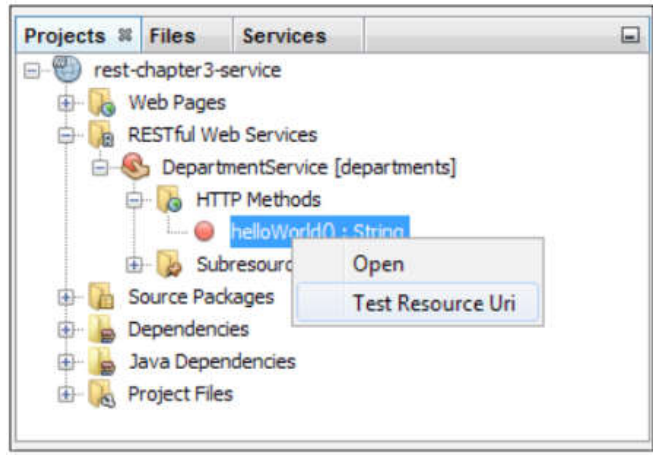


Figure 3. Projects Explorer

3.3 THE DESCRIPTION AND DISCOVERY OF RESTFUL WEB SERVICES

- WSDL (Web Services Description Language) is used for describing the functionality offered by a SOAP web service. In SOAP web service, it is a widely accepted standard and is kept by many enterprises today.
- In contrast, for RESTful web services, different metadata formats are used by various enterprises.
- Following objectives in common among all these metadata formats for RESTful APIs, although they differ in their syntax and semantics:
  - Entry points for the service
  - Resource paths for accessing each resource
  - HTTP methods allowed to access these resources, such as GET, POST, PUT, and DELETE
  - Additional parameters that need to be supplied with these methods, such as pagination parameters, while reading large collections
  - Format types used for representing the request and response body contents such as JSON, XML, and TEXT
  - Status codes and error messages returned by the APIs
  - Human readable documentation for REST APIs, which includes the documentation of the request methods, input and output parameters, response codes (success or error), API security, and business logic The Well liked metadata formats used for describing REST APIs are Web Application Description Language (WADL), Swagger, RESTful API Modeling Language (RAML), API Blueprint, and WSDL 2.0.

3.4 DESIGN GUIDELINES FOR BUILDING RESTFUL WEB SERVICES

- REST is a style of software architecture and not a specification, for building extensible web services. Considering RESTful web services do not have any strict specifications for designing and building APIs, There are many interpretations for how the RESTful web API should work.
- Here we will learn standards, best practices, conventions, and tips and tricks that we can apply to RESTful web service applications. The important points are as follows:
    - Naming RESTful web resources
    - Implementing partial response
    - Paging resource collection
    - Using HATEOAS in response representation
    - Versioning RESTful web APIs
    - Caching RESTful web API results
    - Microservice architecture style for RESTful web applications
    - Open Data Protocol with RESTful web APIs

Identifying resources in a problem domain

- The basic steps that , may need to take while building a RESTful web API for a specific problem domain are as follows:

1. Identify all possible objects in the problem domain. This can be done by identifying all nouns in the problem domain. Examples, if building an application to handle employees in a department, the obvious nouns are department and employee.
2. The next step is to recognize the objects that can be manipulated using the CRUD operations. These objects can be classified as resources. Note that, should be careful while choosing resources. Based on the usage pattern, , can classify resources as top-level and nested resources (which are the children of a top-level resource). Not required to expose all resources for use by the client; expose only those resources that are required for implementing the business use case.

Transforming operations to HTTP methods

- After identifying all the resources, the next step is to map the operations defined on the resources to the suitable HTTP methods.
- The most often used HTTP methods in RESTful web APIs are POST, GET, PUT, and DELETE.

- There is no one-to-one mapping between the CRUD operations states on the resources and the HTTP methods.
- Here are some ideas for finding the most suitable HTTP method for the operations that o perform on the resources:
  - GET: this method for reading a representation of a resource from the server. According to the HTTP specification, GET is a safe operation, which means that it is only deliberate for retrieving data, not for making any state changes. As this is an idempotent operation, multiple identical GET requests will act in the same manner. A GET method can return the 200 OK HTTP response code on the successful retrieving of resources. If there is any error, it can return an proper status code such as 404 NOT FOUND or 400 BAD REQUEST.
  - DELETE: this use method for deleting resources. After successful deletion, DELETE can return the 200 OK status code. According to the HTTP specification, DELETE is a self-duplicating operation. Note that when call DELETE on the same resource for the second time, the server may return the 404 NOT FOUND status code after all it was already deleted, which is different from the response for the first request. The change in response for the second call is perfectly valid here. However, multiple DELETE calls on the same resource produce the same result (state) on the server.
  - PUT: According to the HTTP specification, this method is idempotent. When a client calls the PUT method on a resource, the resource available at the given URL is completely replaced with the resource representation sent by the client. When a client uses the PUT request on a resource, it will send all the available properties of the resource to the server.
  - POST: This method is not self repeating. This method enables users to use the POST method to create or update resources when they do not know all the available attributes of a resource. Example, consider a framework where the identifier field for an entity resource is generated at the server when the entity is persisted in the data store. The POST method for creating such resources as the client does not have an identifier attribute while issuing the request. Example that illustrates this framework. In this example, the employeeID attribute is generated on the server:

    POST /hrapp/api/employees HTTP/1.1

    Host: packtdata.com

    {employee entity resource in JSON}

- On successful creation of a resource, it is recommended to return the status of 201 Created and the location of the newly created resource.
- This allows the client to access the newly created resource after. The sample response for the preceding example will look like as follows:

201 Created

Location: /hrapp/api/employees/1001

Understanding the difference between PUT and POST

- PUT method used for creating or updating a resource when the client has the full resource content available. In this instance, all values are with the client, and the server does not generate a value for any of the fields.
- POST method used for creating or updating a resource if the client has only partial resource content at hand.

Naming RESTful web resources

- Resources are an underlying concept in a RESTful web service.
- A resource shows an entity that is approachable via the URI that provides. The URI, which refers to a resource (called as RESTful web API), should have a logically meaningful name.
- Having meaningful names enhance the intuitiveness of the APIs, and therefore, their usability. Some of the broadly followed recommendations for naming resources are shown here:
- It is suggested to use nouns to name both resources and path segments that will appear in the resource URI. here try to avoid using verbs for naming resources and resource path segments. Using nouns to name a resource upgrades the readability of the corresponding RESTful web API, particularly when planning to release the API over the Internet for the mass public.
- Always use plural nouns to refer to a collection of resources. Make sure that , do not mix up singular and plural nouns while forming the REST URIs.
- For instance, to get all departments, the resource URI must look like /departments. To read a specific department from the collection, the URI becomes /departments/{id}. Following the convention, the URI for reading the details of the HR department identified by id=10 should look like /departments/10.
- The following table shows how can map the HTTP methods to the operations defined for the departments' resources:

Resource	GET	POST	PUT	DELETE
`/departments`	Get all departments	Create a new department	Bulk update on departments	Delete all departments
`/departments/10`	Get the `HR` department with `id=10`	Not allowed	Update the `HR` department	Delete the `HR` department

Table 1. HTTP methods

- While naming resources, esteem more concrete names over generic names.
- For instance, when reading all programmers' details of a software firm, it is preferable to have a resource URI of the form /programmers (which tells about the type of resource), over the much generic form /employees.
- This improves the perceptions of the APIs by clearly communicating the type of resources that it deals with.
- Keep the resource names that appear in the URI in the lowercase to upgrade the readability of the resulting resource URI.
- Resource names may include hyphens; avoid using underscores and other punctuation.
- If the entity resource is represented in the JSON format, field names used in the resource must obey to the following guidelines:
  - Use meaningful names for the setting properties
  - Follow camel-case naming convention: the first letter of the name is in lowercase, for example, departmentName
  - The first character must be a form of letter, an underscore (_), or a dollar sign ($), and the subsequent characters can be letters, digits, underscores, and/or dollar signs
  - Try to avoid using the reserved JavaScript keywords

Fine-grained and coarse-grained resource APIs

- While building a RESTful web API, try to avoid the chattiness of APIs. On the other hand, APIs should not be overly coarse-grained as well.
- Highly coarse-grained APIs become too complex to use because the response representation may contain a lot of information, all of which may not be used by a majority of ,r API clients.
- Example: the difference between fine-grained and coarse-grained approaches for building APIs. While building a very fine-grained API to read the employee details as follows:
  - API to read employee name: GET /employees/10/name
  - API to read employee address1: GET /employees/10/address1
  - API to read employee address2: GET /employees/10/address2
  - API to read employee e-mail: GET /employees/10/email

Using header parameter for content negotiation

- It is recommended to have an appropriate header parameter in the client request and in the server response to indicate how the entity body is serialized when transmitted over a wire.
- **Accept:** This request header field defines a list of acceptable response formats for the response, for example,

Accept: application/json,application/xml

The javax.ws.rs.client.WebTarget class allows , to specify the Accept header via the request() method for a JAX-RS client application

- **Content-Type:** This header field defines the type of the request or response message body content, for example, Content-Type: text/plain; charset=UTF-8

Multilingual RESTful web API resources

- If RESTful web API needs to return the resource representations in different languages depending upon the client locale, use the content negotiation offering in HTTP:
- Clients can use the Accept-Language request header to specify language preferences.
- While generating a response, the server is expected to translate the messages into the language supported by the client. The server can use the Content-Language entity-header field to describe the natural language(s) of the intended audience.

Representing date and time in RESTful web resources

- ISO 8601 is the International Standard for the representation of dates and times. It is recommended to use the ISO-8601 format for representing the date and time in RESTful web APIs.

Example for the ISO-8601 date and time: YYYY-MM-DDThh:mm:ss.sTZD (for example, 2015-06-16T11:20:30.45+01:00)

Implementing partial response

- Partial response refers to an optimization technique offered by the RESTful web APIs to return only the information (fields) required by the client. In this mechanism, the client sends the required field names as the query parameters for an API to the server, and the server trims down the default response content by removing the fields that are not required by the client.

Implementing partial update

- When a client changes only one part of the resource, optimize the entire update process by allowing the client to send only the modified

part to the server, thereby saving the bandwidth and server resources. RFC 5789 proposes a solution for this use case via a new HTTP method called PATCH.

The PATCH method has the following form:

PATCH /departments/10 HTTP/1.1[Description of changes]

3.5 SECURE RESTFUL WEB SERVICES

- Security is an important part of any enterprise application. In the era of cloud and the Internet of Things, controlling access to the application via the public web API is an essential requirement for any enterprise.
- The security implementation in a RESTful web service application decides who can access the RESTful web APIs and what they can do once they are logged in.
- The following concept use for Securing RESTful web services:
  - HTTP basic authentication
  - HTTP digest authentication
  - Securing RESTful web services with OAuth
  - Authorizing the RESTful web service accesses
  - Input validation

Securing and authenticating web services

- In the context of RESTful web services There are two forms of security:
- firstly, securing access to web services;
- Secondly, accessing web services on behalf of our users.
- Security has two essential elements: authentication and authorization.
  - Authentication: It is the process of verifying the identity of the user who is trying to access the application or web service. This is typically performed by obtaining user credentials, such as username and password, and validating them against the user details configured on the server.
  - Authorization: This is the process of verifying what an authenticated user is permitted to do in the application or service.

HTTP basic authentication

- Basic HTTP authentication works by sending the Base64 encoded username and the password as a pair in the HTTP authorization

header. The username and password must be sent for every HTTP request made by the client.

Building JAX-RS clients with basic authentication

- A normal client-server transaction, when using HTTP basic authentication, can take two forms. On one hand, a client makes a request to the server without authentication credentials.
- On the other hand, a client makes a request to the server with authentication credentials.
- When a client makes a request without authentication credentials, the server sends a response with an HTTP error code of 401 (unauthorized access).
- If the request is executed from a web browser, users see the ubiquitous Authentication Required browser popup, as shown below:

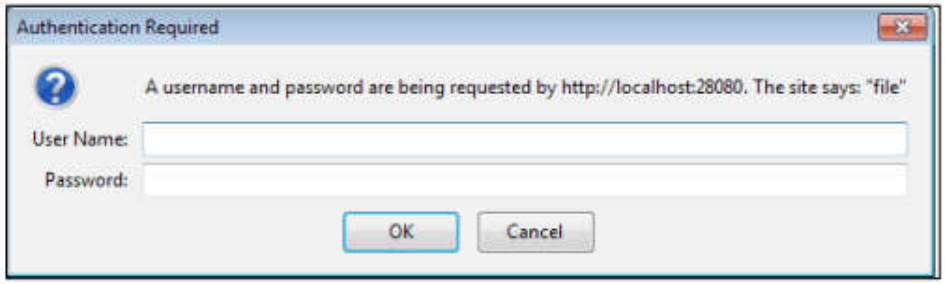


Figure 5. Basic authentication

- Users can then enter a valid username and password to complete the request. Note that the web browser keeps track of the 401 response, and is charged with sending the proper authentication credentials back with the original URI. This makes the transaction seamless for users. Now, if we were using a client other than a web browser, we would need to programmatically intercept the 401 response and then provide valid credentials to complete the original request.
- The second scenario that comes up while using HTTP basic authentication is when we do not wait for a server's 401 response, but provide authentication credentials at the beginning itself.

Securing JAX-RS services with basic authentication

- The basic authentication configuration depends on the web container being used.
- GlassFish server for every application that requires a Java web container; therefore, this example also assumes GlassFish as the target server for running the RESTful web APIs.

Configuring JAX-RS application for basic authentication

- First, secure a Java web application, deployed in a Java EE container, via basic authentication by making appropriate security entries in the web.xml descriptor file, and also in the container (vendor) specific deployment descriptor file(glassfish- web.xml)
- The first step is to modify the web.xml file to be shown as follows. Note that web.xml is found in the WEB-INF folder of , web application; generate a new one if it is found missing:
- Summary of the core elements that , saw in the preceding web.xml file:

○ The <security-constraint> element in web.xml is used to secure a collection of resources by restricting access to them with the appropriate URL mapping:

<web-resource-collection>	The <web-resource-collection>subelement describes the protected resources in the application, which are identified via URL patterns and HTTP methods.
<auth-constraint>	The subelement, <auth-constraint>, defines the user roles that are authorized to access constrained resources.
<user-data-constraint>	The subelement, <user-data-constraint>, defines how data is protected in the transmission channel. It takes the following values: CONFIDENTIAL, INTEGRAL, or NONE. CONFIDENTIAL and INTEGRAL are treated in the same way by the Java EE container, and these values imply the use of Transport Layer security (HTTPS) to all incoming requests matching the URL patterns present in <web-resource-collection>.

○ The element, <login-config>, defines the login configurations used in the application. The subelement, <realm-name>, refers to a collection of security information that is checked for authenticating the user when a secured page (resource) is accessed at runtime.

○ The realm name that enter should match with the security realm that , configured on the server. The subelement, <auth-method>, defines the authorization method used in the application. The possible values are as follows:

BASIC:	This is the HTTP basic authentication that we discussed a while ago.
DIGEST:	This is the same as HTTP basic authentication except that instead of a password, the client sends a cryptographic hash of user credentials along with the username.
FORM:	This uses an HTML form for login, having field names that match with specific convention. For instance, j_username and j_ password are used as names for the username and password fields respectively.
CLIENT-CERT:	This uses a certificate or other custom tokens in order to authenticate a user.

Defining groups and users in the GlassFish server

- Glass Fish allows defining users for the application using the concept of realms.
- A security realm can be treated as a mechanism that allows us to define users and groups.
- GlassFish offers various credential realms, including FileRealm, JDBCRealm, JNDIRealm, LDAPRealm, and so on. In this example we will use an existing file realm that comes with GlassFish by default.
- Steps for adding users and groups to the file realm in GlassFish:

1. Start the GlassFish server.
2. Log in as the administrator to the Admin interface.
3. Navigate to Configurations | server-config | Security | Realms | File. In this example, we use a file to store user information. In a real life scenario, , may use LDAP or RDBMS.
4. Click on the Manage User button at the top of the page.
5. On the File Users page, click on New and add a user and give a password. Set the Group List value as appropriate. In web.xml, we have configured Users as a group, so specify the same name as a value for Group List, for this example.
6. Click on OK to save change.

HTTP digest authentication

- The HTTP digest authentication authenticates a user based on a username and a password. However, unlike with basic authentication,

the password is not transmitted in clear text between the client and the server.

- Instead, the client sends a one way cryptographic hash of the username, password, and a few other security related fields using the MD5 message-digest hash algorithm.
- When the server receives the request, it regenerates the hashed value for all the fields as done by the client and compares it with the one present in the request.
- If the hashes match, the request is treated as authenticated and valid.

Securing RESTful web services with OAuth

- OAuth is an open standard for authorization, used by many enterprises and service providers to protect resources.
- OAuth solves a different security problem than what HTTP basic authentication has been used for.
- OAuth protocol allows client applications to access protected resources on behalf of the resource owner.

3.6 SUMMARY

- A number of RESTful web API metadata standards have emerged in the recent past.
- The popular RESTful web API documentation tools, namely WADL, RAML, and Swagger.
- To choose the right solution for organization, , should start by looking at client applications that access the APIs and their usage pattern, and then choose a tool that makes the API consumption easier, which may eventually improve the adoption of APIs by the customers.
- The best practices and coding guidelines that developers will find useful when building RESTful web service applications.
- Summarized the design guidelines, best practices, and coding tips that developers will find useful when building RESTful web services.
- We have learned how to build scalable and well-performing RESTful applications by using the JAX-RS and Jersey APIs.

3.7 REFERENCE FOR FURTHER READING

- Web Services: Principles and Technology, Michael P. Papazoglou, Pearson Education Limited, 2008
- RESTful Java Web Services, JobineshPurushothaman, PACKT Publishing,2nd Edition, 2015

3.8 UNIT END EXERCISES

1. Explain the building of RESTful web service application using NETBEANs IDE
2. Write a short note on WSDL
3. Explain the security of RESTful web services
4. Design guidelines for building RESTful web services

4

DEVELOPING SERVICE-ORIENTED APPLICATIONS WITH WCF

Unit Structure :

4.0 INTRODUCTION

Web services are open standard (XML, SOAP, HTTP, etc.) based web applications that interact with other web applications for the purpose of exchanging data. Web services can convert your existing applications into web applications.

What Is Windows Communication Foundation?

Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data. A few sample scenarios include:

- A secure service to process business transactions.
- A service that supplies current data to others, such as a traffic report or other monitoring service.
- A chat service that allows two people to communicate or exchange data in real time.
- A dashboard application that polls one or more services for data and presents it in a logical presentation.
- Exposing a workflow implemented using Windows Workflow Foundation as a WCF service.

4.1 FEATURES OF WCF

- **Service Orientation**

One consequence of using WS standards is that WCF enables you to create service oriented applications. Service-oriented architecture (SOA) is the reliance on Web services to send and receive data. The services have the general advantage of being loosely-coupled instead of hard-coded from one application to another. A loosely-coupled relationship implies that any client created on any platform can connect to any service as long as the essential contracts are met.

- **Interoperability**

WCF implements modern industry standards for Web service interoperability.

- **Multiple Message Patterns**

Messages are exchanged in one of several patterns. The most common pattern is the request/reply pattern, where one endpoint requests data from a second endpoint. The second endpoint replies. There are other patterns such as a one-way message in which a single endpoint sends a message without any expectation of a reply. A more complex pattern is the duplex exchange pattern where two endpoints establish a connection and send data back and forth, similar to an instant messaging program.

- **Service Metadata**

WCF supports publishing service metadata using formats specified in industry standards such as WSDL, XML Schema and WS-Policy. This metadata can be used to automatically generate and configure clients for accessing WCF services. Metadata can be published over HTTP and HTTPS or using the Web Service Metadata Exchange standard.

- **Data Contracts**

Because WCF is built using the .NET Framework, it also includes code-friendly methods of supplying the contracts you want to enforce. One of

the universal types of contracts is the data contract. In essence, as you code your service using Visual C# or Visual Basic, the easiest way to handle data is by creating classes that represent a data entity with properties that belong to the data entity. WCF includes a comprehensive system for working with data in this easy manner. Once you have created the classes that represent data, your service automatically generates the metadata that allows clients to comply with the data types you have designed

- **Security**

Messages can be encrypted to protect privacy and you can require users to authenticate themselves before being allowed to receive messages. Security can be implemented using well-known standards such as SSL or WS-SecureConversation.

- **Multiple Transports and Encodings**

Messages can be sent on any of several built-in transport protocols and encodings. The most common protocol and encoding is to send text encoded SOAP messages using the HyperText Transfer Protocol (HTTP) for use on the World Wide Web. Alternatively, WCF allows you to send messages over TCP, named pipes, or MSMQ. These messages can be encoded as text or using an optimized binary format. Binary data can be sent efficiently using the MTOM standard. If none of the provided transports or encodings suit your needs you can create your own custom transport or encoding.

- **Reliable and Queued Messages**

WCF supports reliable message exchange using reliable sessions implemented over WS-Reliable Messaging and using MSMQ.

- **Durable Messages**

A durable message is one that is never lost due to a disruption in the communication. The messages in a durable message pattern are always saved to a database. If a disruption occurs, the database allows you to resume the message exchange when the connection is restored. You can also create a durable message using the Windows Workflow Foundation (WF). For more information.

- **Transactions**

WCF also supports transactions using one of three transaction models: WS-Atomic Transactions, the APIs in the System. Transactions name space, and Microsoft Distributed Transaction Coordinator

- **AJAX and REST Support**

REST is an example of an evolving Web 2.0 technology. WCF can be configured to process "plain" XML data that is not wrapped in a SOAP envelope. WCF can also be extended to support specific XML formats,

such as ATOM (a popular RSS standard), and even non-XML formats, such as JavaScript Object Notation (JSON).

- **Extensibility**

The WCF architecture has a number of extensibility points. If extra capability is required, there are a number of entry points that allow you to customize the behavior of a service.

4.2 WCF INTEGRATION WITH OTHER MICROSOFT TECHNOLOGIES

WCF is a flexible platform. Because of this extreme flexibility, WCF is also used in several other Microsoft products. By understanding the basics of WCF, you have an immediate advantage if you also use any of these products.

The first technology to pair with WCF was the Windows Workflow Foundation (WF). Workflows simplify application development by encapsulating steps in the workflow as "activities." In the first version of Windows Workflow Foundation, a developer had to create a host for the workflow. The next version of Windows Workflow Foundation was integrated with WCF. That allowed any workflow to be easily hosted in a WCF service. You can do this by automatically choosing the WF/WCF project type in Visual Studio 2012 or later.

Microsoft BizTalk Server R2 also utilizes WCF as a communication technology. BizTalk is designed to receive and transform data from one standardized format to another. Messages must be delivered to its central message box where the message can be transformed using either a strict mapping or by using one of the BizTalk features such as its workflow engine. BizTalk can now use the WCF Line of Business (LOB) adapter to deliver messages to the message box.

The hosting features of Windows Server AppFabric application server are specifically designed for deploying and managing applications that use WCF for communication. The hosting features include rich tooling and configuration options specifically designed for WCF-enabled applications.

4.3 SPECIAL BEHAVIOURAL CHARACTERISTICS

XML-Based

Web services use XML at data representation and data transportation layers. Using XML eliminates any networking, operating system, or platform binding. Web services based applications are highly interoperable at their core level.

Loosely Coupled

A consumer of a web service is not tied to that web service directly. The web service interface can change over time without compromising the

client's ability to interact with the service. A tightly coupled system implies that the client and server logic are closely tied to one another, implying that if one interface changes, the other must be updated. Adopting a loosely coupled architecture tends to make software systems more manageable and allows simpler integration between different systems.

Coarse-Grained

Object-oriented technologies such as Java expose their services through individual methods. An individual method is too fine an operation to provide any useful capability at a corporate level. Building a Java program from scratch requires the creation of several fine-grained methods that are then composed into a coarse-grained service that is consumed by either a client or another service.

Businesses and the interfaces that they expose should be coarse-grained. Web services technology provides a natural way of defining coarse-grained services that access the right amount of business logic.

Ability to be Synchronous or Asynchronous

Synchronicity refers to the binding of the client to the execution of the service. In synchronous invocations, the client blocks and waits for the service to complete its operation before continuing. Asynchronous operations allow a client to invoke a service and then execute other functions.

Asynchronous clients retrieve their result at a later point in time, while synchronous clients receive their result when the service has completed. Asynchronous capability is a key factor in enabling loosely coupled systems.

Supports Remote Procedure Calls(RPCs)

Web services allow clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol. Remote procedures expose input and output parameters that a web service must support.

Component development through Enterprise JavaBeans (EJBs) and .NET Components has increasingly become a part of architectures and enterprise deployments over the past couple of years. Both technologies are distributed and accessible through a variety of RPC mechanisms.

A web service supports RPC by providing services of its own, equivalent to those of a traditional component, or by translating incoming invocations into an invocation of an EJB or a .NET component.

Supports Document Exchange

One of the key advantages of XML is its generic way of representing not only data, but also complex documents. These documents can be as simple as representing a current address, or they can be as complex as

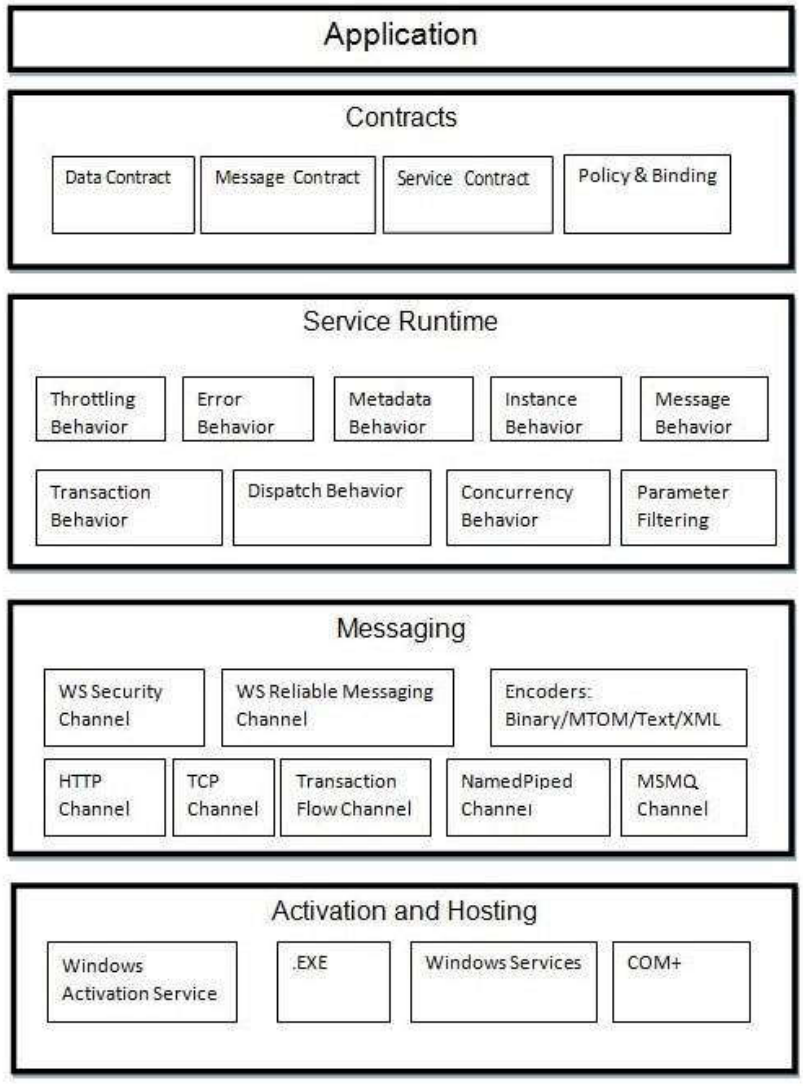
representing an entire book or Request for Quotation (RFQ). Web services support the transparent exchange of documents to facilitate business integration.

4.4 WEB SERVICES – ARCHITECTURE

There are two ways to view the web service architecture −

- The first is to examine the individual roles of each web service actor.
- The second is to examine the emerging web service protocol stack.

WCF has a layered architecture that offers ample support for developing various distributed applications. The architecture is explained below in detail.



Contracts

The contracts layer is just next to the application layer and contains information similar to that of a real-world contract that specifies the

operation of a service and the kind of accessible information it will make. Contracts are basically of four types discussed below in brief −

- **Service contract** − This contract provides information to the client as well as to the outer world about the offerings of the endpoint, and the protocols to be used in the communication process.
- **Data contract** − The data exchanged by a service is defined by a data contract. Both the client and the service has to be in agreement with the data contract.
- **Message contract** − A data contract is controlled by a message contract. It primarily does the customization of the type formatting of the SOAP message parameters. Here, it should be mentioned that WCF employs SOAP format for the purpose of communication. SOAP stands for Simple Object Access Protocol.
- **Policy and Binding** − There are certain pre-conditions for communication with a service, and such conditions are defined by policy and binding contract. A client needs to follow this contract.

Service Runtime

The service runtime layer is just below the contracts layer. It specifies the various service behaviors that occur during runtime. There are many types of behaviors that can undergo configuration and come under the service runtime.

- **Throttling Behavior** − Manages the number of messages processed.
- **Error Behavior** − Defines the result of any internal service error occurrence.
- **Metadata Behavior** − Specifies the availability of metadata to the outside world.
- **Instance Behavior** − Defines the number of instances that needs to be created to make them available for the client.
- **Transaction Behavior** − Enables a change in transaction state in case of any failure.
- **Dispatch Behavior** − Controls the way by which a message gets processed by the infrastructure of WCF.
- **Concurrency Behavior** − Controls the functions that run parallel during a client-server communication.
- **Parameter Filtering** − Features the process of validation of parameters to a method before it gets invoked.

Messaging

This layer, composed of several channels, mainly deals with the message content to be communicated between two endpoints. A set of channels

form a channel stack and the two major types of channels that comprise the channel stack are the following ones −

- **Transport Channels** − These channels are present at the bottom of a stack and are accountable for sending and receiving messages using transport protocols like HTTP, TCP, Peer-to-Peer, Named Pipes, and MSMQ.

- **Protocol Channels** − Present at the top of a stack, these channels also known as layered channels, implement wire-level protocols by modifying messages.

Activation and Hosting

The last layer of WCF architecture is the place where services are actually hosted or can be executed for easy access by the client. This is done by various mechanisms discussed below in brief.

- **IIS** − IIS stands for Internet Information Service. It offers a myriad of advantages using the HTTP protocol by a service. Here, it is not required to have the host code for activating the service code; instead, the service code gets activated automatically.

- **Windows Activation Service** − This is popularly known as WAS and comes with IIS 7.0. Both HTTP and non-HTTP based communication is possible here by using TCP or Namedpipe protocols.

- **Self-hosting** − This is a mechanism by which a WCF service gets self-hosted as a console application. This mechanism offers amazing flexibility in terms of choosing the desired protocols and setting own addressing scheme.

- **Windows Service** − Hosting a WCF service with this mechanism is advantageous, as the services then remain activated and accessible to the client due to no runtime activation.

4.5 WEB SERVICE ROLES

There are three major roles within the web service architecture −

Service Provider

This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

Service Requestor

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Service Registry

This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

4.6 WEB SERVICE PROTOCOL STACK

A second option for viewing the web service architecture is to examine the emerging web service protocol stack. The stack is still evolving, but currently has four main layers.

Service Transport

This layer is responsible for transporting messages between applications. Currently, this layer includes Hyper Text Transport Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and newer protocols such as Blocks Extensible Exchange Protocol (BEEP).

XML Messaging

This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.

Service Description

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

Service Discovery

This layer is responsible for centralizing services into a common registry and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

As web services evolve, additional layers may be added and additional technologies may be added to each layer.

Few Words about Service Transport

The bottom of the web service protocol stack is service transport. This layer is responsible for actually transporting XML messages between two computers.

Hyper Text Transfer Protocol (HTTP)

Currently, HTTP is the most popular option for service transport. HTTP is simple, stable, and widely deployed. Furthermore, most firewalls allow HTTP traffic. This allows XMLRPC or SOAP messages to masquerade as HTTP messages. This is good if you want to integrate remote applications,

but it does raise a number of security concerns.ise a number of security concerns.

Blocks Extensible Exchange Protocol (BEEP)

This is a promising alternative to HTTP. BEEP is a new Internet Engineering Task Force (IETF) framework for building new protocols. BEEP is layered directly on TCP and includes a number of built-in features, including an initial handshake protocol, authentication, security, and error handling. Using BEEP, one can create new protocols for a variety of applications, including instant messaging, file transfer, content syndication, and network management.

SOAP is not tied to any specific transport protocol. In fact, you can use SOAP via HTTP, SMTP, or FTP. One promising idea is therefore to use SOAP over BEEP.

Web Services Components:

Over the past few years, three primary technologies have emerged as worldwide standards that make up the core of today's web services technology. These technologies are discussed below.

XML-RPC

This is the simplest XML-based protocol for exchanging information between computers.

- XML-RPC is a simple protocol that uses XML messages to perform RPCs.
- Requests are encoded in XML and sent via HTTP POST.
- XML responses are embedded in the body of the HTTP response.
- XML-RPC is platform-independent.
- XML-RPC allows diverse applications to communicate.
- A Java client can speak XML-RPC to a Perl server.
- XML-RPC is the easiest way to get started with web services.

4.7 SOAP

SOAP is an XML-based protocol for exchanging information between computers.

- SOAP is a communication protocol.
- SOAP is for communication between applications.
- SOAP is a format for sending messages.

- SOAP is designed to communicate via Internet.
- SOAP is platform independent.
- SOAP is language independent.
- SOAP is simple and extensible.
- SOAP allows you to get around firewalls.
- SOAP will be developed as a W3C standard.

4.8 WSDL

WSDL is an XML-based language for describing web services and how to access them.

- WSDL stands for Web Services Description Language.
- WSDL was developed jointly by Microsoft and IBM.
- WSDL is an XML based protocol for information exchange in decentralized and distributed environments.
- WSDL is the standard format for describing a web service.
- WSDL definition describes how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of UDDI, an XML-based worldwide business registry.
- WSDL is the language that UDDI uses.
- WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'.

4.9 UDDI

UDDI is an XML-based standard for describing, publishing, and finding web services.

- UDDI stands for Universal Description, Discovery, and Integration.
- UDDI is a specification for a distributed registry of web services.
- UDDI is platform independent, open framework.
- UDDI can communicate via SOAP, CORBA, and Java RMI Protocol.
- UDDI uses WSDL to describe interfaces to web services.

- UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.
- UDDI is an open industry initiative enabling businesses to discover each other and define how they interact over the Internet.

4.10 WEB SERVICES – EXAMPLES

Based on the web service architecture, we create the following two components as a part of web services implementation −

Service Provider or Publisher

This is the provider of the web service. The service provider implements the service and makes it available on the Internet or intranet.

We will write and publish a simple web service using .NET SDK.

Service Requestor or Consumer

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

We will also write two web service requestors: one web-based consumer (ASP.NET application) and another Windows application-based consumer.

Given below is our first web service example which works as a service provider and exposes two methods (add and SayHello) as the web services to be used by applications. This is a standard template for a web service. .NET web services use the .asmx extension. Note that a method exposed as a web service has the WebMethod attribute. Save this file as FirstService.asmx in the IIS virtual directory (as explained in configuring IIS; for example, c:\MyWebSerces).

FirstService.asmx

```
<%@WebService language ="C#"class="FirstService"%>


usingSystem;
usingSystem.Web.Services;
usingSystem.Xml.Serialization;


[WebService(Namespace="http://localhost/MyWebServices/")]
publicclassFirstService:WebService{
```

```
[WebMethod]

publicintAdd(int a,int b){

return a + b;

}


[WebMethod]

publicStringSayHello(){

return"Hello World";

}

}
```

To test a web service, it must be published. A web service can be published either on an intranet or the Internet. We will publish this web service on IIS running on a local machine. Let us start with configuring the IIS.

- Open Start → Settings → Control Panel → Administrative tools → Internet Services Manager.
- Expand and right-click on the default web site; select New &#rarr; Virtual Directory. The Virtual Directory Creation Wizard opens. Click Next.
- The "Virtual Directory Alias" screen opens. Type the virtual directory name. For example, MyWebServices. Click Next.
- The "Web Site Content Directory" screen opens.
- Enter the directory path name for the virtual directory. For example, c:\MyWebServices. Click Next.
- The "Access Permission" screen opens. Change the settings as per your requirements. Let us keep the default settings for this exercise.
- Click the Next button. It completes the IIS configuration.
- Click Finish to complete the configuration.

To test whether the IIS has been configured properly, copy an HTML file (For example, x.html) in the virtual directory (C:\MyWebServices) created above. Now, open Internet Explorer and type **http://localhost/MyWebServices/x.html**. It should open the x.html file.

Note − If it does not work, try replacing the localhost with the IP address of your machine. If it still does not work, check whether IIS is running; you may need to reconfigure the IIS and the Virtual Directory.

To test this web service, copy FirstService.asmx in the IIS virtual directory created above (C:\MyWebServices). Open the web service in Internet Explorer (http://localhost/MyWebServices/FirstService.asmx). It should open your web service page. The page should have links to two methods exposed as web services by our application. Congratulations! You have written your first web service!

4.11 TESTING THE WEB SERVICE

As we have just seen, writing web services is easy in the .NET Framework. Writing web service consumers is also easy in the .NET framework; however, it is a bit more involved. As said earlier, we will write two types of service consumers, one web-based and another Windows application-based consumer. Let us write our first web service consumer.

Web-Based Service Consumer

Write a web-based consumer as given below. Call it WebApp.aspx. Note that it is an ASP.NET application. Save this in the virtual directory of the web service (c:\MyWebServices\WebApp.axpx).

This application has two text fields that are used to get numbers from the user to be added. It has one button, Execute, that when clicked gets the Add and SayHello web services.

```
WebApp.aspx

<%@PageLanguage="C#"%>

<script runat="server">

voidrunSrvice_Click(Object sender,EventArgs e){

FirstServicemySvc=newFirstService();

Label1.Text=mySvc.SayHello();

Label2.Text=mySvc.Add(Int32.Parse(txtNum1.Text),Int32.Parse(txtNum
2.Text)).ToString();

}

</script>


<html>

<head></head>
```

```
<body>
<form runat="server">
<p>
<em>FirstNumber to Add</em>:
<asp:TextBox id
="txtNum1"runat="server"Width="43px">4</asp:TextBox>
</p>

<p>
<em>SecondNumberToAdd</em>:
<asp:TextBox id
="txtNum2"runat="server"Width="44px">5</asp:TextBox>
</p>

<p>
<strong><u>WebServiceResult-</u></strong>
</p>

<p>
<em>Hello world Service</em>:
<asp:Label id ="Label1"runat="server"Font-
Underline="True">Label</asp:Label>
</p>

<p>
<em>AddService</em>:
&<asp:Label id ="Label2"runat="server"Font-
Underline="True">Label</asp:Label>
</p>
```

```
<p align ="left">

<asp:Button id ="runSrvice" onclick ="runSrvice_Click"runat="server"Text="Execute"></asp:Button>

</p>

</form>

</body>

</html>
```

After the consumer is created, we need to create a proxy for the web service to be consumed. This work is done automatically by Visual Studio .NET for us when referencing a web service that has been added. Here are the steps to be followed −

- Create a proxy for the Web Service to be consumed. The proxy is created using the WSDL utility supplied with the .NET SDK. This utility extracts information from the Web Service and creates a proxy. The proxy is valid only for a particular Web Service. If you need to consume other Web Services, you need to create a proxy for this service as well. Visual Studio .NET creates a proxy automatically for you when the Web Service reference is added. Create a proxy for the Web Service using the WSDL utility supplied with the .NET SDK. It will create FirstSevice.cs file in the current directory. We need to compile it to create FirstService.dll (proxy) for the Web Service.

c:> WSDL http://localhost/MyWebServices/FirstService.asmx?WSDL

c:> csc /t:libraryFirstService.cs

- Put the compiled proxy in the bin directory of the virtual directory of the Web Service (c:\MyWebServices\bin). Internet Information Services (IIS) looks for the proxy in this directory.

- Create the service consumer, in the same way we already did. Note that an object of the Web Service proxy is instantiated in the consumer. This proxy takes care of interacting with the service.

- Type the URL of the consumer in IE to test it (for example, http://localhost/MyWebServices/WebApp.aspx).

4.12 WINDOWS APPLICATION-BASED WEB SERVICE CONSUMER

Writing a Windows application-based web service consumer is the same as writing any other Windows application. You only need to create the proxy (which we have already done) and reference this proxy when compiling the application. Following is our Windows application that uses the web service. This application creates a web service object (of course, proxy) and calls the SayHello, and Add methods on it.

```
WinApp.cs


usingSystem;

usingSystem.IO;


namespaceSvcConsumer{

classSvcEater{

publicstaticvoidMain(String[]args){

FirstServicemySvc=newFirstService();

Console.WriteLine("Calling Hello World Service: "+mySvc.SayHello());

Console.WriteLine("Calling Add(2, 3) Service: "+mySvc.Add(2,3).ToString());

}

}

}
```

Compile it using c:\>csc /r:FirstService.dll WinApp.cs. It will create WinApp.exe. Run it to test the application and the web service.

Now, the question arises: How can you be sure that this application is actually calling the web service?

It is simple to test. Stop your web server so that the web service cannot be contacted. Now, run the WinApp application. It will fire a runtime exception. Now, start the web server again. It should work.

4.13 SUMMARY

Web services are open standard (XML, SOAP, HTTP, etc.) based web applications that interact with other web applications for the purpose of exchanging data. Web services can convert your existing applications into web application**s.**

4.14 QUESTIONS

1. Explain WCF.
2. List out Features of WCF.
3. Explain Web services Architecture
4. Explain web service Roles.

5. Give short notes on:

   a) SOAP

   b) WSDL

   c) UDDI

4.15 REFERENCE

- https://learn.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf
- https://www.tutorialspoint.com/mfc/mfc_windows_fundamentals.htm
- https://www.tutorialspoint.com/wcf/wcf_architecture.htm

❖❖❖❖

5

BASIC WCF PROGRAMMING

Unit Structure :

5.0 INTRODUCTION

NET Framework Client Profile is a lightweight version of the full .NET Framework designed for clients that don't need the entire framework. Not all of Windows Communication Foundation is supported by the client framework.

5.1 WCF FEATURES SUPPORTED BY THE .NET FRAMEWORK CLIENT PROFILE

The following Windows Communication Foundation features are supported by .NET Framework Client Profile:

- All of WCF is supported except for Cardspace and web hosting.
- Remoting TCP/IP channels are supported.
- Asmx (Web Services) are not supported.

5.2 BASIC WCF PROGRAMMING

Windows Communication Foundation (WCF) enables applications to communicate whether they are on the same computer, across the Internet, or on different application platforms.

The Basic Tasks

The basic tasks to perform are, in order:

1. Define the service contract. A service contract specifies the signature of a service, the data it exchanges, and other contractually required data. For more information..
2. Implement the contract. To implement a service contract, create a class that implements the contract and specify custom behaviors that the runtime should have. For more information.
3. Configure the service by specifying endpoints and other behavior information. For more information.
4. Host the service.
5. Build a client application. For more information.

5.3 DESIGNING AND IMPLEMENTING SERVICES

This section shows you how to define and implement WCF contracts. A service contract specifies what an endpoint communicates to the outside world. At a more concrete level, it is a statement about a set of specific messages organized into basic message exchange patterns (MEPs), such as request/reply, one-way, and duplex. If a service contract is a logically related set of message exchanges, a service operation is a single message exchange. For example, a Hello operation must obviously accept one message (so the caller can announce the greeting) and may or may not return a message (depending upon the courtesy of the operation).

Overview

This topic provides a high level conceptual orientation to designing and implementing WCF services. Subtopics provide more detailed information about the specifics of design and implementation. Before designing and implementing your WCF application, it is recommended that you:

- Understand what a service contract is, how it works, and how to create one.
- Understand that contracts state minimum requirements that runtime configuration or the hosting environment may not support.

5.4 SERVICE CONTRACTS

A service contract specifies the following:

- The operations a contract exposes.
- The signature of the operations in terms of messages exchanged.
- The data types of these messages.
- The location of the operations.
- The specific protocols and serialization formats that are used to support successful communication with the service.

For example, a purchase order contract might have a Create Order operation that accepts an input of order information types and returns success or failure information, including an order identifier. It might also have a GetOrderStatus operation that accepts an order identifier and returns order status information. A service contract of this sort would specify:

1. That the purchase order contract consisted of Create Order and Get Order Status operations.
2. That the operations have specified input messages and output messages.
3. The data that these messages can carry.
4. Categorical statements about the communication infrastructure necessary to successfully process the messages. For example, these details include whether and what forms of security are required to establish successful communication.

To convey this kind of information to other applications on many platforms (including non-Microsoft platforms), XML service contracts are publicly expressed in standard XML formats, such as Web Services Description Language (WSDL) and XML Schema (XSD), among others. Developers for many platforms can use this public contract information to create applications that can communicate with the service, both because they understand the language of the specification and because those languages are designed to enable interoperation by describing the public forms, formats, and protocols that the service supports.

Contracts can be expressed many ways, and while WSDL and XSD are excellent languages to describe services in an accessible way, they are difficult languages to use directly and are merely descriptions of a service, not service contract implementations. Therefore, WCF applications use managed attributes, interfaces, and classes both to define the structure of a service and to implement it.

The resulting contract defined in managed types can be exported as metadata—WSDL and XSD—when needed by clients or other service implementers. The result is a straightforward programming model that can be described (using public metadata) to any client application. The details of the underlying SOAP messages, the transportation and security-related information, and so on, can be left to WCF, which performs the necessary conversions to and from the service contract type system to the XML type system automatically.

5.5 MESSAGES UP FRONT AND CENTER

Using managed interfaces, classes, and methods to model service operations is straightforward when you are used to remote procedure call (RPC)-style method signatures, in which passing parameters into a method and receiving return values is the normal form of requesting functionality from an object or other type of code. For example, programmers using managed languages such as Visual Basic and C++ COM can apply their knowledge of the RPC-style approach (whether using objects or interfaces) to the creation of WCF service contracts without experiencing the problems inherent in RPC-style distributed object systems. Service orientation provides the benefits of loosely coupled, message-oriented programming while retaining the ease and familiarity of the RPC programming experience.

Many programmers are more comfortable with message-oriented application programming interfaces, such as message queues like Microsoft MSMQ, the System.Messaging namespaces in the .NET Framework, or sending unstructured XML in HTTP requests, to name a few.

Understanding the Hierarchy of Requirements

A service contract groups the operations; specifies the message exchange pattern, message types, and data types those messages carry; and indicates categories of run-time behavior an implementation must have to support the contract (for example, it may require that messages be encrypted and signed). The service contract itself does not specify precisely how these requirements are met, only that they must be. The type of encryption or the manner in which a message is signed is up to the implementation and configuration of a compliant service.

Notice the way that the contract requires certain things of the service contract implementation and the run-time configuration to add behavior. The set of requirements that must be met to expose a service for use builds on the preceding set of requirements. If a contract makes requirements of the implementation, an implementation can require yet more of the configuration and bindings that enable the service to run. Finally, the host application must also support any requirements that the service configuration and bindings add.

This additive requirement process is important to keep in mind while designing, implementing, configuring, and hosting a Windows Communication Foundation (WCF) service application. For example, the contract can specify that it needs to support a session. If so, then you must configure the binding to support that contractual requirement, or the service implementation will not work. Or if your service requires Windows Integrated Authentication and is hosted in Internet Information Services (IIS), the Web application in which the service resides must have Windows Integrated Authentication turned on and anonymous support turned off.

5.6 DESIGNING SERVICE CONTRACTS

This topic describes what service contracts are, how they are defined, what operations are available (and the implications for the underlying message exchanges), what data types are used, and other issues that help you design operations that satisfy the requirements of your scenario.

Creating a Service Contract

Services expose a number of operations. In Windows Communication Foundation (WCF) applications, define the operations by creating a method and marking it with the OperationContractAttribute attribute. Then, to create a service contract, group together your operations, either by declaring them within an interface marked with the Service Contract Attribute attribute, or by defining them in a class marked with the same attribute.

Any methods that do not have a OperationContractAttribute attribute are not service operations and are not exposed by WCF services.

5.7 CLASSES OR INTERFACES

Both classes and interfaces represent a grouping of functionality and, therefore, both can be used to define a WCF service contract. However, it is recommended that you use interfaces because they directly model service contracts. Without an implementation, interfaces do no more than define a grouping of methods with certain signatures. Implement a service contract interface and you have implemented a WCF service.

All the benefits of managed interfaces apply to service contract interfaces:

- Service contract interfaces can extend any number of other service contract interfaces.
- A single class can implement any number of service contracts by implementing those service contract interfaces.
- You can modify the implementation of a service contract by changing the interface implementation, while the service contract remains the same.

- You can version your service by implementing the old interface and the new one. Old clients connect to the original version, while newer clients can connect to the newer version.

5.8 PARAMETERS AND RETURN VALUES

Each operation has a return value and a parameter, even if these are void. However, unlike a local method, in which you can pass references to objects from one object to another, service operations do not pass references to objects. Instead, they pass copies of the objects.

This is significant because each type used in a parameter or return value must be serializable; that is, it must be possible to convert an object of that type into a stream of bytes and from a stream of bytes into an object.

Primitive types are serializable by default, as are many types in the .NET Framework.

Data Contracts

Service-oriented applications like Windows Communication Foundation (WCF) applications are designed to interoperate with the widest possible number of client applications on both Microsoft and non-Microsoft platforms. For the widest possible interoperability, it is recommended that you mark your types with the Data Contract Attribute and Data Member Attribute attributes to create a data contract, which is the portion of the service contract that describes the data that your service operations exchange.

Data contracts are opt-in style contracts: No type or data member is serialized unless you explicitly apply the data contract attribute. Data contracts are unrelated to the access scope of the managed code: Private data members can be serialized and sent elsewhere to be accessed publicly. (For a basic example of a data contract, WCF handles the definition of the underlying SOAP messages that enable the operation's functionality as well as the serialization of your data types into and out of the body of the messages. As long as your data types are serializable, you do not need to think about the underlying message exchange infrastructure when designing your operations.

Although the typical WCF application uses the Data Contract Attribute and Data Member Attribute attributes to create data contracts for operations, you can use other serialization mechanisms. The standard I Serializable, Serializable Attribute, and IX ml Serializable mechanisms all work to handle the serialization of your data types into the underlying SOAP messages that carry them from one application to another. You can employ more serialization strategies if your data types require special support.

Mapping Parameters and Return Values to Message Exchanges

Service operations are supported by an underlying exchange of SOAP messages that transfer application data back and forth, in addition to the data required by the application to support certain standard security, transaction, and session-related features. Because this is the case, the signature of a service operation dictates a certain underlying message exchange pattern (MEP) that can support the data transfer and the features an operation requires.

Request/ Reply

A request/reply pattern is one in which a request sender (a client application) receives a reply with which the request is correlated. This is the default MEP because it supports an operation in which one or more parameters are passed to the operation and a return value is passed back to the caller. For example, the following C# code example shows a basic service operation that takes one string and returns a string.

C#Copy

```
[OperationContractAttribute]

stringHello(string greeting);
```

The following is the equivalent Visual Basic code.

VBCopy

```
<OperationContractAttribute()>

Function Hello (ByVal greeting AsString) AsString
```

This operation signature dictates the form of underlying message exchange. If no correlation existed, WCF cannot determine for which operation the return value is intended.

Note that unless you specify a different underlying message pattern, even service operations that return void (Nothing in Visual Basic) are request/reply message exchanges. The result for your operation is that unless a client invokes the operation asynchronously, the client stops processing until the return message is received, even though that message is empty in the normal case. The following C# code example shows an operation that does not return until the client has received an empty message in response.

C#Copy

```
[OperationContractAttribute]

voidHello(string greeting);
```

The following is the equivalent Visual Basic code.

VBCopy

<OperationContractAttribute()>

Sub Hello (ByVal greeting AsString)

The preceding example can slow client performance and responsiveness if the operation takes a long time to perform, but there are advantages to request/reply operations even when they return void. The most obvious one is that SOAP faults can be returned in the response message, which indicates that some service-related error condition has occurred, whether in communication or processing. SOAP faults that are specified in a service contract are passed to the client application as a Fault Exception<TDetail> object, where the type parameter is the type specified in the service contract. This makes notifying clients about error conditions in WCF services easy.

One-way

If the client of a WCF service application should not wait for the operation to complete and does not process SOAP faults, the operation can specify a one-way message pattern. A one-way operation is one in which a client invokes an operation and continues processing after WCF writes the message to the network. Typically this means that unless the data being sent in the outbound message is extremely large the client continues running almost immediately (unless there is an error sending the data). This type of message exchange pattern supports event-like behavior from a client to a service application.

A message exchange in which one message is sent and none are received cannot support a service operation that specifies a return value other than void; in this case an InvalidOperationException exception is thrown.

No return message also means that there can be no SOAP fault returned to indicate any errors in processing or communication. (Communicating error information when operations are one-way operations requires a duplex message exchange pattern.)

To specify a one-way message exchange for an operation that returns void, set the IsOneWay property to true, as in the following C# code example.

C#Copy

[OperationContractAttribute(IsOneWay=true)]

voidHello(string greeting);

The following is the equivalent Visual Basic code.

VBCopy

<OperationContractAttribute(IsOneWay := True)>

Sub Hello (ByVal greeting AsString)

This method is identical to the preceding request/reply example, but setting the IsOneWay property to true means that although the method is identical, the service operation does not send a return message and clients return immediately once the outbound message has been handed to the channel layer.

Duplex

A duplex pattern is characterized by the ability of both the service and the client to send messages to each other independently whether using one-way or request/reply messaging. This form of two-way communication is useful for services that must communicate directly to the client or for providing an asynchronous experience to either side of a message exchange, including event-like behavior.

The duplex pattern is slightly more complex than the request/reply or one-way patterns because of the additional mechanism for communicating with the client.

To design a duplex contract, you must also design a callback contract and assign the type of that callback contract to the CallbackContract property of the ServiceContractAttribute attribute that marks your service contract.

To implement a duplex pattern, you must create a second interface that contains the method declarations that are called on the client.

Caution

When a service receives a duplex message, it looks at the ReplyTo element in that incoming message to determine where to send the reply. If the channel that is used to receive the message is not secured, then an untrusted client could send a malicious message with a target machine's ReplyTo, leading to a denial of service (DOS) of that target machine.

Out and Ref Parameters

In most cases, you can use in parameters (ByVal in Visual Basic) and out and ref parameters (ByRef in Visual Basic). Because both out and ref parameters indicate that data is returned from an operation, an operation signature such as the following specifies that a request/ reply operation is required even though the operation signature returns void.

C#Copy

```
[ServiceContractAttribute]

publicinterfaceIMyContract

{

 [OperationContractAttribute]
```

```
publicvoidPopulateData(refCustomDataType data);
}
```

The following is the equivalent Visual Basic code.

VBCopy

```
<ServiceContractAttribute()> _
PublicInterfaceIMyContract
<OperationContractAttribute()> _
PublicSubPopulateData(ByRef data AsCustomDataType)
EndInterface
```

The only exceptions are those cases in which your signature has a particular structure. For example, you can use the Net Msmq Binding binding to communicate with clients only if the method used to declare an operation returns void; there can be no output value, whether it is a return value, ref, or out parameter.

In addition, using out or ref parameters requires that the operation have an underlying response message to carry back the modified object. If your operation is a one-way operation, an InvalidOperationException exception is thrown at run time.

5.9 SPECIFY MESSAGE PROTECTION LEVEL ON THE CONTRACT

When designing your contract, you must also decide the message protection level of services that implement your contract. This is necessary only if message security is applied to the binding in the contract's endpoint. If the binding has security turned off (that is, if the system-provided binding sets the System.ServiceModel.SecurityMode to the value SecurityMode.None) then you do not have to decide on the message protection level for the contract. In most cases, system-provided bindings with message-level security applied provide a sufficient protection level and you do not have to consider the protection level for each operation or for each message.

The protection level is a value that specifies whether the messages (or message parts) that support a service are signed, signed and encrypted, or sent without signatures or encryption. The protection level can be set at various scopes: At the service level, for a particular operation, for a message within that operation, or a message part. Values set at one scope become the default value for smaller scopes unless explicitly overridden. If a binding configuration cannot provide the required minimum protection level for the contract, an exception is thrown. And when no protection level values are explicitly set on the contract, the binding

configuration controls the protection level for all messages if the binding has message security. This is the default behavior.

Important

Deciding whether to explicitly set various scopes of a contract to less than the full protection level of **ProtectionLevel.EncryptAndSign** is generally a decision that trades some degree of security for increased performance. In these cases, your decisions must revolve around your operations and the value of the data they exchange.

C#Copy

```
[ServiceContract]
publicinterfaceISampleService
{
  [OperationContractAttribute]
publicstringGetString();

  [OperationContractAttribute]
publicintGetInt();
}
```

The following is the equivalent Visual Basic code.

VBCopy

```
<ServiceContractAttribute()> _
PublicInterfaceISampleService
<OperationContractAttribute()> _
PublicFunctionGetString()AsString

<OperationContractAttribute()> _
PublicFunctionGetData() AsInteger
EndInterface
```

When interacting with an ISampleService implementation in an endpoint with a default WSHttp Binding (the default System. Service Model. Security Mode, which is Message), all messages are encrypted and signed because this is the default protection level. However, when an I Sample Service service is used with a default Basic Http Binding (the

default Security Mode, which is None), all messages are sent as text because there is no security for this binding and so the protection level is ignored (that is, the messages are neither encrypted nor signed). If the SecurityMode was changed to Message, then these messages would be encrypted and signed (because that would now be the binding's default protection level).

If you want to explicitly specify or adjust the protection requirements for your contract, set the Protection Level property (or any of the ProtectionLevel properties at a smaller scope) to the level your service contract requires. In this case, using an explicit setting requires the binding to support that setting at a minimum for the scope used. For example, the following code example specifies one Protection Level value explicitly, for the Get Guid operation.

C#Copy

```
[ServiceContract]
publicinterfaceIExplicitProtectionLevelSampleService
{
  [OperationContractAttribute]
publicstringGetString();

  [OperationContractAttribute(ProtectionLevel=ProtectionLevel.None)]
publicintGetInt();

[OperationContractAttribute(ProtectionLevel=ProtectionLevel.EncryptAn
dSign)]
publicintGetGuid();
}
```

The following is the equivalent Visual Basic code.

VBCopy

```
<ServiceContract()> _
PublicInterfaceIExplicitProtectionLevelSampleService
<OperationContract()> _
PublicFunctionGetString() AsString
EndFunction
```

```
<OperationContract(ProtectionLevel := ProtectionLevel.None)> _
PublicFunctionGetInt() AsInteger
EndFunction
<OperationContractAttribute(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
PublicFunctionGetGuid() AsInteger
EndFunction
EndInterface
```

A service that implements this I Explicit Protection Level Sample Service contract and has an endpoint that uses the default WS Http Binding (the default System. Service Model. Security Mode, which is Message) has the following behavior:

- The GetString operation messages are encrypted and signed.
- The GetInt operation messages are sent as unencrypted and unsigned (that is, plain) text.
- The GetGuid operation System.Guid is returned in a message that is encrypted and signed.

Other Operation Signature Requirements

Some application features require a particular kind of operation signature. For example, the NetMsmqBinding binding supports durable services and clients, in which an application can restart in the middle of communication and pick up where it left off without missing any messages.

Another example is the use of Stream types in operations. Because the Stream parameter includes the entire message body, if an input or an output (that is, ref parameter, out parameter, or return value) is of type Stream, then it must be the only input or output specified in your operation. In addition, the parameter or return type must be either Stream, System.ServiceModel.Channels.Message, or System.Xml.Serialization.IXmlSerializable.

Names, Namespaces, and Obfuscation

The names and namespaces of the .NET types in the definition of contracts and operations are significant when contracts are converted into WSDL and when contract messages are created and sent. Therefore, it is strongly recommended that service contract names and namespaces are explicitly set using the Name and Namespace properties of all supporting contract attributes such as the Service Contract Attribute, Operation Contract Attribute, Data Contract Attribute, Data Member Attribute, and other contract attributes.

One result of this is that if the names and namespaces are not explicitly set, the use of IL obfuscation on the assembly alters the contract type names and namespaces and results in modified WSDL and wire exchanges that typically fail. If you do not set the contract names and namespaces explicitly but do intend to use obfuscation, use the Obfuscation Attribute and Obfuscate Assembly Attribute attributes to prevent the modification of the contract type names and name spaces.

5.10 CONFIGURING WCF SERVICES

Once you have designed and implemented your service contract, you are ready to configure your service. This is where you define and customize how your service is exposed to clients, including specifying the address where it can be found, the transport and message encoding it uses to send and receive messages, and the type of security it requires.

Configuration as used here includes all the ways, imperatively in code or by using a configuration file, in which you can define and customize the various aspects of a service, such as specifying its endpoint addresses, the transports used, and its security schemes. In practice, writing configuration is a major part of programming WCF applications.

Simplified Configuration Starting with. NET Framework 4, WCF comes with a new default configuration model that simplifies WCF configuration requirements. If you do not provide any WCF configuration for a particular service, the runtime automatically configures your service with default endpoints, bindings, and behaviors.

Configuring Services Using Configuration Files A Windows Communication Foundation (WCF) service is configurable using the .NET Framework configuration technology. Most commonly, XML elements are added to the Web.config file for an Internet Information Services (IIS) site that hosts a WCF service. The elements allow you to change details, such as the endpoint addresses (the actual addresses used to communicate with the service) on a machine-by-machine basis.

Bindings In addition, WCF includes several system-provided common configurations in the form of bindings that allow you to quickly select the most basic features for how a client and service communicate, such as the transports, security, and message encodings used.

Endpoints All communication with a WCF service occurs through the endpoints of the service. Endpoints contain the contract, the configuration information that is specified in the bindings, and the addresses that indicate where to find the service or where to obtain information about the service.

Securing Services Using WCF and existing security mechanisms, you can implement confidentiality, integrity, authentication, and authorization into any service. You can also audit for security successes and failures.

5.11 SUMMARY

To understand the concept of WAS hosting, we need to comprehend how a system is configured and how a service contract is created, enabling different binding to the hosted service.

A WCF service boasts of a robust security system with two security modes or levels so that only an intended client can access the services. The security threats that are common in a distributed transaction are moderated to a large extent by WCF.

WCF has a layered architecture that offers ample support for developing various distributed applications.

5.12 QUESTIONS

1. Explain WCF features supported by the .NET Framework Client Profile
2. Explain basics of WCF programming
3. Explain service contracts
4. Explain Parameters and return values
5. How to configure WCF services.

5.13 REFERENCE

- https://learn.microsoft.com/en-us/dotnet/framework/wcf/wcf-and-net-framework-client-profile
- https://www.tutorialspoint.com/wcf/wcf_creating_service.htm
- https://www.tutorialspoint.com/wcf/wcf_versus_web_service.htm

6

WEB SERVICE QOS

Unit Structure :

6.0 INTRODUCTION

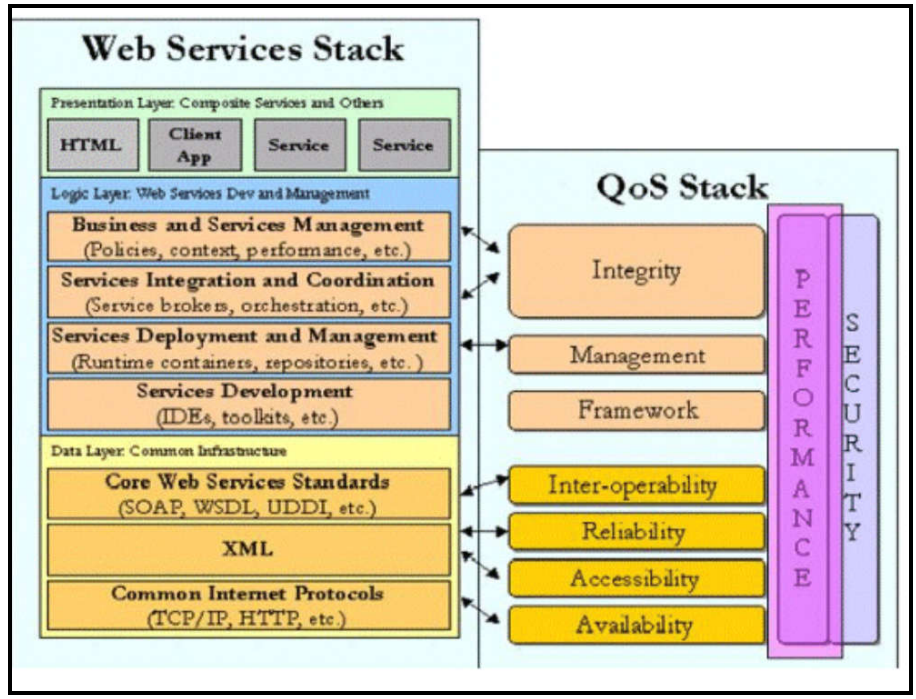
With the proliferation of Web services as a business solution to enterprise application integration, the quality of service (QoS) offered by Web services is becoming the utmost priority for service provider and their partners. Due to the dynamic and unpredictable nature of the Web, providing the acceptable QoS is really a challenging task. In addition to this, the different applications that are collaborating for Web service interaction with different requirements will compete for network resources. The preceding factors will force service providers to understand and achieve Web services QoS. Also, a better QoS for a Web service will bring a competitive advantage over others by being a unique selling point for a service provider.

The Web services QoS requirement mainly refers to the quality, both functional as well as non-functional, aspect of a Web service. This includes performance, reliability, integrity, accessibility, availability, interoperability, and security. In the first part of this article, we covered each of the above QoS aspects from a broad perspective. Now, in each coming article we will look into each of the QoS in detail and will try to come up with solutions and explore best practices to be followed. In this

article, we are trying to demystify the "performance" aspect of Web Services.

6.1 WEB SERVICES PERFORMANCE

The Web services-QoS stack we proposed is shown in the following fig



6.2 PROACTIVE WEB SERVICE QOS

Service providers can proactively provide high QoS to the service requestors by using ν Caching ν Load balancing ν both approaches can be done in web server level and web application level

6.3 CACHING WEB SERVICES

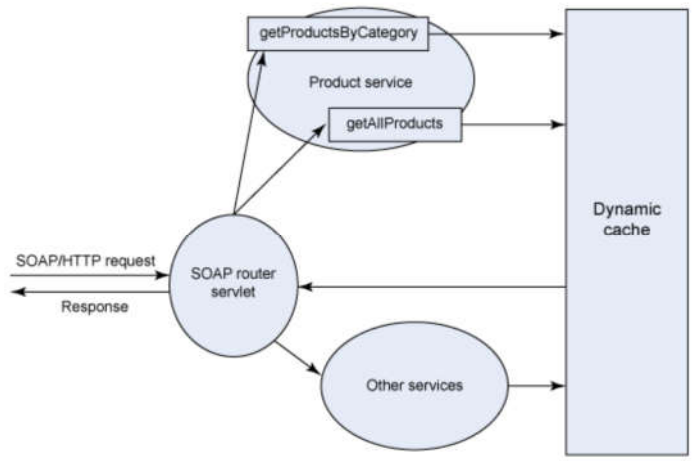


Figure 2. Principles for calling caching web services

Caching refers to storing the server response in the client itself ν A client not make a server request for the same resource again and again. ν Because a server response should have information about how caching is to be done ν So that a client caches the response for a time-period or never caches the server response ν If the data of web services does not change frequently, properly caching could boost the performance

Load Balancing

Prioritize various types of traffic and ensure that each request is treated appropriately to the business value ν A Web service provider can perform capacity modeling to create a top-down model of requesttraffic, current capacity utilization, and the resulting QoS ν Also categorize Web service traffic by the volume of traffic, traffic for different application service categories, and traffic from different sources.

Load Balancing ν Help in understanding the capacity that will be required to provide good QoS ν for a volume of service demand and for future planning E.g. capacity type of load balancing web application servers/Web servers the number of servers required for setting up a clustered server farm

Service providers can provide differentiated servicing by using the capacity model to….. ν determine the capacity needed for different customers and service types ν ensure appropriate QoS levels for different applications and customers ν For example; A multimedia Web Service might require good throughput, BUT a banking Web Service might require security and transactional QoS.

6.4 SERVICE QUALITY REQUIREMENTS

The major requirements for supporting QoS in Web services are as follows: ν Performance ν Availability ν Accessibility ν Integrity ν Reliability ν Interoperability ν Security 20 ν The other web service QoS characteristics ν Execution Time ν Cost/Price ν Transaction (ACID) ν Reputation ν Etc

QoS: Performance

Demystifying

Performance is measured by throughput and latency. Performance can also be determined by response time to guarantee maximum time required to complete a service request

Limitations

The overall performance of WS depends on application logic, network, and underlying messaging and transport protocols, such as SOAP and HTTP. The SOAP protocol uses a multi-step process to complete a communication cycle. This whole process is a timeconsuming one, which

requires various levels of XML parsing and XML validation and hence hits the performance of the Web Service. B

Best Practices

optimize the XML processing s

steps 1. use of efficient and lightweight parsers.

2. Efficient use of XML validation in production mode.

3. Use of compressed XML for sending the messages over network.

4. Web Service Caching 5. Use of simple data types in SOAP messages as far as possible.

QoS: Availability

Demystifying

Availability defines whether the Web Service is ready for immediate consumption. Associated with availability is Time-toRepair (TTR). TTR represents the time it takes to repair the Web Service.

Limitations

Building fault-tolerant systems for highly available Web Services is expensive. As companies roll out Web Services, the ability to manage this diverse, dynamic, distributed environment will become critical. Questions such as the following arise: Has one of my key servers become unavailable? Is a system being overly burdened? Why are requests taking so long?

Best Practices

Web Service Management Web Service Clustering

QoS: Accessiblity

Demystifying

Accessibility defines whether the Web Service is capable of serving the client's request. High accessibility of Web Services can be achieved by building highly scalable systems

Limitations

Building scalable systems are expensive, and this may cause smaller companies to defer this requirement. Also, this becomes an infrastructure issue for companies that deploy Web Services within their enterprise.

Best Practices S

service pooling

Load balancing (Scalability)

QoS: Integrity

Demystifying

Integrity is the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data.

Limitations

Data Integrity is important for any object's proper functioning and must be assured or it could corrupt a larger program and generate a very difficult to trace error. Web Services transactions tend to be asynchronous and long running in nature. Transaction integrity is just one of several QoS elements, including security and process orchestration, which are missing from the first incarnations of Web Services standards of SOAP, UDDI, and WSDL.

Best Practices

1. Emerging standards in the business process management and transactions will help to achieve the desired QoS.
2. Adopting standards such as BPEL4WS, WS-Coordination, WSTransaction, and BTP would benefit service providers.

QoS: Reliability

Demystifying

Reliability is the overall measure of a Web Service to maintain its service quality. The number of failures per day, week, month, or year represents an overall measure of reliability for a Web Service. Reliability also refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers.

Limitations

The Web Services currently rely on transport protocols such as HTTP, which are inherently stateless and follow a best-effort delivery mechanism. It does not guarantee whether the message will be delivered to the destination.

Best Practices

The above problem rising due to the use of unreliable protocols can be solved using the following techniques:

1. Use of asynchronous message queues.

2. Adoption of new reliable transport protocols (such as HTTPR, REST, and BEEP).

QoS: Interoperability

Demystifying

The fundamental goal of interoperability in Web Services is to cross the lines between the development environments used to implement services so that developers using those services don't have to think about which programming language or operating system the services are hosted on.

Limitations

Most of the Web Services specifications are defined under standards bodies. As these activities are under way, there seems to be a delay in the implementations. Vendors partly implement the specification in their products due to the competitive nature of this market. This results in poor interoperability.

Best Practices

1. Key to enabling seamless Web Services interoperability is the ability of one Web Services framework to consume the WSDL documents generated by other frameworks.
2. Web Services-Interoperability (WS-I) Profiles. The Basic Profile defines how a selected set of specified Web Services technologies, such as messaging and discovery, should be used together in an interoperable manner

QoS: Security

Demystifying

Security for Web Services refers to authentication mechanisms, messages encryption and access control, confidentiality, nonrepudiation and resilience to denial-of-service attacks.

Limitations

SOAP is a de-facto messaging standard for Web Services; inherently, it does not support many security features. Some of the Web Services-enabled applications also require role-based security features, which expose different functionalities, depending on user credentials.

Best Practices .

The following measures can be used while architecting the secure Web Services:

1. Use of XML Encryption.
2. Use of XML Key Management Specification. 3. Use of Private WANs, Web Service Network, and VPNs.
3. Use of Private WANs, Web Service Network, and VPNs.

4. P3P (Platform for Privacy Preferences) is an emerging standard for specifying privacy preferences for a user while using Web Services.

5. Use of security assertions.

6.5 WHY IS QOS IMPORTANT FOR WEB SERVICES?

Web services allow individual components of business logic to be published as building blocks to larger applications and services. Since each Web service is only a small segment of a larger application and not an entire large, monolithic application, there will be a significant number of competitive implementations for each Web service. Quality of service will be one of the most important differentiators between all of these competitive solutions.

Quality of quality of serviceservice encapsulates not only implementation details that affect metrics such as performance, but also deployment issues. For example, an application that is deployed on an application server can automatically scale and create additional ...

As you can see, the performance block (highlighted in purple) in the figure spans across almost all of the Web service layers on the left. It requires tuning at all layers to get a desired level of performance from Web service. Let us first briefly analyze this quality aspect, for demystifying and to see its limitations, and then work in depth on some of the optimal solutions.

6.6 WEB SERVICES PERFORMANCE-DEMYSTIFYING

The performance of a Web Service is measured in terms of throughput, latency, execution time, and transaction time. Throughput represents the number of Web service requests served at a given time period. Latency is the round-trip time between sending a request and receiving the response. Execution time is the time taken by a Web Service to process its sequence of activities. Transaction time represents the period of time that passes while the Web Service is completing one complete transaction. Higher throughput, lower latency, lower execution and transaction times represent good performing Web Services.

6.7 WEB SERVICES PERFORMANCE-LIMITATIONS

The overall performance of a Web service depends on application logic, network, and most importantly on underlying messaging and transport protocols such as the SOAP and HTTP it uses. The SOAP protocol is still maturing and harbors a lot of performance and scalability problems. The SOAP protocol uses a multi-step process to complete a communication cycle.

The SOAP request begins with the business logic of your application learning the method and parameter to call from a Web Services Description Language (WSDL) document. This whole process is time

consuming; it requires various levels of XML parsing and XML validation and hence hits the performance of the Web service.

Apart from these factors, the performance of a Web service becomes crucial if it uses the synchronous one where a consuming application is blocked for the time it gets a response from the Web service.

6.8 WEB SERVICES PERFORMANCE-BEST PRACTICES AND SOLUTIONS

Analyzing the above situations, following are some of the useful practices and solutions which we should have in mind while developing and exposing it to outside world. I am dealing with them layer by layer of the Web service stack.

a. **At the Data Layer:** At data layer of a Web service stack, the Web service is only sending/receiving XML messages through the network. Some of the best practices to be followed to achieve the same are listed below:

1. Use of compressed XML for sending the messages over network: SOAP/XML messages are generally bigger in size than normal GET/POST calls, so we should try to compress the message if the network latency is bigger than CPU overhead for compression. Lots of XML compression tools are available from different vendors such as Oracle XDK and there are many open source tools also there to fit the budget.

2. Use of simple data types in SOAP messages as far as possible reduces complexity to great extent. This reduces the processing time and increases maintainability; enhancements are easy to do.

b. At the Logical Layer: At the logical layer, Web service mainly deals in processing SOAP/XML messages. Being XML intensive at this layer, we should try to optimize XML processing steps. Some of the best practices to be followed to achieve the same are listed below:

1. Use of efficient & lightweight parsers: There are many parsers available on the market, each vying for its performance. But, to get performance we should first analyze our application and its requirement. If the application requires big XML files to be exchanged, a SAX parser would be the best over a DOM parser. A DOM parser actually loads the whole document in memory and thus utilizes a considerable amount of the memory and processor. Instead, a SAX parser is event based, is faster, and does not consume much memory.

   Even better, if we know about XML messages that are coming (which is generally the case) from a client/application, we should try to use pull parsers instead of push ones. Using a pull parser will load only that part of XML, which is required by the application that time.

2. Efficient use of XML validation in production mode. XML validation should be kept to minimum during the production mode because it generally slows down the parser. XML validation can be turned OFF for trusted parties after authorization has been done. And, instead of performing XML validation at the parser level, some critical validations, depending on application requirements, can be done at the application level.

3. Web Service Caching: One must cache request results wherever possible in their Web service. Even the WSDL can be cached at the client side to minimize the multiple requests.

b. **At the Presentation Layer:** At the presentation layer, a Web service deals mainly with how to present the responses in an effective manner. Depending on the client, one can apply the following strategies:

1. Repeated SOAP-client calls to access server state can choke a network and degrade the server performance. Cache data on the client whenever possible to avoid requests to the server.

2. Ensure the client data remains up to date by using a call to a service, which blocks until data is changed.

6.9 PERFORMANCE MONITORING

Tracing the performance of a Web service in real time gives lot of hints. We can actually identify the bottlenecks in the execution of a Web service. Once we know the problems, we can actually find solutions to it by applying best practices as mentioned above or by using better hardware. There are lot of tools and methodologies available for monitoring the performance.

6.10 SUMMARY

- QoS is an important requirement of business-to-business transactions necessary element in Web services.
- The various QoS properties need to be addressed in the implementation of Web service applications.
- The properties become even more complex when you add the need for transactional features to Web services.
- Some of the limitations of protocols such as HTTP and SOAP may hinder QoS implementation, but there are a number of ways to provide proactive QoS in Web services.

6.11 QUESTIONS

1. Explain Qos ?
2. Why is Qos important for web services ?
3. Explain web service performance and best practices ?
4. Explain Caching Web Services ?

6.12 REFERENCE

- https://www.oreilly.com/library/view/developing-enterprise-web/0131401602/0131401602_ch09lev1sec2.html
- https://www.developer.com/web-services/quality-of-service-for-web-services-demystification-limitations-and-best-practices-for-performance/
- https://myweb.cmu.ac.th/choosak_s/954376/Chapter%205/Lecture5-QoS%20for%20Web%20Service.pdf