CHAPTER 2

2.1 Web Services and Middleware

Web services:

Web Services is an application integration technology. It acts as a bridge, allowing different software applications to communicate and share data, regardless of their underlying technologies or programming languages.

Example: Imagine an online travel booking website. It needs to integrate with various systems:

- Airlines for flight availability and booking.
- Hotels for room reservations.
- Payment gateways for processing transactions.
- Car rental companies for vehicle bookings.

Web Services allow applications to be integrated more rapidly, easily and less expensively. Web services use standardized protocols (like HTTP and XML/JSON), making integration simpler and faster. This reduces the need for custom-built integration solutions, which are often time-consuming and expensive.

- XML (Extensible Markup Language): A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It's widely used for data exchange over the internet.
- JSON (JavaScript Object Notation): A lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.

The Web did for program-to-user interactions; Web Services are developed for program-to- program interactions. **Example:**

- **Web (Program-to-User):** When you use a search engine (like Google), your browser (the program) sends a request to Google's servers. The servers process your request and send back the search results, which your browser displays.
- **Web Services (Program-to-Program):** A weather application on your smartphone might use a web service provided by a meteorological agency to fetch real-time weather data. The app (program) sends a request to the agency's web service, which returns the data in a structured

format (like JSON).

Web Services allow companies to reduce the cost of doing e-business, to deploy solutions faster and to open up new opportunities. Web services model built on existing and emerging standards such as HTTP, Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI).

- XML Web Services *expose useful functionality* to Web users through a standard Web protocol. In most cases, the protocol used is SOAP.
- XML Web services *provide a way to describe their interfaces in enough detail* to allow a user to build a client application to talk to them. This description is usually provided in an XML document called a Web Services Description Language (WSDL) document.
- XML Web services *are registered* so that potential users can find them easily. This is done with Universal Discovery Description and Integration (UDDI).

Web service is developed in order to distribute an object and serve it to various users in the web environments. It can be also used in the server situations while solving the web-scalability problem of the other distributed object technologies. Fig.2.1 demonstrates the web service architecture.

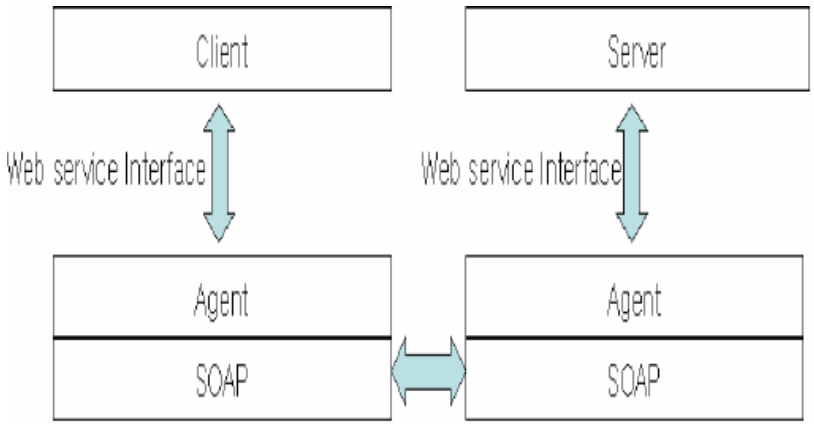

Figure.2.1. Web service architecture

1. A client that wants to be serviced should first find the supported services from the pre-existing *registry* before compiling a code.
2. After finding its services through searching, the client gains the *Web Service Description Language (WSDL)* that a server previously registers. From the WSDL, the client knows the service provider location and the parameters to the found method.
3. After the client binds the described service during the compile time, it calls the local agent whenever the client invokes a method call, and the local agent delivers it to the server side agent through *Simple Object Access Protocol (SOAP)* over HTTP, FTP, SMTP, IIOP, and TCP during the runtime.
4. The server side agent activates the appropriate object, and delivers the calls to the object.

All of the communication methods such as a WSDL, and SOAP exploit the eX*tensible Markup Language (XML)*. WSDL is an XML describing the web service. SOAP is an XML describing the called method, its parameters, and its return value, can be delivered over the HTTP.

2.2 Network Programming

2.2.1 Basic Network Concepts

Network:

- A network is a collection of computers and other devices that can send data to and receive data from each other.
- A network is often connected by wires.
- However, wireless networks transmit data through infrared light and microwaves.

Node:

- Each machine on a network is called a node.
- Most nodes are computers, but printers, routers, bridges, gateways etc.. can also be nodes.
- Nodes that are fully functional computers are also called hosts.

Packet:

- All modern computer networks are packet-switched networks: data traveling on the network is broken into chunks called packets and each packet is handled separately.
- Each packet contains information about who sent it and where it's going.

Protocol:

- A protocol is a precise set of rules defining how computers communicate: the format of addresses, how data is split into packets, and so on.
- There are many different protocols defining different aspects of network communication.

IP:

- IP was designed to allow multiple routes between any two points and to route packets of data around damaged routers*.*

TCP:

- Since there are multiple routes between two points, and since the quickest path between two points may change over time as a function of network traffic and other factors), the packets that make up a particular data stream may not all take the same route.
- Furthermore, they may not arrive in the order they were sent, if they even arrive at all.

UDP:

- UDP is an unreliable protocol that does not guarantee that packets will arrive at their destination or that they will arrive in the same order they were sent.

Ports:

- Each computer with an IP address has several thousand logical ports.
- Each port is identified by a number between 1 and 65,535. Each port can be allocated to a particular service.
- Port numbers 1 through 255 are reserved by IP for well-known services. A well-known service is a service that is widely implemented which resides at a published, "well-known", port. If you connect to port 80 of a host, for instance, you may expect to find an HTTP server.

2.2.2 Network Programming Concepts:

Internet:

The Internet is the world's largest IP-based network. The Internet is all about connecting machines together. One of the most exciting aspects of Java is that it incorporates an easy-to-use, cross- platform model for network communications.

What is a Socket?

Sockets are a means of using IP to communicate between machines, so sockets are one major feature that allows Java to interoperate with legacy systems by simply talking to existing servers using their pre-defined protocol.

Other common protocols are layered on top of the Internet protocol: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). Applications can make use of these two protocols to communicate over the network.

Internet Addresses or IP Addresses

Every network node has an address, a series of bytes that uniquely identify it. Internet addresses are manipulated in Java by the use of the InetAddress class. InetAddress takes care of the Domain Name System (DNS) look-up and reverse look-up; IP addresses can be specified by either the host name or

the raw IP address. InetAddress provides methods to getByName(), getAllByName(), getLocalHost(), getAddress(), etc.

IP addresses are a 32-bit number, often represented as a "quad" of four 8-bit numbers separated by periods. They are organized into classes (A, B, C, D, and E). For example 126.255.255.255

Client/Server Computing

You can use the Java language to communicate with remote file systems using a client/server model. A server listens for connection requests from clients across the network or even from the same machine. Clients know how to connect to the server via an IP address and port number. Upon connection, the server reads the request sent by the client and responds appropriately. In this way, applications can be broken down into specific tasks that are accomplished in separate locations.

The data that is sent back and forth over a socket can be anything you like. Normally, the client sends a request for information or processing to the server, which performs a task or sends data back. The IP and port number of the server is generally well-known and advertised so the client knows where to find the service.

How UDPclients and UDPservers communicate over sockets

Creating UDP Servers:

To create a server with UDP, do the following:

1. Create a DatagramSocket attached to a port.

```
int port = 1234;
DatagramSocket socket = new DatagramSocket(port);
```

2. Allocate space to hold the incoming packet, and create an instance of DatagramPacket to hold the incoming data.

```
byte[] buffer = new byte[1024];
DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
```

3. Block until a packet is received, then extract the information you need from the packet.

```
// Block on receive()
socket.receive(packet);
// Find out where packet came from
// so we can reply to the same host/port
InetAddress remoteHost = packet.getAddress();
int remotePort = packet.getPort();
// Extract the packet data
byte[] data = packet.getData();
```

The server can now process the data it has received from the client, and issue an appropriate reply in response to the client's request.

Creating UDP Clients

Writing code for a UDP client is similar to what we did for a server. Again, we need a DatagramSocket and a DatagramPacket. The only real difference is that we must specify the destination address with each packet, so the form of the DatagramPacket constructor used here specifies the destination host and port number. Then, of course, we initially send packets instead of receiving.

1. First allocate space to hold the data we are sending and create an instance of DatagramPacket to hold the data.

```
byte[] buffer = new byte[1024];
int port = 1234;
InetAddress host = InetAddress.getByName("magelang.com");
DatagramPacket packet = new DatagramPacket(buffer, buffer.length, host, port);
```

2. Create a DatagramSocket and send the packet using this socket.

```
DatagramSocket socket = new DatagramSocket();
socket.send(packet);
```

The DatagramSocket constructor that takes no arguments will allocate a **free local port to use**. You can find out what local port number has been allocated for your socket, along with other information about your socket if needed.

```
// Find out where we are sending from
InetAddress localHostname = socket.getLocalAddress();
int localPort = socket.getLocalPort();
```

The client then waits for a reply from the server. Many protocols require the server to reply to the host and port number that the client used, so the client can now invoke socket.receive() to wait for information from the server.

How TCPclients and TCPservers communicate over sockets

Creating TCP Servers:

To create a TCP server, do the following:

1. Create a ServerSocket attached to a port number.

```
ServerSocket server = new ServerSocket(port);
```

2. Wait for connections from clients requesting connections to that port.

```
// Block on accept()
Socket channel = server.accept();
```

You'll get a Socket object as a result of the connection.

3. Get input and output streams associated with the socket.

out = new PrintWriter (channel.getOutputStream());

reader = new InputStreamReader (channel.getInputStream());

in = new BufferedReader (reader);

Now you can read and write to the socket, thus, communicating with the client.

String data = in.readLine();

out.println("Hey! I heard you over this socket!");

When a server invokes the accept() method of the ServerSocket instance, the main server thread blocks until a client connects to the server; it is then prevented from accepting further client connections until the server has processed the client's request. This is known as an *iterative* server, since the main server method handles each client request in its entirety before moving on to the next request. Iterative servers are good when the requests take a known, short period of time. For example, requesting the current day and time from a time-of-day server.

Creating TCP Clients:

To create a TCP client, do the following:

1. Create a Socket object attached to a remote host, port.

Socket client = new Socket(host, port);

When the constructor returns, you have a connection.

2. Get input and output streams associated with the socket.

out = new PrintWriter (client.getOutputStream());

reader = new InputStreamReader (client.getInputStream());

in = new BufferedReader (reader);

Now you can read and write to the socket, thus, communicating with the server.

out.println("Watson!" + "Come here...I need you!");

String data = in.readLine();

2.4 Low Level Data Communication

Data communications are the exchange of data between two devices via some form of transmission medium such as a wire cable.

TCP/IP(Transmission Control Protocol/Internet Protocol)

- The Protocol upon which the whole Internet is based
  - Each node must be configured for TCP/IP to function properly.
- A software-based protocol
- TCP/IP is basically the binding together of Internet Protocols used to connect hosts on the internet- Main ones are IP and TCP
- TCP and IP have special packet structure

- IP (Internet Protocol) is responsible for delivering packets of data between systems on the internet and specifies their format. Packets forwarded based on a four byte destination IP address (IP number)
- IP DOES NOT MAKE GUARANTEES! It is very simple - essentially: *send and forget*.
- TCP (Transmission Control Protocol) is responsible for verifying the correct delivery of data/packets from client to server. Data can be lost - so TCP also adds support to detect errors and retransmit data until completely received.
- Together these help form TCP/IP - a means of specifying packets, and delivering them safely.
- There are other protocols in TCP/IP - such as User Datagram Protocol UDP. UDP is a simpler alternative to TCP for aiding the delivery of packets. It makes no guarantees regarding delivery - but does guarantee *data integrity.* UDP also has no flow control, i.e. if messages are sent too quickly, data may be lost

2.4.1 IP Packet Structure

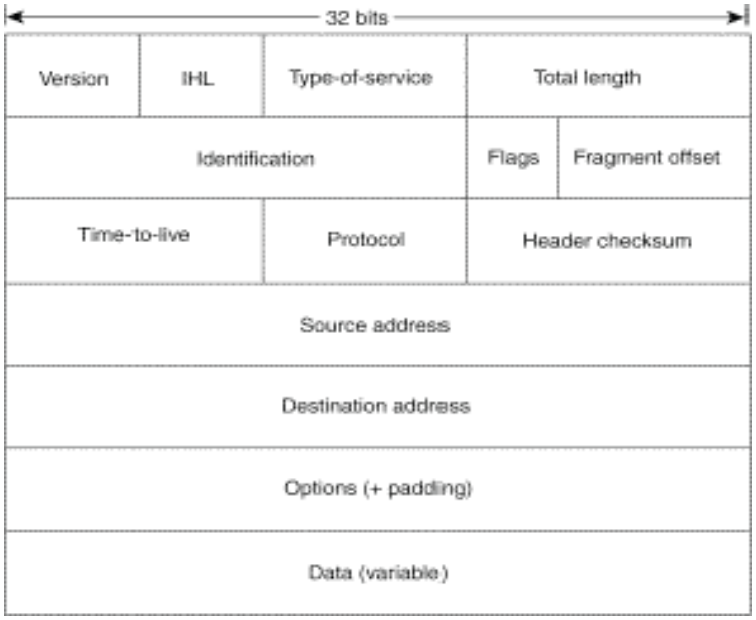

Figure.2.2. IP Packet Structure

IP uses a *Datagram* to transfer *packets* between *end systems* (usually computers) using *routers*. There are fourteen fields in an IP Packet .

Version — Indicates the version of IP currently used.

IP Header Length (IHL) — Indicates the datagram header length in 32-bit words.

Type-of-Service — Specifies how an upper-layer protocol would like a current datagram to be handled, and assigns datagram various levels of importance.

Total Length — Specifies the length, in bytes, of the entire IP packet, including the data and header.

Identification — Contains an integer that identifies the current datagram. This field is used to help piece together datagram fragments.

Flags — Consists of a 3-bit field of which the two low-order (least-significant) bits control fragmentation. The low-order bit specifies whether the packet can be fragmented. The middle bit specifies whether the packet is the last fragment in a series of fragmented packets. The third or high-order bit is not used.

Fragment Offset — Indicates the position of the fragment's data relative to the beginning of the data in the original datagram, which allows the destination IP process to properly reconstruct the original datagram.

Time-to-Live — Maintains a counter that gradually decrements down to zero, at which point the datagram is discarded. This keeps packets from looping endlessly.

Protocol — Indicates which upper-layer protocol receives incoming packets after IP processing is complete.

Header Checksum — Helps ensure IP header integrity.

Source Address — Specifies the sending node.

Destination Address — Specifies the receiving node.

Options — Allows IP to support various options, such as security.

Data — Contains upper-layer sent in packet.

2.4.2 TCP Packet Structure:

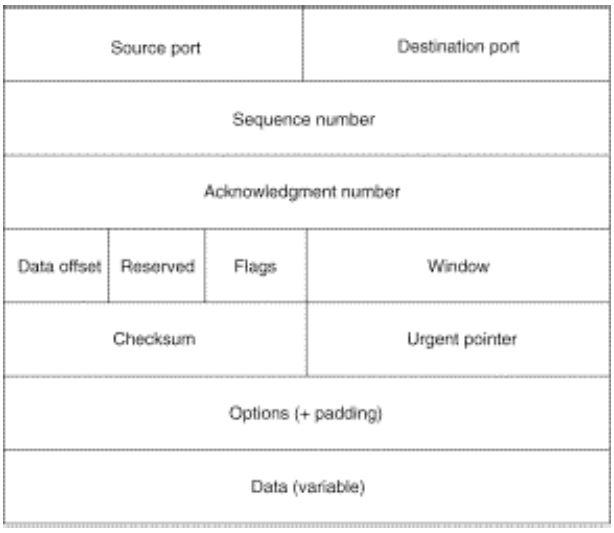

Figure.2.2. TCP Packet Structure

There are 12 fields in TCP Packet:

Source Port and **Destination Port** — Identifies points at which upper-layer source and destination processes receive TCP services.

Sequence Number — Usually specifies the number assigned to the first byte of data in the current message. In the connection-establishment phase, this field also can be used to identify an initial sequence number to be used in an upcoming transmission.

Acknowledgment Number — Contains the sequence number of the next byte of data the sender of the packet expects to receive.

Data Offset — Indicates the number of 32-bit words in the TCP header.

Reserved — Remains reserved for future use.

Flags — Carries a variety of control information, including the SYN and ACK bits used for connection establishment, and the FIN bit used for connection termination.

Window — Specifies the size of the sender's receive window (that is, the buffer space available for incoming data).

Checksum — Indicates whether the header was damaged in transit.

Urgent Pointer — Points to the first urgent data byte in the packet.

Options — Specifies various TCP options.

Data — Contains upper-layer sent in packet.