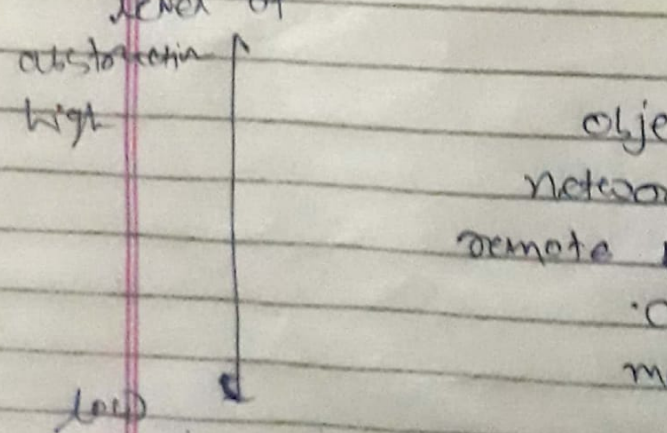
by MV

Distributed computing Paradigms :

⇒ Interprocess comm. : A distributed app. require the participation of two or more independent entities (processes). To do so, the processes must have the ability to exchange data among themselves.

⇒ Event synchronization :
In a distributed app, the sending & receiving of the data among the participants of a distributed application must be synchronized



Message passing paradigm

- The most fundamental paradigm
-
- The msg is delivered to a mail

Restful Webservices

RESTful architecture :

⇒ REST stands for REpresentational State Transfer. REST is web standards based architecture & uses HTTP protocol

⇒ In REST arch., a REST server simply provides access to resources & REST client accesses & modifies the resources

⇒ Here each resource is identified by URLs/ global IDs.

⇒ REST uses various representation to represent a resource like text, JSON, XML.

⇒ JSON is most popular one.

Intro. to RESTful web services.

⇒ A web service is a collection of open protocols & standards used for exchanging data betⁿ applications or systems

⇒ This interoperability (e.g. betⁿ Java & python, or windows & Linux apps) is due to the use of open standards.

⇒ Web services based on REST archi. are known as.

⇒ These web services uses HTTP methods to implement the concept of REST archi.

⇒ A RESTful web service usually defines a URI, Uniform Resource Identifies

#	SOAP	REST
⇒	It is a protocol	⇒ It is an architectural style
⇒	SOAP is stands for simple object access protocol	⇒ REST stands for Representational State Transfer.
⇒	SOAP can't use REST bcz it is a protocol	⇒ REST can use SOAP web service bcz it is a concept & can use any protocol like HTTP, SOAP.
⇒	SOAP uses services interfaces to expose the business logic	⇒ REST uses URI to expose business logic
⇒	JAX-WS is the java API for SOAP web services	⇒ JAX-RS is the java API for RESTful web services
⇒	SOAP defines standards to be strictly followed.	⇒ REST does not define too much standards like SOAP
⇒	SOAP requires more bandwidth & resource than REST	⇒ REST requires less bandwidth & resource
⇒	SOAP defines its own security	⇒ RESTful web services inherits security measures from the underlying transport
⇒	SOAP permits XML data format only	⇒ REST permits different data format such as plain text, HTML, XML, JSON etc
⇒	SOAP is less preferred than REST.	⇒ REST is more preferred than SOAP.

Java Web Service API

JAX WS (SOAP)

RPC style

document style

JAX-RS (RESTful)

Jersey

RESTeasy

JAX-RS annotations

→ package → javax.ws.rs

path → It defines the URI path. It can be specified on class as method

Pathparam → represents the parameters of the URI path specified method

GET → ~~represents the parameter~~ responds to GET req

POST → " " POST "

PUT → " " PUT "

HEAD → specifies method responds to HEAD req

DELETE → " " DELETE "

Produces → defines media type for response such as XML, JSON, PLAIN etc. It defines the media type that the methods of a resource class can produce. MessageBodyWriters can produce.

Resource :-

⇒ REST archi. treats every content as a resource. It can be . . .

⇒ . . . .

Representation of Resources

→ A resource in REST is a similar Object in OOP or is like an Entity in a DB.

⇒ Once a resource is identified then its representation is to be decided using a standard format so that the server can send the resource in the above said format & client can understand the same format.

URL vs URI

→ For starters, URI → uniform re
URL →

→ URLs are a subset of URIs. That means all URLs are URIs. It doesn't work the opposite way though

⇒ Ex:- Your name could be a URI bcz it identifies you.

Types of URI

→ two types of resource URIs

(a) Instance resource URIs

weather / zipcode / 387001

(2) Collection resource URIs

Eg: in.bookmyshow.com/ahmedabad/movies

Example: /messages/{messageid}/comments/{commentid}/message ? offset=30 & limit=10.

ESB. (Enterprice service Bus)

⇒ It is flexible connectivity infrastructure for integrating systems, app^ns & services to powes your soa.

⇒ It provides fundamental services for complex archi. via an event-driven & standards-based messaging engine (the bus).

Why need ESB ?

⇒ services are generally autonomous & availability of a resvice at a certain moment of time can't be gurcunteed.

→ It reduces the no. of point to point connections bet^n communicating app^ns

⇒ It is easies to monitor for failure & misbehaviours in highly complex systems & allows easies changing of components.

⇒ Every client directs all its requests through the ESB instead of passing it directly to a potential resves.

⇒ This indirection allows the ESB to monitor & log the traffic.

Features of ESB

- ⇒ Management Capabilities

(1) Routing msgs bet^n services
(2) Converting transport protocols bet^n requestor & service
(3) Transforming msg format bet^n requestors & service
(4) Handling Business events from dispersate sources.

- ⇒ quality-of-service support

1) reliability
2) fault-tolerance
3) security

- ⇒ Connectors for a broad range of technologie (such as J2EE & .NET components, messaging middleware, legacy app^ns & TP monitors)

Services supported by ESB

- Mediation services
  1) Routing
  2) Transformation

- Event services
  1) publish & subscribe

- Trans post services
  1) Synchronous / Asynchronous
  2) Persistent / Non-persistent
  3) Loosely-coupled / Tightly-coupled

Mediation services

- Mediation capabilities
  - Transformations: XML-to-XML translation
  - Msg Validation
  - Content & quality service selections
  - content based routing
  - Customized logging, metering & monitoring
  - Policy management

ESB implementation:

1) Java Business Integration (JBI):
   provides standard interfaces for integration components & protocols to plug into. JBI is based on JSR 208 (an extension of Java 2 EE) & is specific for JBI.

2) Open ESB
   It is an open-source implementation of JSR-208 (JBI).

3) WSO2 & servicemix (from apache) are also available

SOA lifecycle of projects.

Model, Management, Assemble, Deploy
Governance & best practices

1) Model
- Gathes requirements
- model & simulate
- Design.

The project is started by gathering business requirements designing, simulating & optimizing their desired business processes. We make sure that we are setting the right steps in motion before further action is taken.

2) Assemble
- Discover
- construct & test
- compose.

Once we have optimized the business process, we implement it by combining newly created & reused existing services to form composite apps.

3) Deploy
- Integrate people
- " processes
- " information

The assets are then deployed into a secure & integrated environment taking advantage of specialized services that provide support for integrating ppl, processes & info.

This level of integration help to ensure...

4) Manage
- Manage IT resources
- " services
- Monitor business metrics
- Monitor services & underlying resources.

The customers manage & monitor the composite apps & underlying resources from both an IT & a business perspective. Info. gathered during the manage phase is used to gain real-time insight into business processes enabling better business decisions & feeding info. back into the lifecycle for continuous process improvement.

5) Governance & best practices (during all above four phases)
- sharing & reuse of services
- Establish decision rights
- policies, measurement & control.

It provides guidance & oversight (these, supervision) for the SOA project.

Model	Assemble	Deploy	Manage
UML	SDO BPEL J2EE Eclipse	WS-BPEL	common base events

Implementation of RESTful Web services &

New project → Web app
Name: RestfulService [Finish]

Add JAR → jersey jars → add all jar files

New → Java class → Uses
package name → com.rest

Uses.java

```
import java.io.Serializable;
import javax.xml.bind.annotation.XmlElement;
        XmlRootElement;

@XmlRootElement(name = "Uses")
public class Uses implements Serializable
{
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
            profession;
```

// default constructor
// 3 argu. constructor
// getter & setter method
on setter methods. write "@XmlElement"

Create new java class : UserDao,
Package com.test

```
public class UserDao {
    public List<User> getAllUsers()
    {
        List<User> userList = null;
        try {
            File file = new File("User.dat");
            if (!file.exists())
            {
                User user = new User(1, "Rashmi"
                                    "student");
                userList = new ArrayList<User>();
                userList.add(user);
                saveUserList(userList);
            }
            else
            {
                FileInputStream fis = new ... (File);
                ObjectInputStream ois = new ... (fis);
                userList = (List<User>) ois.readObject();
                ois.close();
            }
            catch(IOException e) { ... }
            catch(ClassNotFoundException e) { ... }
        }
        return userList;
```

```
public void saveUsesList (List<Uses> )
{ try {
    File file = new File ("Users.dat");
    FileOutputStream fos;
    fos = new FileOutputStream (file);
    ObjectOutputStream obs = new
        ObjectOutputStream (fos);
```

New java class : UsesService
pkg : com.rest

```
@Path ("UsesService")
public class UsesService {
    UsesDao usesDao = new UsesDao();
    @GET
    @Path ("/users")
    @Produces (MediaType.APPLICATION_XML)
    public List<Uses> getUsers()
    {
        return usesDao.getAllUsers();
    }
}
```

In WEB-INF → create web.xml

```
<display-name>RestfulService</...>
```

<servlet>
<servlet-name>JerseyRESTfulApplication</...>
<servlet-class>org.glassfish.jersey.servlet.ServletContainer</...>
<init-param>
<param-name>jersey.config.server.provider.packages</...>.
<param-value>com.rest</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>JerseyRESTfulApplication</servlet-name>
<url-pattern>/rest/*</url-pattern>
</servlet-mapping>

Create
New Servlet
Name : JerseyRESTfulApplication

pkg : com.rest.

Add web.xml content
Build & Deploy.

Open Postman.

URL http://localhost:/JerseyRESTfulApplication/rest/userService/users

send

29/8/1

Date : / /

⇒ Create new web app - ServiceFromDb.

~~Add~~

Database → connect → derby

⇒ New → WebService → Restful ws from db

Data source : jdbc/sample

customer ~~Add~~.

☑ Include selected table

next

pkg : entities.

Finish