

Experiment No.: 1

Modeling using xADL

Learning Objective: Student should be able to do Modeling using xADL

Tool:- Apigen , Data Binding Library

Theory:

xADL is a highly-extensible software architecture description language (ADL). It is used to describe various aspects of the architecture of a software system. The architecture of a software system is its high-level design; design at the level of components, connectors, and their configurations. Like other ADLs such as Rapide, Darwin, and Wright, xADL's core models the four most common architectural constructs, namely:

- **Components** : (the loci of computation),
- **Connectors** : (the loci of communication),
- **Interfaces** : (the exposed entry and exit points for components and connectors), and
- **Configurations** : (topological arrangements of components and connectors as realized by links).
- xADL is an XML-based language. XML, the eXtensible Markup Language, was originally created to annotate, or "mark up" text documents with semantic information. Elements of text are marked up using tags, or special strings, that delimit a section of text. Tags begin with an open angle-bracket (<) and end with a closing angle-bracket (>). In XML documents, tags generally come in pairs, signifying the start and end of a text element. Start tags contain a tag name immediately after the opening angle-bracket, and end-tags contain the same name, prefaced by a forward slash (/) immediately after the opening angle-bracket. Elements may be nested as necessary, but may not overlap. An example of some marked up text in XML might be:


```
<name><first>Herb</first> <last>Mahler</last></name>
```
- To HTML users, this format may look familiar. This is because XML and HTML both share a common historical ancestor, SGML (the Standard Generalized MarkupLanguage). In HTML, however, there is a finite set of allowed tags, each of which has a specific meaning dedicated to screen layout. So, tags like <H1> in HTML indicate that a text element is to be laid out in the Heading 1 style (usually large and bold, although this varies depending on the layout engine used), but do not indicate any other semantics

about the element--is it a story headline? Is it someone's name? This information is not present in HTML.

- This lack of a static set of allowed tags introduces a new problem into XML applications. What tags are allowed and what do they mean? How do two parties sharing marked-up documents come to an agreement on what elements are allowed, and where? How can they ensure that their information is marked up in a consistent way that is meaningful to both of them?
- The answer to this problem is to introduce a *meta-language*, a language for defining languages. In XML, meta-languages define what elements are allowed, where they are allowed to occur (and what their cardinality is), and what data may be part of each element. The XML 1.0 standard included such a meta-language, called the DTD (Document Type Definition) language. The DTD meta-language was sufficient for expressing XML-based languages as a set of production rules, much like a BNF (Backus-Naur Form) grammar, but proved insufficient for certain types of applications. To remedy some issues that users had with DTDs, the W3C (World Wide Web Committee) drafted anew meta-language for XML called XML Schema. XML schemas are more expressive than DTDs, they have an XML-like syntax (DTDs do not), and they allow type relationships among element types much like object-oriented inheritance.
- The development of XML schemas made it much easier for developers to create and evolve XML-based languages, and fostered the development of modular languages like xADL. A full treatment of XML is far out of scope for this guide.
- xADL's XML basis means that data in xADL is arranged hierarchically. Connections between data elements that are not hierarchically arranged are managed using simple XML links; xADL's tool support facilitates navigating these links.
- As an XML-based language, xADL documents are readable and writable by hand, as simple text documents. However, because xADL is defined in multiple schemas, each schema having its own XML namespace, the actual code can get quite complicated. For example, this is a real component description in xADL:

The xADL Type System

xADL adopts the more traditional types-and-instances model found in many programming languages. In this model, components, connectors, and interfaces all have types (called *component types*, *connector types*, and *interface types*, respectively).

Links do not have types because links do not have any architectural semantics. The relationships between types, structure, and instances are shown in the following table:

<u>Instance (Run-time)</u>	<u>Structure (Design-time)</u>	<u>Type (Design-time)</u>
<u>Component Instance</u>	<u>Component</u>	<u>Component Type</u>
<u>Connector Instance</u>	<u>Connector</u>	<u>Connector Type</u>
<u>Interface Instance</u>	<u>Interface</u>	<u>Interface Type</u>
<u>Link Instance</u>	<u>Link</u>	<u>(None)</u>
<u>Group</u>	<u>Group</u>	<u>(None)</u>

- Component Types
 - Sub architectures for Component Types
 - Signatures
- Connector Types*
 - Sub architectures for Component Types
 - Signatures
- Interface Types

Modeling of xADL

The instances schema gives xADL the ability to model running instances of architectural constructs like components, connectors, interfaces, and links. However, much work on software architecture is centered around the *design* of the architecture, rather than capturing properties of a running one.

For the purposes of this discussion, we make a distinction between architecture instances, which exist at run-time, and structural elements, which exist at design-time.

<u>Run-Time</u>	<u>Design-Time</u>
<u>Instances</u>	<u>Structure</u>

The xADL constructs available for modeling architectural structure mirror almost exactly those available for modeling architecture instances. The constructs defined in the instance schema are:

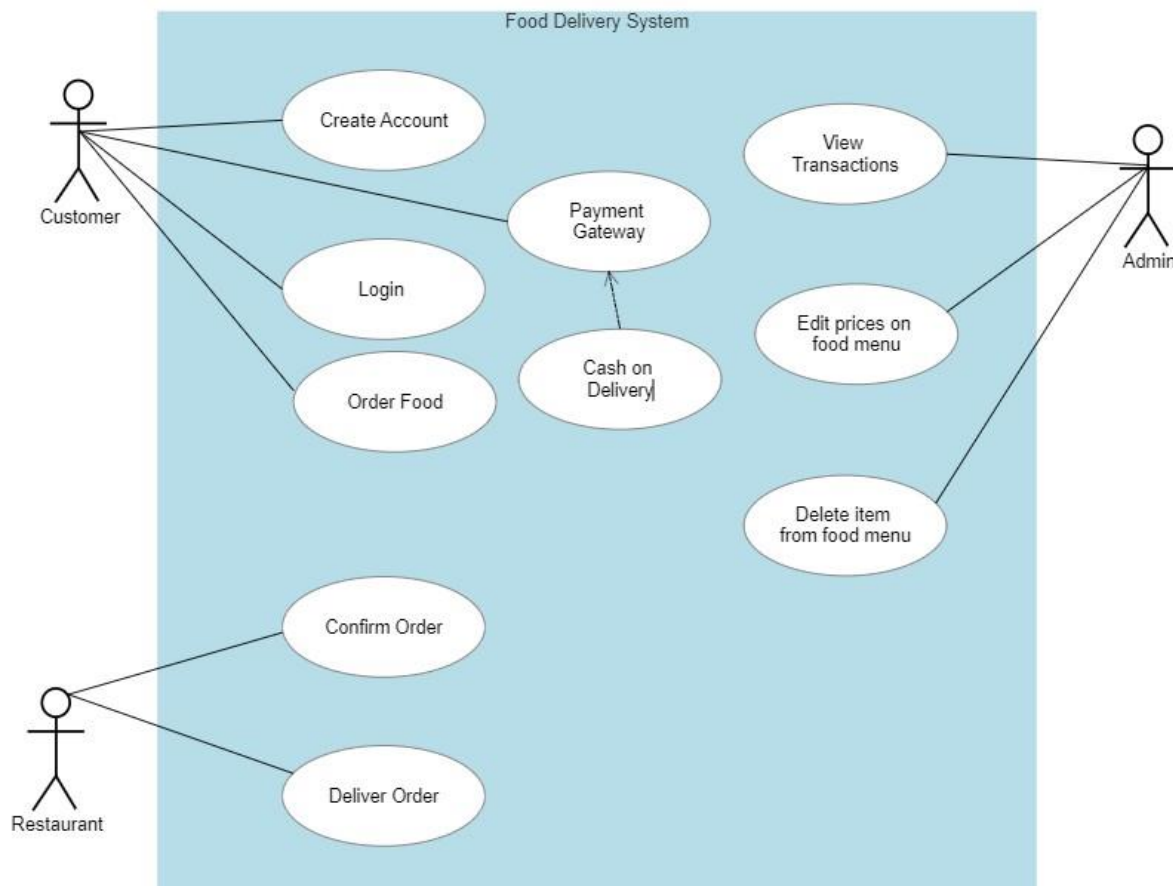
- Components
- Connectors

- Interfaces
- Links
- General Groups

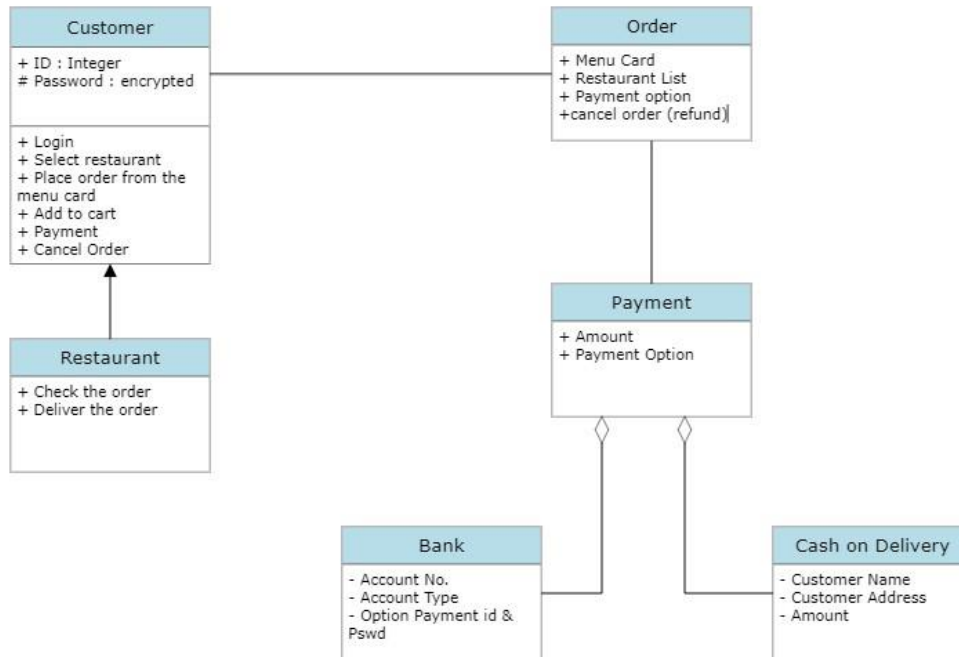
FORMAL DESCRIPTIVE LANGUAGE

- It should have login and registration features
- It should display menu and restaurant list
- Select restaurant from list and select food from menu
- Add to cart
- Customer selects payment option
- Payment options include cash on delivery, online transaction
- Check order and delivery order for restaurants
- Cancel order and refund

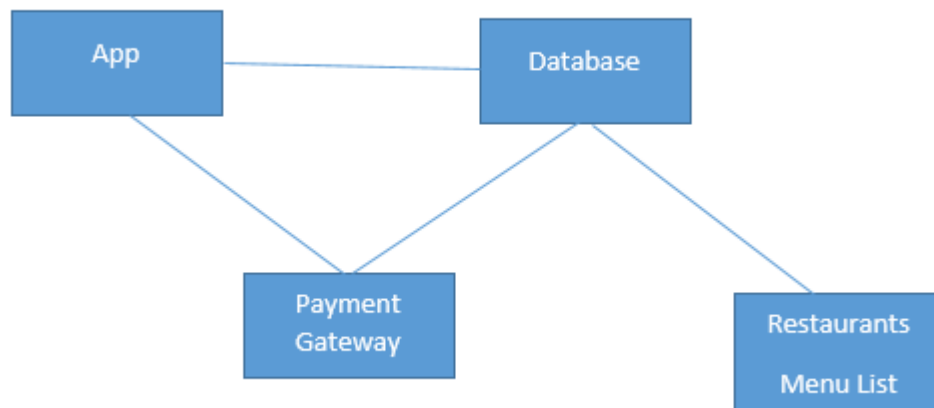
UML Diagram



Class Diagram



Box and Line:



"component" is used as a structural construct, as opposed to "component instance," which is used to describe a run-time instance. Similarly, "connector," "interface," and "link" are used instead of "connector instance," "interface instance," and "link instance."

Result and Discussion:

Learning Outcomes: Students should be able to

LO1: Define xADL.

LO2: Identify xADL Command.

LO3: Apply xADL Command for desired Output.

Course Outcomes: Upon completion of the course students will be able to do Modeling using xADL.

Conclusion:

Viva Questions:

1. Define xADL.
2. Explain xADL.
3. Explain syntax of xADL.
4. Explain modeling using xADL
5. Explain xADL type system.

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	

Experiment No.: 2

Visualization using xADL 2.0

Learning Objective: Student should understand and able to do Visualization using xADL 2.0

Tool:- Apigen or Data binding Library.

Theory:

What is xADL 2.0?

xADL 2.0 is a software architecture description language (ADL) developed by the University of California, Irvine for modeling the architecture of software systems. Unlike many other ADLs, xADL 2.0 is defined as a set of XML schemas. This gives xADL 2.0 unprecedented extensibility and flexibility, as well as basic support from the many available commercial XML tools.

The current set of xADL 2.0 schemas includes modeling support for:

- run-time and design-time elements of a system;
- support for architectural types;
- advanced configuration management concepts such as versions, options, and variants;
- product family architectures; and
- architecture "diff"ing (initial support).

xADL 2.0 is also an application of xArch, a core XML schema defined jointly by UCI and Carnegie Mellon University.

xADL 2.0 includes constructs that permit modeling of:

- Architecture structure and types,
- Product families (architectural versions, options, and variants), and
- Implementation mappings (mappings from architecture types to their implementations).

xADL 2.0's modules are defined as XML Schemas, making xADL 2.0 an XML-based language. All xADL 2.0 documents i.e. architecture descriptions are XML documents that are valid with respect to the xADL 2.0 schemas.

xADL 2.0 can be extended by end-users to optimize the language for particular domains. Tools are available that provide users with support for both using existing modules (schemas) and creating and manipulating their own extensions to xADL 2.0.

xADL 2.0 is an XML-based language. XML, the extensible Markup Language, was originally created to annotate, or "mark up" text documents with semantic information. Elements of text are marked up using tags, or special strings, that delimit a section of text. Tags begin with an open angle-bracket (<) and end with a closing angle-bracket (>). In XML documents, tags generally come in pairs, signifying the start and end of a text element. Start tags contain a tag name immediately after the opening angle-bracket, and end-tags contain the same name, prefaced by a forward slash (/) immediately after the opening angle-bracket. Elements may be nested as necessary, but may not overlap. An example of some marked up text in XML might be:

```
<name><first>Herb</first> <last>Mahler</last></name>
```

Constructs Defined in the Instances Schema :

A common theme throughout xADL 2.0, defined first in the instance schema, is that of IDs and Descriptions. Many elements in xADL 2.0 have an ID and/or a Description. Identifiers are assumed to be unique to a particular document. They do not necessarily have to be human-readable, although it helps if they are. Descriptions are intended to be human-readable identifiers of the described constructs.

Furthermore, the instance schema defines an element type called an XMLLink that is used over and over again in other xADL 2.0 schemas. XMLLinks are links to other XML elements. xADL 2.0 borrows the linking strategy from the XLink standard. However, because of poor support for the XLink standard in terms of real tools, xADL 2.0 document authors are advised to follow the following simplified convention for specifying XMLLinks.

In xADL 2.0, anything with an ID can be the target (i.e. the thing being pointed to) for an XMLLink. Every XMLLink has two parts (implemented as XML attributes), type and href; these are defined by the XLink standard. For xADL 2.0 XMLLinks, the type field should always be the string simple, indicating a simple XLink. The href field should be filled out with a URL such as:

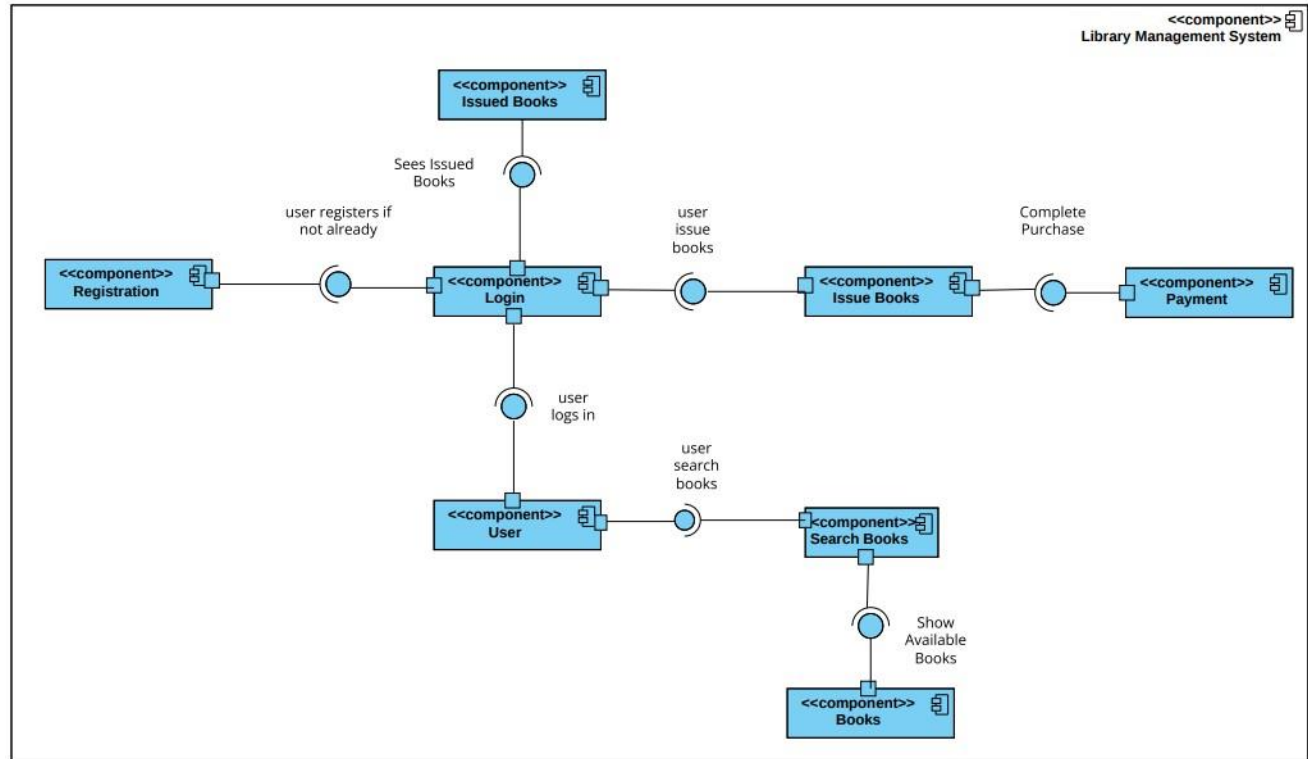
<http://server/directory/document.xml#id>

This is a fairly standard fully-specified URL, linking to a document, but using the *anchor* part of the URL (i.e. the part after the pound sign ('#')) to indicate the identifier of the specific target element. Of course, if you are linking to an element in the same document, it is often preferable to link using a relative URL, such as:

#id

Which would be the element with ID *id* in the current document. So, two examples of valid hrefs might be:

<http://www.isr.uci.edu/foo/bar/archstudio.xml#ArchEdit>
#ArchEdit (from within the file archstudio.xml)



Result and Discussion:

Learning Outcomes: Students should have be able to understand LO1:

Define xADL 2.0.

LO2: Identify xADL Command.

LO3: Apply xADL Command for desired Output.

Course Outcomes: Upon completion of the course students will be able to do visualization using xADL 2.0.

Conclusion:

Viva Questions:

1. Define xADL 2.0.
2. Explain xADL 2.0.
3. Explain syntax of xADL 2.0.
4. Explain syntax of Constructs Defined in the Instances

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	

Experiment No.: 3

Creating Web service

Learning Objective: Student should be able to understand creating web services using and also different web services components.

Theory: A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.

The architecture of web service interacts among three roles: service provider, service requester, and service registry. The interaction involves the three operations: publish, find, and bind. These operations and roles act upon the web services artifacts. The web service artifacts are the web service software module and its description.

Web Service Architecture

There are three roles in web service architecture:

- i. **Service Provider:** From an architectural perspective, it is the platform that hosts the services.
- ii. **Service Requestor:** Service requestor is the application that is looking for and invoking or initiating an interaction with a service. The browser plays the requester role, driven by a consumer or a program without a user interface.
- iii. **Service Registry:** Service requestors find service and obtain binding information for services during development.

Different process in web service Architecture:

- Publication of service descriptions (**Publish**)
- Finding of services descriptions (**Find**)
- Invoking of service based on service descriptions (**Bind**)

Publish: In the publish operation, a service description must be published so that a service requester can find the service.

Find: In the find operation, the service requestor retrieves the service description directly. It can be involved in two different lifecycle phases for the service requestor:

- At design, time to retrieve the service's interface description for program development.
- And, at the runtime to retrieve the service's binding and location description for invocation.

Bind: In the bind operation, the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact, and invoke the service.

Artifacts of the web service

There are two artifacts of web services:

- Service
- Service Registry

Service: A service is an **interface** described by a service description. The service description is the implementation of the service. A service is a software module deployed on network-accessible platforms provided by the service provider. It interacts with a service requestor. Sometimes it also functions as a requestor, using other Web Services in its implementation.

Service Description: The service description comprises the details of the interface and implementation of the service. It includes its data types, operations, binding information, and network location. It can also categorize other metadata to enable discovery and utilize by service requestors. It can be published to a service requestor or a service registry.

Web Service Implementation:

Requirements Phase: The objective of the requirements phase is to understand the business requirement and translate them into the web services requirement. The requirement analyst should do requirement elicitation (it is the practice of researching and discovering the requirements of the system from the user, customer, and other stakeholders). The analyst should interpret, consolidate, and communicate these requirements to the development team. The requirements should be grouped in a centralized repository where they can be viewed, prioritized, and mined for interactive features.

Analysis Phase: The purpose of the analysis phase is to refine and translate the web service into conceptual models by which the technical development team can understand. It also defines the high-level structure and identifies the web service interface contracts.

Design Phase: In this phase, the detailed design of web services is done. The designers define web service interface contract that has been identified in the analysis phase. The defined web service interface contract identifies the elements and the corresponding data types as well as mode of interaction between web services and client.

Coding Phase: Coding and debugging phase is quite similar to other software component-based coding and debugging phase. The main difference lies in the creation of additional web service interface wrappers, generation of WSDL, and client stubs.

Test Phase: In this phase, the tester performs interoperability testing between the platform and the client's program. Testing to be conducted is to ensure that web services can bear the maximum load and stress. Other tasks like profiling of the web service application and inspection of the SOAP message should also perform in the test phase.

Deployment Phase: The purpose of the deployment phase is to ensure that the web service is properly deployed in the distributed system. It executes after the testing phase. The primary task of deployer is to ensure that the web service has been properly configured and managed. Other optional tasks like specifying and registering the web service with a UDDI registry also done in this phase.

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).

- **Service transport** is responsible for actually transporting XML messages between two computers.

Hyper Text Transfer Protocol (HTTP)

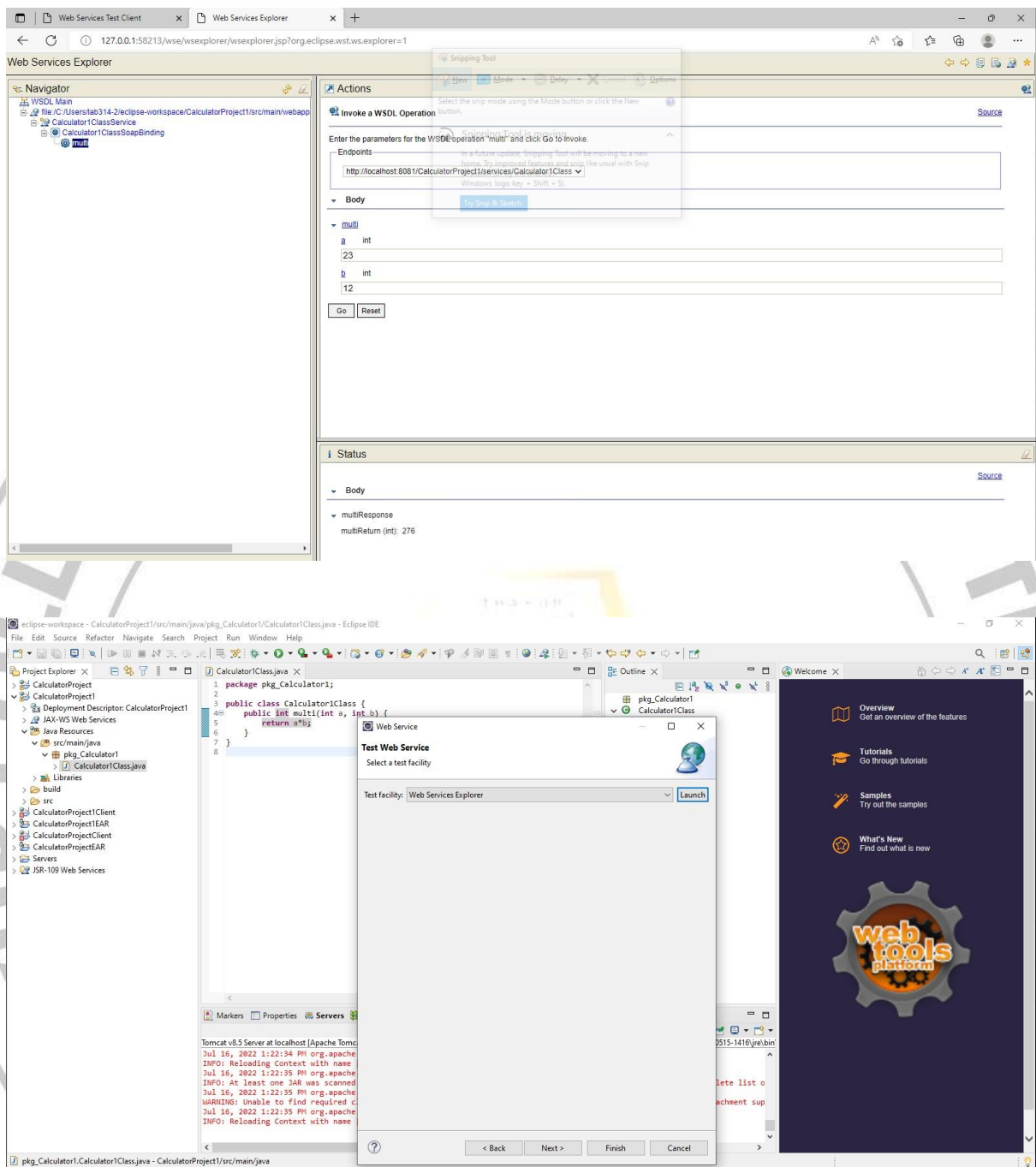
Currently, HTTP is the most popular option for service transport. HTTP is simple, stable, and widely deployed. Furthermore, most firewalls allow HTTP traffic. This allows XMLRPC or SOAP messages to masquerade as HTTP messages. This is good if you want to integrate remote applications, but it does raise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

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 BEE

OUTPUT:



The screenshot displays the Eclipse IDE interface with the Web Services Explorer and Calculator1Class.java files open.

Web Services Explorer:

- Navigator:** Shows the project structure with `Calculator1ClassService` and `Calculator1ClassSoapBinding`.
- Actions:** Displays the `Invoke a WSDL Operation` dialog. The `Endpoints` field is set to `http://localhost:8081/CalculatorProject/Services/Calculator1Class`. The `Body` section shows the `multi` operation with parameters `a` (int, 23) and `b` (int, 12). The `Status` section shows the `multiResponse` operation with the `multiReturn (int): 276`.

Calculator1Class.java:

```
1 package pkg_calculator1;
2
3 public class Calculator1Class {
4     public int multi(int a, int b) {
5         return a*b;
6     }
7 }
8
```

Test Web Service:

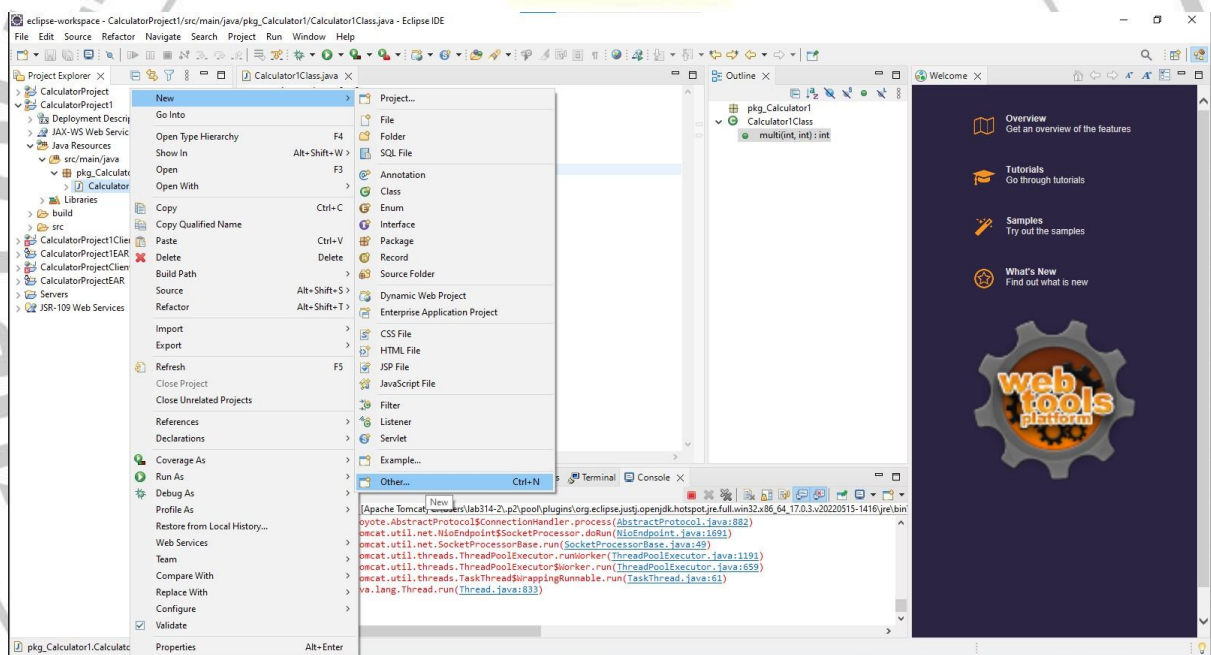
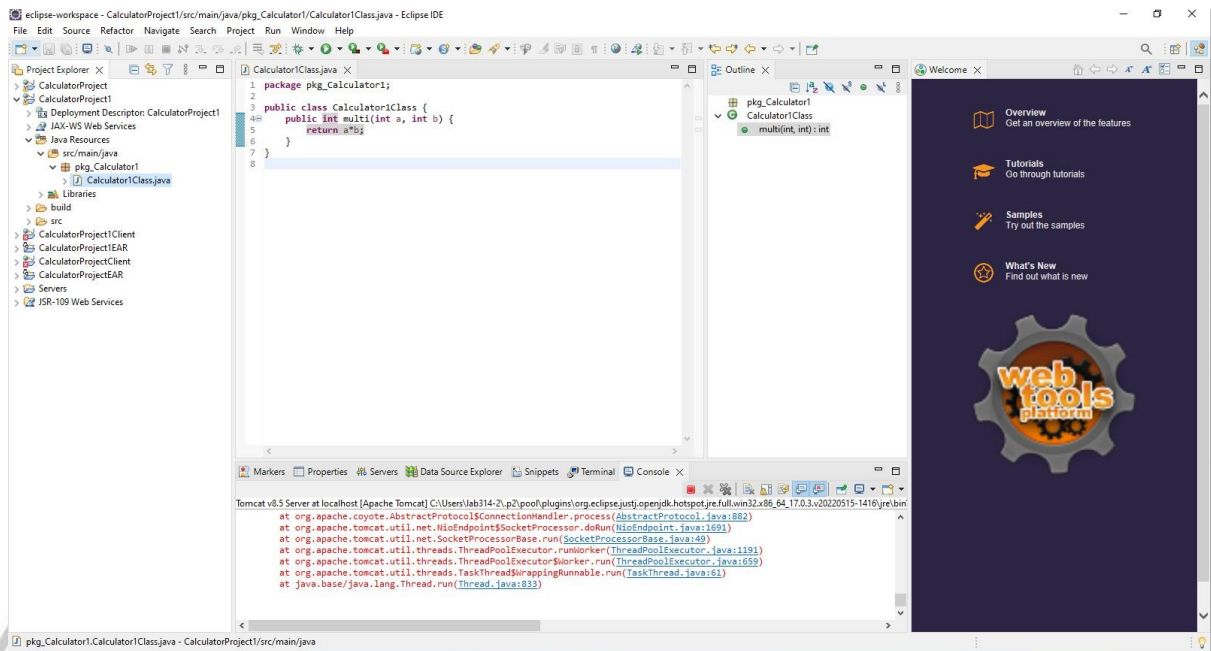
- Web Service:** Select a test facility.
- Test facility:** Web Services Explorer.
- Launch:** Button to execute the test.

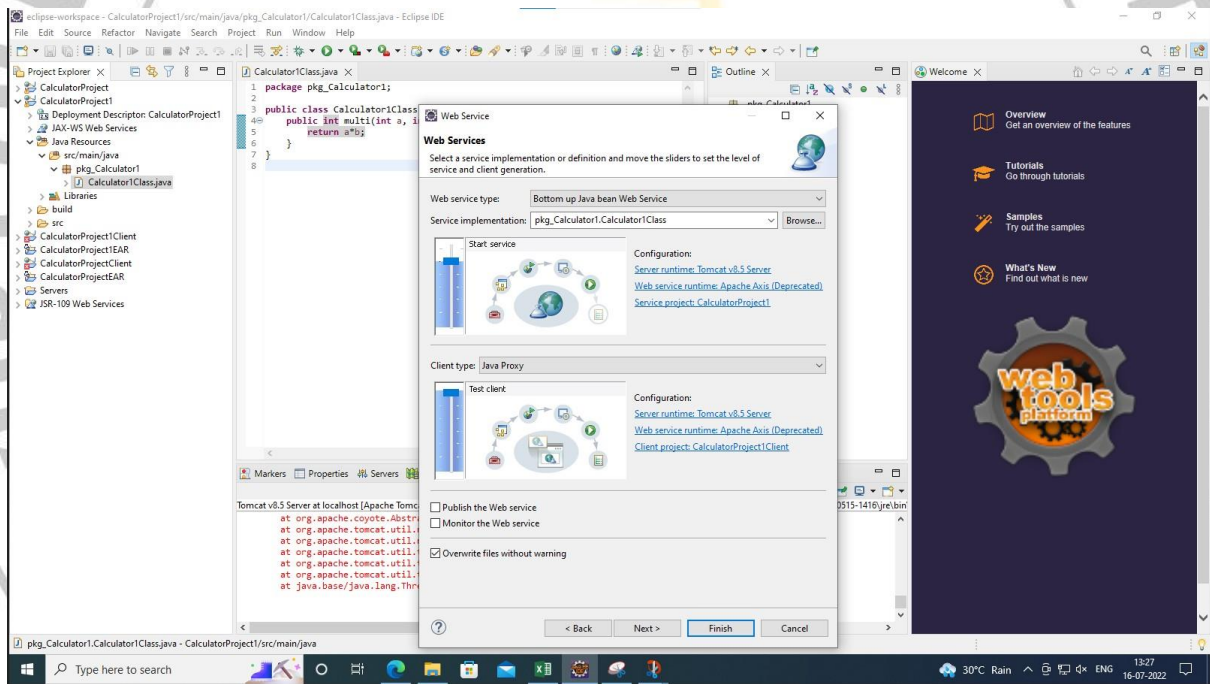
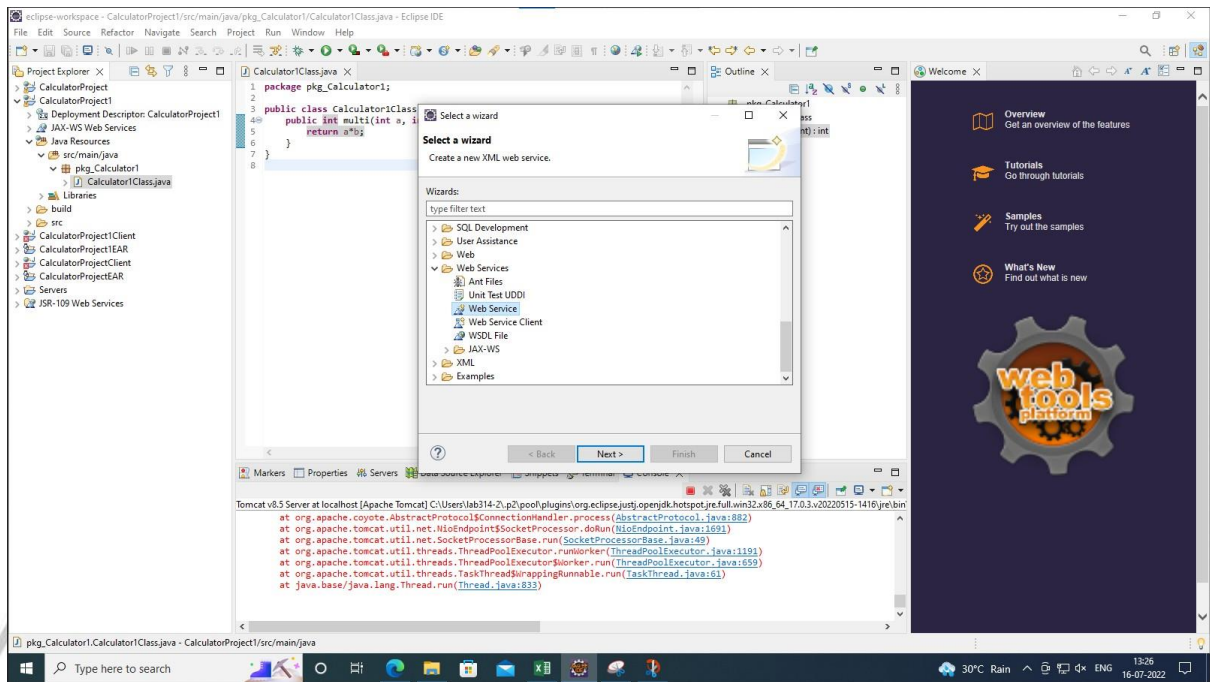
Tomcat v8.5 Server at localhost [Apache Tomcat]:

```
Jul 16, 2022 1:22:34 PM org.apache
INFO: Reloading Context with name
Jul 16, 2022 1:22:35 PM org.apache
INFO: At least one JAR was scanned
Jul 16, 2022 1:22:35 PM org.apache
WARNING: Unable to find required c
Jul 16, 2022 1:22:35 PM org.apache
INFO: Reloading Context with name
```

web tools platform:

- Overview:** Get an overview of the features.
- Tutorials:** Go through tutorials.
- Samples:** Try out the samples.
- What's New:** Find out what is new.





Result and Discussion:

Learning Outcomes: Students should have the ability to

LO1: Define Web services.

LO2: Identify different phases of web services.

LO3: Explain web service protocol.

Course Outcomes: Upon completion of the course students will be able to know about web services and its implementation.

Conclusion:

Viva Questions:

1. Define web services.
2. Explain artifacts of web services.
3. Explain different phases in web service Implementation.
4. Explain Service Transport in web services.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	

Experiment No.: 4

Integrate Software Components using middleware

Learning Objective: Student should be able to integrate software components using middleware

Tool:- Java RMI

Software system integration is essential where communication between different applications running on different platform is needed. Suppose a system designed for payroll running with Human Resource System. In that case employees' data need to be inserted in both systems. The system integration benefits a lot in these cases where data and services needed to be shared.

Web services are becoming very popular to share data between systems over the network and over the internet as well. In software industry the software integration carried same steps as software development and hence demands same kind of development procedures and testing.

This ensures the meaningful and clear communication between the systems. Systems integration becomes inevitable in Enterprise Systems where the whole organization needed to share data and services and give the feel to user as one system. The core purpose of integration is to make the systems communicate and also to make the whole system flexible and expandable.

The integration of different softwares written in different language and based on different platforms can be tricky. In that situation a middleware is necessary to enable the communication between different softwares. The middleware enables the software system not only to share data but also share the services

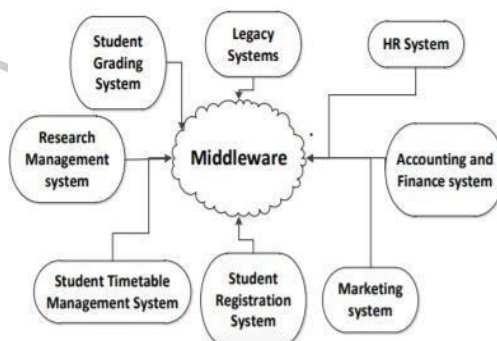


Figure 1: Middleware

A . Middleware

Independently written software systems need to be integrated in large system for example industry, institution, etc. These systems need to agree on common method for integration. To get them communicate there is need to have something in between them. The middle thing is termed commonly as middleware.

B .Service Oriented Architecture

Service Oriented Architecture (SOA) is the architectural design and pattern which is used to provide services to different applications. Its goal is to achieve loose coupling between interacting components of applications. Web Services, Corba, Jini, etc are the technologies used to implement SOA.

Main benefit of SOA is that it can provide the means of communication between completely different applications (built in different technologies). The services are also completely independent and reusable and the nature of reusability provide the less time to market. All the services technologies needs to be implemented using the SOA design pattern and need to be designed on the basis of SOA to get maximum benefit.

C .Web Services

Web service is the SOA technology with additional requirements of using internet protocols (HTTP, FTP, SMTP, etc.) and using XML (Extensible Markup Language) for message transmission. [7] These are application components which communicate between different applications using open protocol. Open protocol is the web protocol for querying and updating information. These components can be used by different kinds of applications to exchange information. HTML (HyperText Markup Language) and XML are the basics of web service implementation.

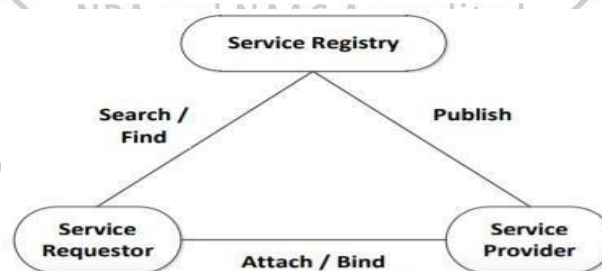


Figure 2: Web-Service Architecture

D. WSDL (Web Service Descriptive Language)

WSDL is a language to describe web services and providing the link to access these web services. It is acting as a publisher in web service architecture. It is the XML document which is recommended by W3C (World Wide Web Consortium) in 2007. In WSDL XML describes the service and

the address from where applications can access the service. [8] WSDL predecessors were COM and CORBA

E. UDDI (Universal Description, Discovery and Integration)

It is directory service / registry service where applications can register and look for web services. It is Platform independent framework. It is acting like service register in web service architecture. It uses HTML, XML and DNS (Domain Name Server) protocols which enables it to become the directory service. It defines the keyword search, categories and classification for an application and registers it into business directory. In that way it is making the application easier to be approached by the customers online

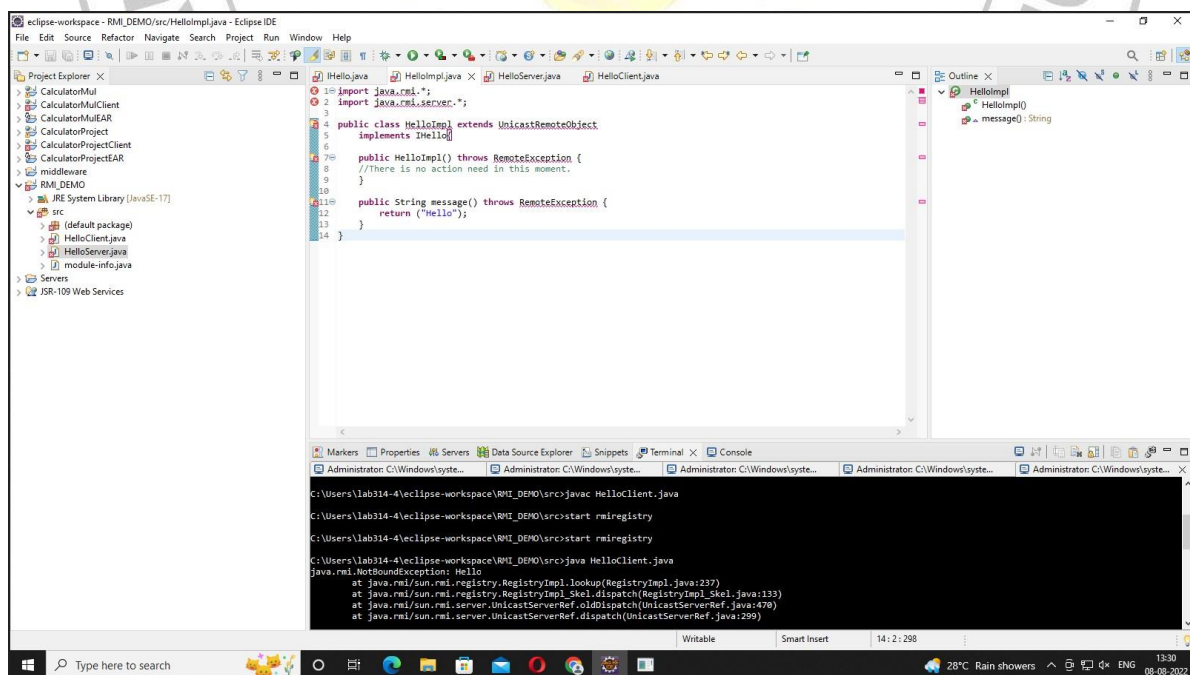
F. SOAP (Simple Object Access Protocol)

It is a messaging / invoker in web service architecture. It is used to send messages in between the service and service consumer; and in between service consumer and service registry. It is used to communicate with web service. It is a message framework which transfers the information between sender and receiver. SOAP doesn't define the service but defines the mechanisms for messaging. It binds the client to the service

G. SOA - A solution to spaghetti architecture

In a fairly medium scale to large software architecture where there is need to integrate or communicate between different kinds of applications the introduction of links can make the architecture messy and it is called spaghetti architecture [9]. Figure 4 shows that problem in detail. It makes the system less flexible and expandability is the nightmare. Service Oriented Architecture (SOA) makes the architecture flexible and expandable.

OUTPUT:



The screenshot shows the Eclipse IDE with a project named 'RMI_DEMO'. The 'src' folder contains three files: 'HelloImpl.java', 'HelloServer.java', and 'HelloClient.java'. The 'HelloImpl.java' file is open, showing the following code:

```

1 import java.rmi.*;
2 import java.rmi.server.*;
3
4 public class HelloImpl extends UnicastRemoteObject
5 implements IHello {
6
7     public HelloImpl() throws RemoteException {
8         //There is no action need in this moment.
9     }
10
11     public String message() throws RemoteException {
12         return ("Hello");
13     }
14 }
  
```

The console output shows the execution of the RMI demo:

```

C:\Users\lab314-4\workspace\RMI_DEMO\src>javac HelloClient.java
C:\Users\lab314-4\workspace\RMI_DEMO\src>start rmiRegistry
C:\Users\lab314-4\workspace\RMI_DEMO\src>java HelloClient.java
java.rmi.NotBoundException: Hello
    at java.rmi.sun.rmi.registry.RegistryImpl.lookup(RegistryImpl.java:237)
    at java.rmi.sun.rmi.registry.RegistryImpl.dispatch(RegistryImpl.java:133)
    at java.rmi.sun.rmi.server.UnicastServerRef.dispatch(UnicastServerRef.java:470)
    at java.rmi.sun.rmi.server.UnicastServerRef.dispatch(UnicastServerRef.java:299)
  
```

eclipse-workspace - RMI_DEMO/src/HelloServer.java - Eclipse IDE

```

1 import java.rmi.*;
2
3 public class HelloServer {
4     private static final String host = "localhost";
5     public static void main(String[] args) throws Exception {
6         /** Step 1
7         /** Declare a reference for the object that will be implemented
8         HelloImpl temp = new HelloImpl();
9         /** Step 2
10        /** Declare a string variable for holding the URL of the object's name
11        String rmiObjectName = "rmi://" + host + "/" + "Hello";
12        /** Step 3
13        /** Binding the object reference to the object name.
14        Naming.rebind(rmiObjectName, temp);
15        /** Step 4
16        /** Tell to the user that the process is completed.
17        System.out.println("Binding complete...\n");
18    }
19 }
  
```

Console:

```

C:\Users\lab314-4\workspace\RMI_DEMO\src>javac HelloClient.java
C:\Users\lab314-4\workspace\RMI_DEMO\src>start rmiregistry
C:\Users\lab314-4\workspace\RMI_DEMO\src>start rmiregistry
C:\Users\lab314-4\workspace\RMI_DEMO\src>java HelloClient.java
java.rmi.NotBoundException: Hello
    at java.rmi.sun.rmi.registry.RegistryImpl.lookup(RegistryImpl.java:237)
    at java.rmi.sun.rmi.registry.RegistryImpl_skel.dispatch(RegistryImpl_skel.java:133)
    at java.rmi.sun.rmi.server.UnicastServerRef.oldDispatch(UnicastServerRef.java:470)
    at java.rmi.sun.rmi.server.UnicastServerRef.dispatch(UnicastServerRef.java:299)
  
```

eclipse-workspace - RMI_DEMO/src/HelloClient.java - Eclipse IDE

```

1 import java.rmi.*;
2 import java.rmi.Naming;
3
4 public class HelloClient
5 {
6     private static final String host = "localhost";
7     public static void main(String[] args)
8     {
9         try
10        {
11            /** Obtain a reference to the object from the registry and next,
12            /** it will be typecasted into the most appropriate type.
13            Hello greeting_message = (Hello) Naming.lookup("rmi://" + host + "/" + "Hello");
14            /** Next, we will use the above reference to invoke the remote
15            /** object method.
16            System.out.println("Message received: " +
17            greeting_message.getMessage());
18        }
19        catch (ConnectException conEx)
20        {
21            System.out.println("Unable to connect to server!");
22            System.exit(1);
23        }
24        catch (Exception ex)
25        {
26            ex.printStackTrace();
27            System.exit(2);
28        }
29    }
30 }
  
```

Console:

```

C:\Users\lab314-4\workspace\RMI_DEMO\src>javac HelloClient.java
C:\Users\lab314-4\workspace\RMI_DEMO\src>start rmiregistry
C:\Users\lab314-4\workspace\RMI_DEMO\src>start rmiregistry
C:\Users\lab314-4\workspace\RMI_DEMO\src>java HelloClient.java
java.rmi.NotBoundException: Hello
    at java.rmi.sun.rmi.registry.RegistryImpl.lookup(RegistryImpl.java:237)
    at java.rmi.sun.rmi.registry.RegistryImpl_skel.dispatch(RegistryImpl_skel.java:133)
    at java.rmi.sun.rmi.server.UnicastServerRef.oldDispatch(UnicastServerRef.java:470)
    at java.rmi.sun.rmi.server.UnicastServerRef.dispatch(UnicastServerRef.java:299)
  
```

Result and Discussion:

Learning Outcomes: Students should have be able to

LO1: Define Middleware.

LO2: Identify different components in middleware.

LO3: Explain Software Components using middleware.

Course Outcomes: Upon completion of the course students will be able to understand middleware and its components.

Conclusion:

Viva Questions:

1. Define Middleware.
2. Explain Service Oriented Architecture.
3. Explain Web Services.
4. Explain Universal Description, Discovery and Integration.

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	

Experiment No.: 5

Use middleware to implement connectors

Learning Objective: Student should be able to understand use of middleware to implement connectors.

Theory:

What is Middleware ?

Middleware is a more effective program that acts as bridge in between various applications and other databases otherwise tools. It is placed in between operating system and other applications which run on it. Middleware allows making better communication, application services, messaging, authentication, API management and management of data between different kinds of applications which help to exchange data.

The connectors sit between the two APIs or you can say and the ends of the connectors are APIs. The connectors receive data from one app/solution and process it to make it understandable and accessible in the other app/solution, regardless of whether any direct form of integration was available in the two apps.

Role of Middleware is :-

Middleware is a potentially useful tool when building software connectors. First, it can be used to bridge thread, process and network boundaries. Second, it can provide pre-built protocols for exchanging data among software components or connectors. Finally, some middleware packages include features of software connectors such as filtering, routing, and broadcast of messages or other data.

A signal interaction is a one-way interaction between an initiating object, called a client, and a responding object, called a server. An operation interaction is an interaction between a client object and server object that is either an interrogation or an announcement. An interrogation is composed of two one-way interactions: a request and a response. An announcement is a one-way request from a client object to a server object in which the client object expects no response, and the server object does not respond. A flow interaction is an ordered set of one or more one-way communications from a producer object to a consumer object. These interactions are a generalized

metamodel for describing communication styles between objects that can be implemented using a variety of middleware such as Remote Procedure Call (RPC) and Remote Method Invocation (RMI) and message queues (as selected and specified in the technology view).

can be implemented using a variety of middleware such as Remote Procedure Call (RPC) and Remote Method Invocation (RMI) and message queues.

Connectors as a primary vehicle for interprocess communication. A single conceptual connector can be “broken up” vertically (a) or horizontally (b) for this purpose.

Vertical Connectors :

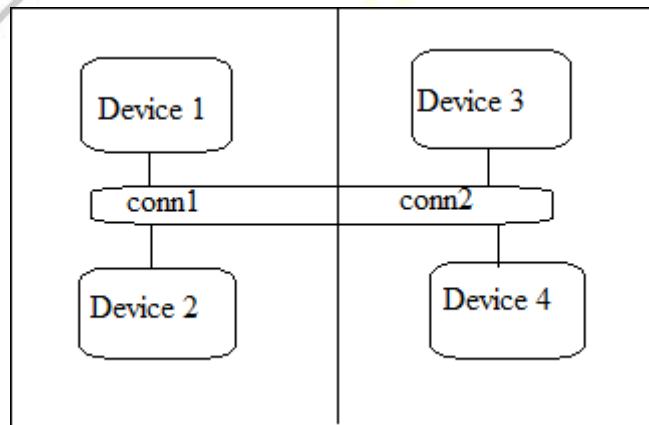


Figure (a)

Horizontal Connectors :

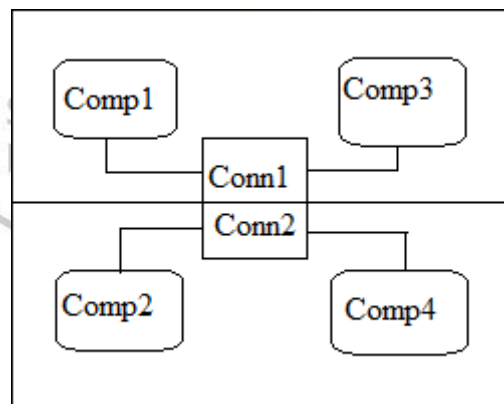


Figure (b)

Linking Ports Across Process Boundaries :

The ports can call methods on each other, sending messages as method parameters. Our intent was to simply use the middleware to exchange port references across process boundaries and use the existing technique for message passing.

The middleware technology would be entirely encapsulated within the port entity and would not be visible to architects or developers. The single process implementation of a C2 connector links two ports together by having each port contain a reference to the other one.

Linking Connectors Across Process Boundaries :

Sharing communication ports across process boundaries gave us fine-grained control over implementing an architecture as a multi-process application. However, it required additional functionality in the C2 implementation framework and did not isolate the change to the appropriate abstraction: the connector. In order to remedy this, we devised two connector-based approaches. Both of these approaches consist of implementing a single conceptual software connector using two or more actual connectors that are linked across process or network boundaries. Each actual connector thus becomes a segment of a single “virtual connector.” All access to the underlying middleware technology is encapsulated entirely within the abstraction of a connector, meaning that it is unseen by both architects and developers. We call the first approach “lateral welding,” depicted in Fig. 2a. Messages sent to any segment of the multi-process connector are broadcast to all other segments. Upon receiving a message, each segment has the responsibility of filtering and forwarding it to components in its process as appropriate. Only messages are sent across process boundaries. While the lateral welding approach allowed us to “vertically slice” a C2 application, we also developed an approach to “horizontally slice” an application, as shown in Fig. 2b. This approach is similar to the idea of lateral welding: a conceptual connector is broken up into top and bottom segments, each of which exhibits the same properties as a single-process connector to the components attached above and below it, respectively. However, the segments themselves are joined using the appropriate middleware. When used with a middleware technology that supports dynamic change at run-time, all of these approaches, both using ports and connectors, can be used to build applications where processes can join and leave a running application.

Using Middleware Technologies :

To explore the use of OTS middleware with software connectors, we chose four representative technologies from the field of available middleware packages. These were Q, an RPC system, Polyolith, a message bus, RMI, a connection mechanism for Java objects, and ILU, a distributed objects package. A description of one of our efforts involving integrating two middleware technologies simultaneously in the same application is given here. With each middleware package, we were able to encapsulate all the middleware functionality within the connectors

themselves. This means that architects and developers can use the middleware-enhanced connectors thus created just as they would use normal, in-process C2 connectors.

Simultaneous Use of Multiple Middleware Packages :

Each middleware technology we evaluated has unique benefits. By combining multiple such technologies in a single application, the application can potentially obtain the benefits of all of them. For instance, a middleware technology that supports multiple platforms but only a single language, such as RMI, could be combined with one that supports multiple languages but a single platform, such as Q, to create an application that supports both multiple languages and multiple platforms. The advantages of combining multiple middleware technologies within software connectors are manifold. In the absence of a single panacea solution that supports all required platforms, languages, and network protocols, the ability to leverage the capabilities of several different middleware technologies significantly widens the range of applications that can be implemented within an architectural style such as C2. We combined our implementations of ILU-C2 and RMI-C2 connectors in a version of the KLAX application, a real time video game application built as an experimental platform for work on the C2 architecture. We were able to do so with no modification to the middleware-enhanced C2 framework or the connectors themselves by combining the lateral welding technique shown in Fig. 2a with the horizontal slicing technique shown in Fig. 2b. This approach works for any combination of OTS connectors that use the lateral welding technique. An alternative approach would have been to create a single connector that supported both ILU and RMI, but this would have required changes to the framework.

OUTPUTS:

Estd. 2001

ISO 9001 : 2015 Certified
NBA and NAAC Accredited

eclipse-workspace - Middleware/src/ClientClass.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

```

1 public class ClientClass {
2
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         InterfaceClass c = new ServerClass();
7         System.out.println(c.sub(11,9));
8     }
9
10
11
12
13 }

```

Outline: ClientClass, main(String[]) : void

Console: <terminated> ClientClass [Java Application] C:\Users\lab314-2\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\java

eclipse-workspace - Middleware/src/ServerClass.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- CalculatorProject
- CalculatorProject1
- CalculatorProject1Client
- CalculatorProject1EAR
- CalculatorProjectClient
- CalculatorProjectEAR
- Middleware
 - JRE System Library [JavaSE-17]
 - src
 - (default package)
 - ClientClass.java
 - InterfaceClass.java
 - ServerClass.java
- Servers
- JSR-109 Web Services

Outline

- ServerClass
 - add(int, int) : int
 - sub(int, int) : int
 - mul(int, int) : int
 - div(int, int) : float

```

1  public class ServerClass implements InterfaceClass {
2
3
4  //public static void main(String[] args) {
5      // TODO Auto-generated method stub
6
7  //}
8  public int add(int a, int b) {
9      return a+b;
10 }
11 public int sub(int a, int b) {
12     return a - b;
13 }
14
15 public int mul(int a, int b) {
16     return a * b;
17 }
18
19 public float div(int a, int b) {
20     return a / b;
21 }
22 }
23
24

```

Markers Properties Servers Data Source Explorer Snippets Terminal Console

<terminated> ClientClass [Java Application] C:\Users\lab314-2\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_b4_1\j3.v20220513-1416\jre\bin\java.exe

Writable Smart Insert 4 : 7 : 6

eclipse-workspace - Middleware/src/ClientClass.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- CalculatorProject
- CalculatorProject1
- CalculatorProject1Client
- CalculatorProject1EAR
- CalculatorProjectClient
- CalculatorProjectEAR
- Middleware
 - JRE System Library [JavaSE-17]
 - src
 - (default package)
 - ClientClass.java
 - InterfaceClass.java
 - ServerClass.java
- Servers
- JSR-109 Web Services

ClientClass.java

```
1 public class ClientClass {
2
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         InterfaceClass c = new ServerClass();
7         System.out.println(c.add(11,9));
8     }
9
10
11
12
13 }
```

Outline

- ClientClass
 - main(String[]): void

Markers Properties Servers Data Source Explorer Snippets Terminal Console

<terminated> ClientClass [Java Application] C:\Users\lab314-2\p2\poo\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220513-1416\jre\bin\java20

Result and Discussion:

Learning Outcomes: Students should have be able to

LO1: Define Middleware.

LO2: Identify different connectors in middleware.

LO3: Explain middleware implements in connectors.

Course Outcomes: Upon completion of the course students will be able to understand middleware and its connectors.

Conclusion:

Viva Questions:

1. Define Middleware.
2. Explain use of middleware.
3. Explain implementation of connectors.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	

Experiment No.: 7

Identifying Design requirements for an Architecture for any specific domain.

Learning Objective: Student should be able to understand Design requirements for an Architecture for any specific domain.

Theory:

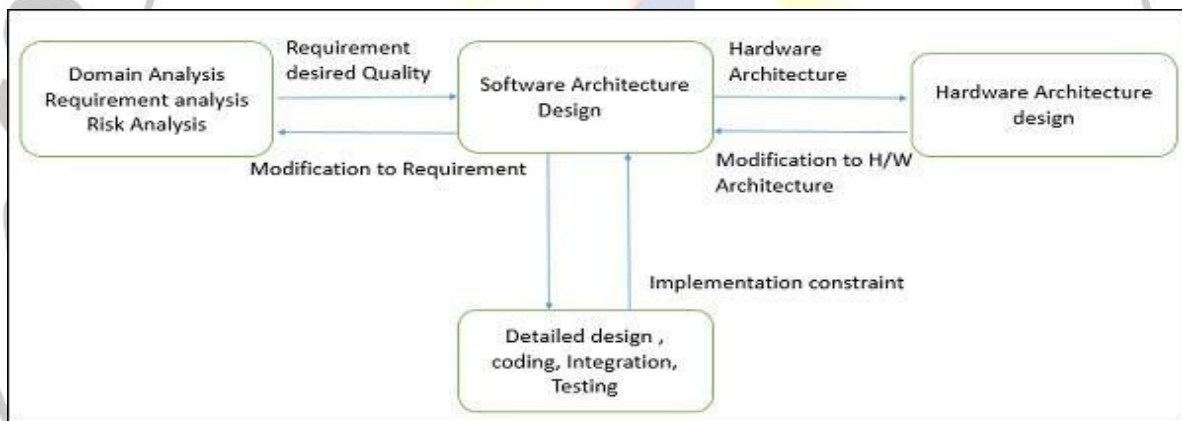
Software design:

Software design provides a design plan that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows –

To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.

Act as a blueprint during the development process.

Guide the implementation tasks, including detailed design, coding, integration, and testing.



It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.

Software design is responsible for the code level design such as, what each module is doing, the classes scope, and the functions purposes, etc. When used strategically, they can make a programmer significantly more efficient by allowing them to avoid reinventing the wheel, instead using methods refined by others already. They also provide a useful common language to conceptualize repeated problems and solutions when discussing with others or managing code in larger teams

Major artifacts of the software design process include:

- **Software requirements specification.** This document describes the expected behavior of the system in the form of functional and non-functional requirements. These requirements should be clear, actionable, measurable, and traceable to business requirements. Requirements should also define how the software should interact with humans, hardware, and other systems.
- **High-level design.** The high-level design breaks the system's architectural design into a less-abstracted view of sub-systems and modules and depicts their interaction with each other. This high-level design perspective focuses on how the system, along with all its components, implements in the form of modules. It recognizes the modular structure of each sub-system and their interaction among one another.
- **Detailed design.** Detailed design involves the implementation of what is visible as a system and its sub-systems in a high-level design. This activity is more detailed towards modules and their implementations. It defines a logical structure of each module and their interfaces to communicate with other modules.

The purpose of an architecture schema or design record is to serve as a vehicle for software understanding by functioning as a collection point for knowledge about the components that make up a DSSA. In particular, the design record organizes

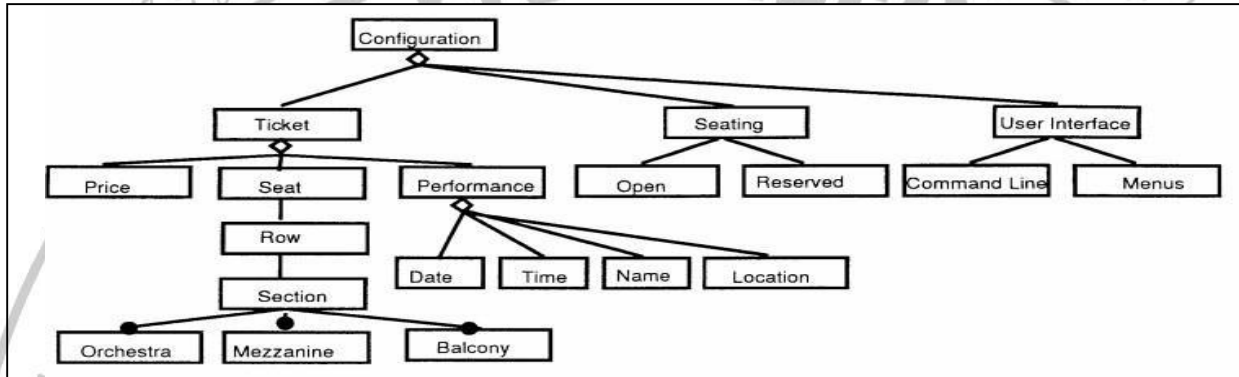
- domain- specific knowledge about components or design alternatives and
- Implementation in knowledge about alternate implementations,

The primary goal of a design record is to adequately describe the components in a reference architecture such that the application engineer can make design decisions and component selections without looking at implementations. The secondary goal of a design record is to provide information that the tools in the supporting environment can use.

The design record data elements used by Loral Federal Systems phases in the software life cycle, include:

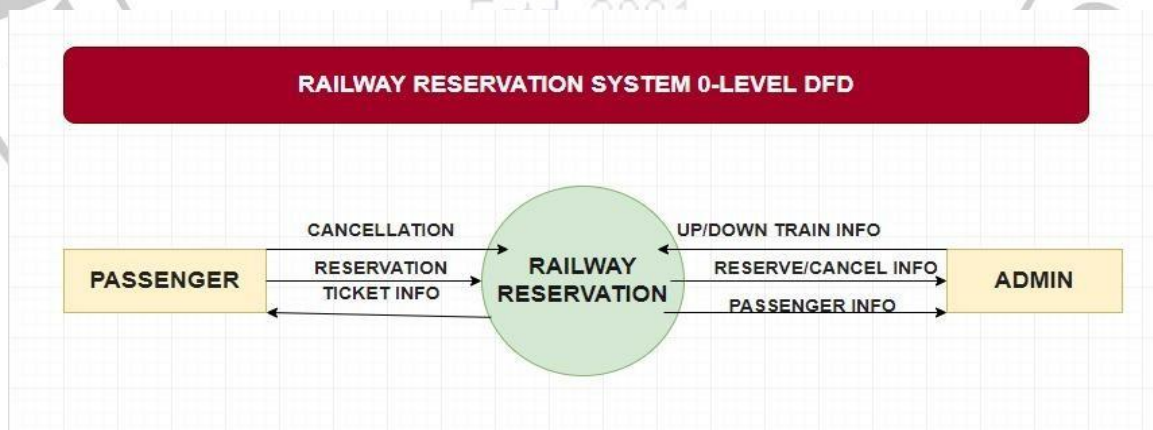
1. Name/type
2. Description
3. reference requirements satisfied,
4. design structure (data flow and control flow diagrams),
5. design rationale,
6. interface and architecture specifications and dependencies,
7. P D L (program Design Language) text,
8. implementation,
9. configuration and version data, and

10. test cases.
11. metric data,
12. access rights,
13. search points,
14. catalog information,
15. library and DSSA links, and hypertext paths

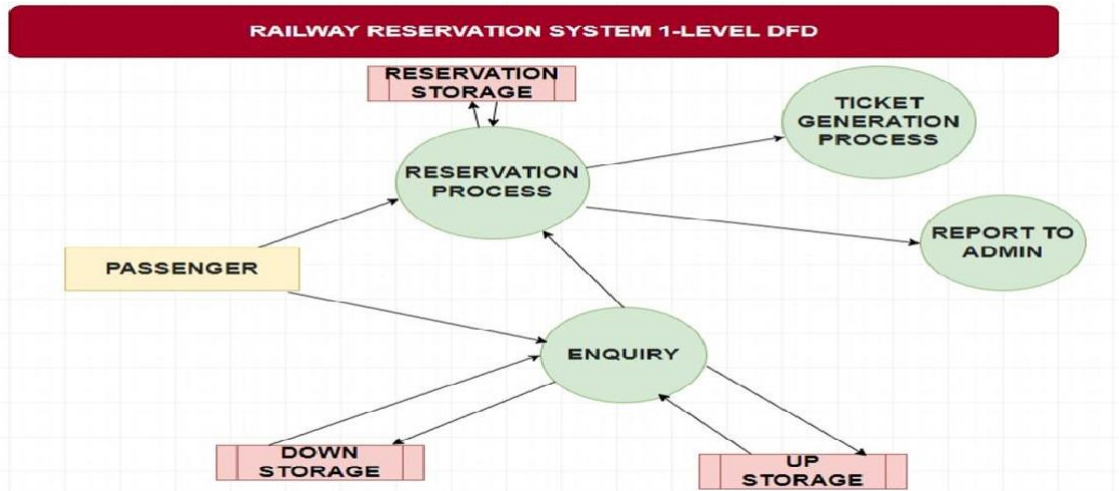


Many different programming formats incorporate the same essential elements. In all cases, the design programming fits within a larger context of planning efforts which can also be programmed. For design programming for a building, we propose a six-step process as follows:

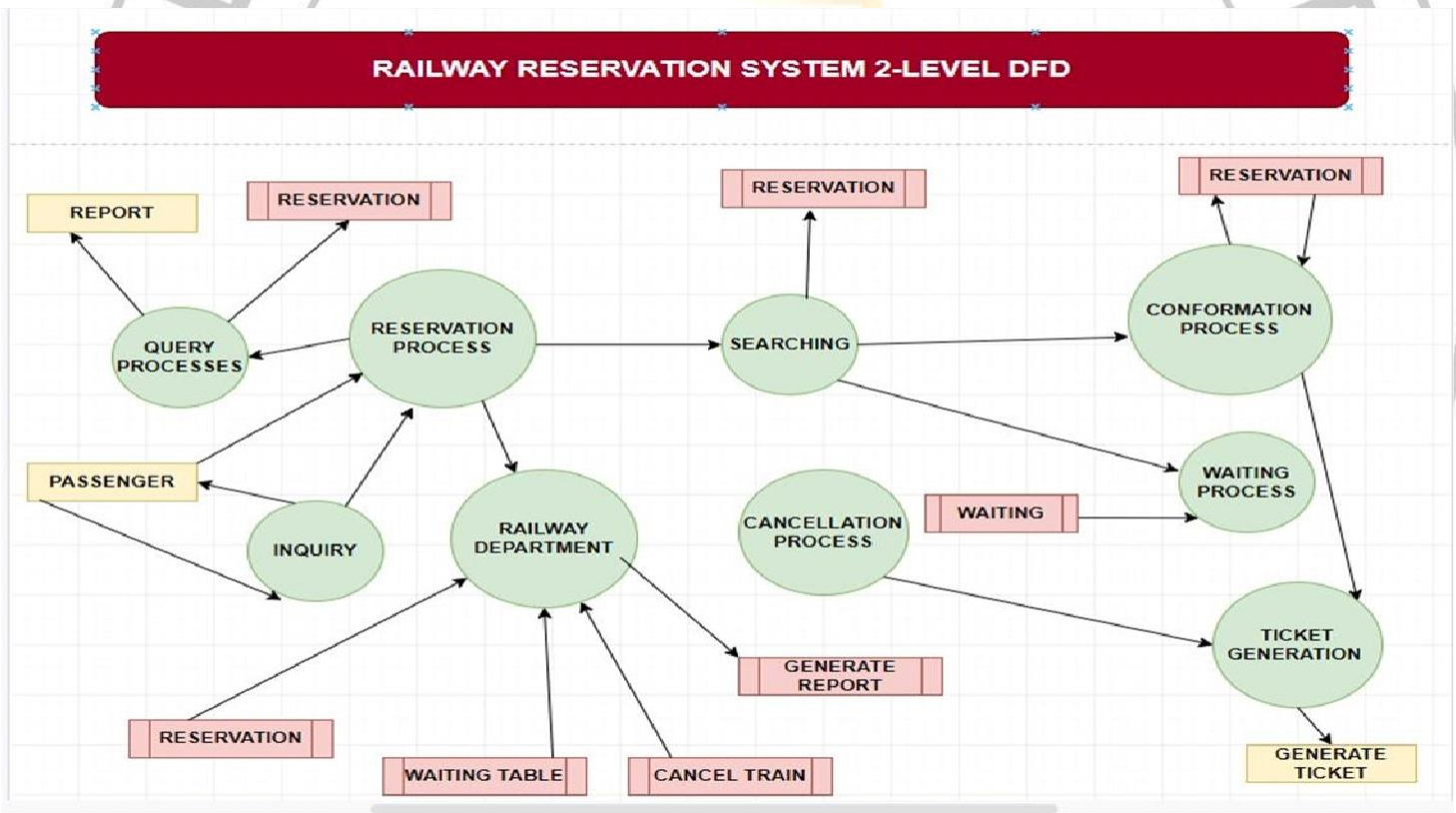
1. Research the project type
2. Establish goals and objectives
3. Gather relevant information
4. Identify strategies
5. Determine quantitative requirements
6. Summarize the program



Context Diagram / Level – 0 DFD



1st Level



2nd Level

Result and Discussion:

Learning Outcomes: Students should have been able to understand

LO1: Define software design.

LO2: Identify different design requirements for software.

LO3: Explain implementation of design requirements of software.

Course Outcomes: Upon completion of the course students will be able to understand design requirement of software Architecture.

Conclusion:

Viva Questions:

1. Define design requirement in software.
2. Explain different design requirements in software Architecture.
3. Explain the phases of programming design.
4. Explain any three design requirements.

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	

Experiment 9

Mapping of non-functional components with system requirements.

Learning Objective: Student should be able to understand Mapping of non-functional components with system requirements.

Theory:

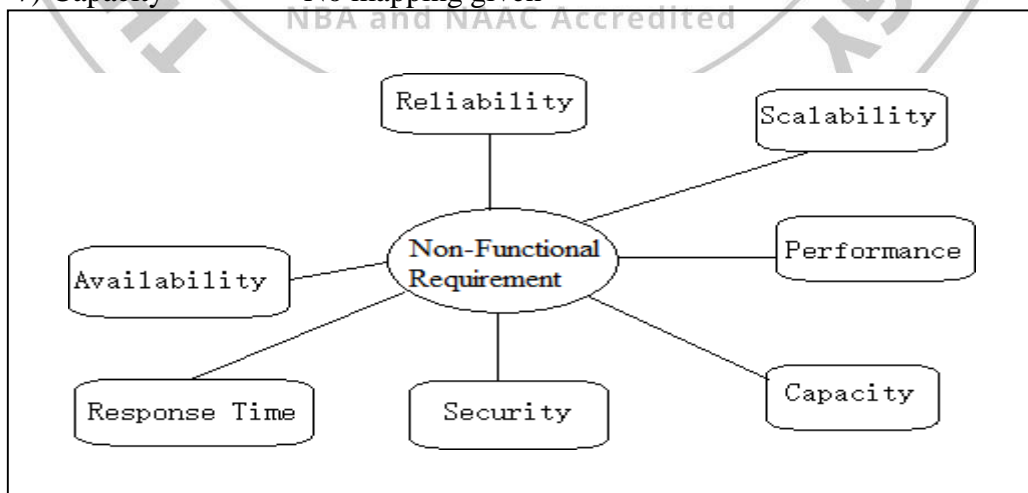
What is Non-Functional Requirement?

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system.

NFRs define the system properties and specify the behavioral pattern under various operating conditions. The various estimation methods help in sizing the application based on the functional requirements. However, most of these methods have overlooked the influence of non-functional requirements.

The key NFRs that can be attributed to an application and their mapping as follows.

- | | |
|------------------|---|
| 1) Reliability | Operation Ease |
| 2) Response Time | No mapping given |
| 3) Performance | Performance, Online Update, Online Date Entry |
| 4) Security | No mapping given |
| 5) Availability | No mapping given |
| 6) Scalability | Transaction rate |
| 7) Capacity | No mapping given |



Middleware is a more effective program that acts as bridge in between various applications and other databases otherwise tools. It is placed in between operating system and other applications which run on it. Middleware allows making better communication, application services, messaging, authentication, API management and management of data between different kinds of applications which help to exchange data.

The connectors sit between the two APIs or you can say the ends of the connectors are APIs. The connectors receive data from one app/solution and process it to make it understandable and accessible in the other app/solution, regardless of whether any direct form of integration was available in the two apps.

Mapping of NFRs:

Operation Ease to Reliability:

An application or the software system once installed and configured on a given platform should require no manual intervention, except for starting and shutting down. The system should be able to maintain a specified level of performance in case of software faults. It should also be able to re-establish its level of performance and to recover all the data directly affected in case of a failure in the minimum time and effort. This is mapped on to the reliability NFR. It may be defined as “a system which is capable of reestablishing its level of performance and recovering the data directly affected in case of a failure and on the time and effort needed for it. The design criteria for reliability can be defined as self-contained the system should have all the features necessary for all its operations including recovering it by itself; completeness- it should be complete in itself and not dependent on anything else; robustness/integrity- it should not easily breakdown; error tolerance- it should be able to tolerate errors and rectify them and continue in its operation. There are “numerous metrics for determining reliability: mean time to failure, defect reports and counts, resource consumption, stability, uptime percentage and even customer perception.”

Performance:

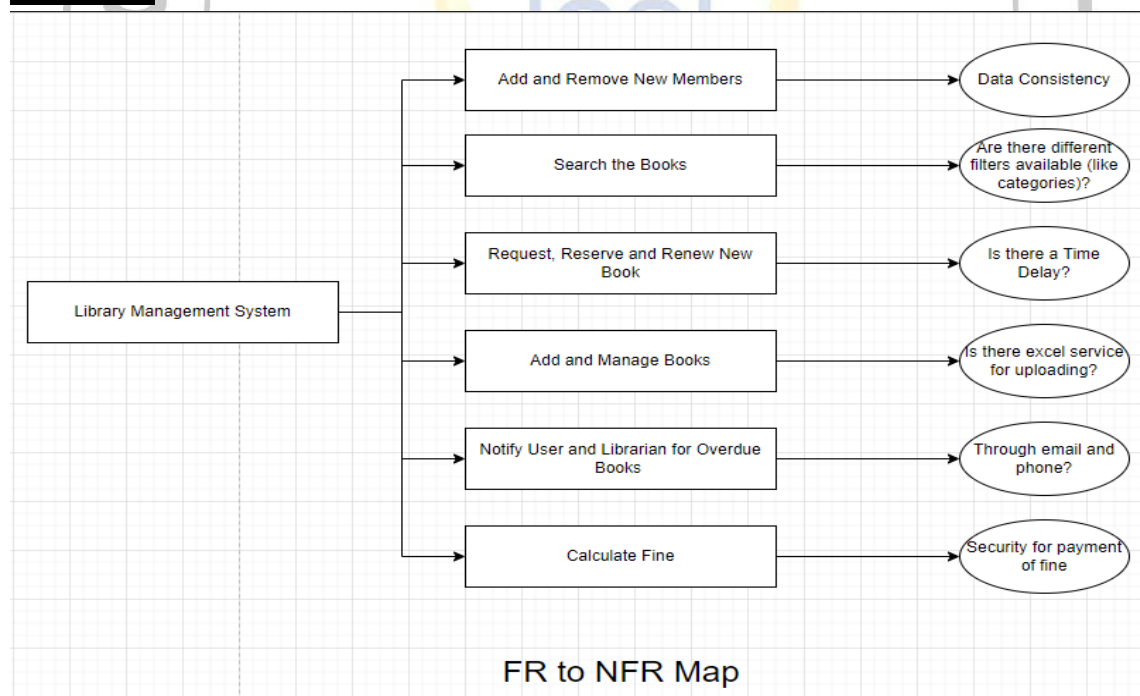
Real time systems have strict performance parameters like performing at the same level even during peak user times, producing high throughput, serving a huge user base, etc. The DI varies from no special performance requirements to response time being critical during all business hours and till performance analysis tools being used in the design. System should meet the desired performance expectation. Also, if online update has to take place, then the performance expectations to be met are very high – fast response, low processing time and high throughput rates. The performance NFR is also based on the Online Data Entry requirements of an application. The present-day trend is to have interactive and real-time data entry. The GUI development

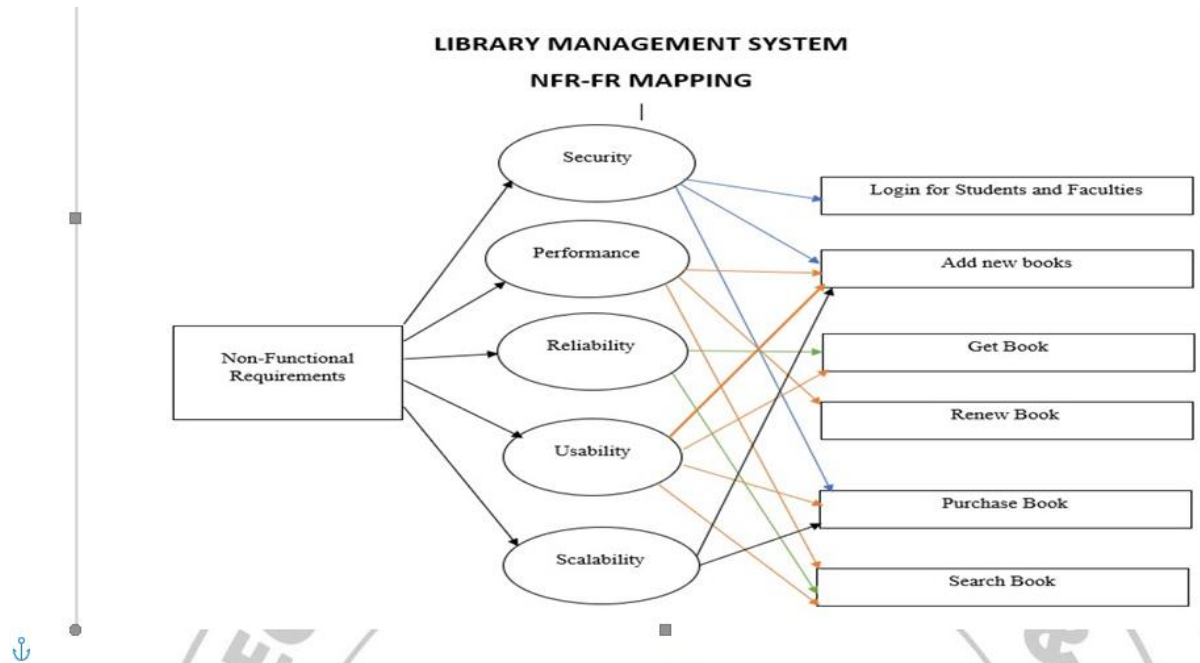
requires a lot of effort as help has to be provided, validation to be implemented, reference information for faster data entry operations, etc. Performance when related to this can be defined as “attributes of software that bear on response and processing times and on throughput rates in performing its function.

Transaction Rate to Scalability:

In many business applications the transaction rate increases to high peak levels once in a day or once in a week with the requirement remaining so that there has to be no dramatic increase in transaction time. This issue has to be looked into in the design, development and/or installation phases of a project. This GSC is mapped on to the scalability NFR. The term scalability implies “the ability to scale up to peak transaction loads. In order to achieve this the application has to be designed in such a way so that it should cater to the highest possible figures thus wasting resources when the transaction rate is low. The architecture should be designed in a multi-layered manner in complex algorithm-based applications to scale up to peak transaction rates. In today’s systems, this GSC does not contribute much to the DI as present-day hardware and operating systems provide built-in features such as high bandwidth network, high speed storage disks with high-speed disk access timings and CPUs with high MHZ processing speed which when combined leads to build in high transaction rates.

OUTPUT:





Result and Discussion:

Learning Outcomes: Students should have been able to

LO1: Define nonfunctional Requirement.

LO2: Identify different component of nonfunctional Requirement.

LO3: Explain mapping.

Course Outcomes: Upon completion of the course students will be able to understand mapping of non-functional components with system requirements.

Conclusion:

Viva Questions:

1. Define nonfunctional Requirement.
2. Explain nonfunctional Requirement component.
3. Explain Transaction Rate to Scalability mapping of NFR.

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	