EXPERIMENT NO. 01: MODELING USING XADL 2.0

AIM: Student should be able to implement Modeling using xADL 2.0

THEORY:

Software architecture can help people to better understand the total structure of the system (Software). In recent years software size and complexity has enormously increasing. This made the description of their architecture more complicate. So to describe these architectures many methods are invented. These are called as Architecture Descriptive languages. But many of these are not succeeded. Since these are devised for a particular style of architecture. This made the invention of an eXtensible Architecture Description language. Research and experimentation in the field of software architecture yielded invention of many ADLs (Architecture Description Languages). xADL 2.0 (pronounced as "zay-dul") is one ADL which is designed to suit that type of problems. This can be used to describe the architecture of the various types of systems. It is developed in that way to adapt the changes in the architecture styles.

Architecture Description Language

Architectural Description language or architectural definition language (or an ADL) is an abstract level description of the software system using the components, connectors, links etc., in a formal language or notations. Actually there is no concrete definition for the ADL so far now. An ADL will provide its users concrete syntax, formal semantics, conceptual framework. These are helpful for explicitly modeling software systems. All these ADLs should have some properties they are components, connectors, interfaces, links and interconnection between the components.

Introduction to xADL

The core of the xADL consists of common modules to describe components, connectors, interfaces, links (configurations). These modules are developed as XML schemas. The idea behind using XML is, it is very good for interchange of structures data exchange, extendible and it is a modular language. To maximize the reusability of these schemas these are made as generic as possible. That is one instance is written to represent components and connectors, but their behavior and communication are represented in another schema. Thus, aspects of elements like behaviors

and constraints on how elements may be arranged are not specified. Such aspects are meant to be defined in extension schemas. The important features of the ADL which are extended from the xADL2.0 are:

1. Separation between design time schemas and runtime schemas.
2. Implementation mappings that map the ADL specifications into code.
3. The ability to model the architecture of product line architectures and configuration management systems.

Advantages of xADL over other ADLs

Actually xADL takes its ides from the ACME and on the survey about the ADLs about the commonalities in the ADLs. The advantages of the xADL over the other ADLs are

- It can be used to represent many architecture domains.
- If any new Architecture domain comes it can be made to describe that domain easily by extending the core of this xADL with other schemas particular to that domain.
- Although it is not used as architecture interchange language it can be used to interchange between the architectural description languages
- It can be used combining two existing domains.
- It serves as basis for experimenting with newer domains. ie., we can check the architecture of the domains for which the architecture is not specified.

Advantages of xADL over the UML

1. UML is vulnerable to changes.
2. UML diagrams are not easy to extend or modify.
3. We cannot model product line architectures efficiently with UML since their architecture changes from version to version.
4. UML encourages object diagrams than the component diagrams.
5. The main purpose of any ADL is to do prior design analysis. But UML does not have any good analysis tools.
6. Using UML diagrams we cannot generate the code which is main need of invention of ADLs. (we can do it UML2.0 through xml Schemas)

7. UML2.0 will not cover all the existing domains. And also if a domain comes it will not able to extend its features to accommodate the changes in its description.

Drawbacks of xADL2.0:

1. xADL 2.0 does not solve the feature interaction problem. Feature interaction problem: In a software system, a feature is an optional unit or increment of functionality. If the system specification is organized by features, then it probably takes the form B + F1 + F2 + F3 . . ., where B is a base specification, each Fi is a feature module, and + denotes some feature-composition operation. A feature interaction is some way in which a feature or features modify or influence another feature in defining overall system behavior. Feature interaction is necessary and inevitable in a feature-oriented specification, because so little can be accomplished by features that are completely independent. A bad feature interaction is one that causes the specification to be incomplete, inconsistent, or unimplementable, or that causes the overall system behavior to be undesirable. Non-associatively of feature composition can also be considered a feature-interaction problem, on the grounds that for features to be truly optional, their compositions must yield the same behavior when performed in any order. Some forms of nondeterministic behavior can also be considered feature-interaction problems. Using xADL we cannot find this feature interaction problem in the architecture of the system.
2. Also when presented with two different schemas xADL will not model the architecture. It will only choose one schema over the other.
3. Other drawback of this ADL is as it is developed recently it does not have much practical experiences. And also the full documentation of the ADL is also not available from the homepage. So the information about the classes (java libraries are also not well documented on the site).

Architecture Modeling:

ArchStudio creates and manipulates architecture descriptions expressed in the xADL architecture description language (ADL). xADL is the first modularly-extensible architecture description language. Rather than having its syntax and semantics defined monolithically, in one huge chunk, xADL breaks up modeling features into modules using standard XML schemas. ArchStudio's

integrated set of tools operates on xADL documents much the same way as a word processor operates on text documents. One major difference, however, is that ArchStudio tools integrate "live"—meaning that a change in any tool is reflected in all others immediately. As noted above, the xADL language can be extended by end-users through the addition of new XML schemas to support domain- or project-specific concerns and modeling needs. New modules are even added to the core xADL language from time to time as they are developed and contributed.

Please do modeling (minimum 2 components & 1 connector) for your chosen project.

Output:

xADL code for the project:

```
<component type="Component" id="component1">
  <description type="Description">Candiadte Login</description>
  <interface type="Interface" id="interface1">
    <description type="Description">To proceed Login</description>
    <direction type="Direction">out</direction>
  </interface>
</component>

<connector type="Connector" id="connector1">
  <description type="Description">Search Jobs</description>
  <interface type="Interface" id="interface1">
    <description type="Description">To proceed Login</description>
    <direction type="Direction">in</direction>
  </interface>
  <interface type="Interface" id="interface2">
```

```
    <description type="Description">To search Jobs</description>
    <direction type="Direction">out</direction>
  </interface>
</connector>

<component type="Component" id="component2">
  <description type="Description">Find Jobs by Title</description>
  <interface type="Interface" id="interface1">
    <description type="Description">To Search job by Title</description>
    <direction type="Direction">in</direction>
  </interface>
  <interface type="Interface" id="interface2">
    <description type="Description">To Apply for Jobs</description>
    <direction type="Direction">out</direction>
  </interface>
</component>

<component type="Component" id="component3">
  <description type="Description">Find Jobs by keyword</description>
  <interface type="Interface" id="interface1">
    <description type="Description">To Search job by keyword</description>
    <direction type="Direction">in</direction>
  </interface>
  <interface type="Interface" id="interface2">
```

```
    <description type="Description">To Apply for Jobs</description>
    <direction type="Direction">out</direction>
  </interface>
</component>

<component type="Component" id="component4">
  <description type="Description">Find Jobs by Filters</description>
  <interface type="Interface" id="interface1">
    <description type="Description">To Search job by Filters</description>
    <direction type="Direction">in</direction>
  </interface>
  <interface type="Interface" id="interface2">
    <description type="Description">To Apply for Jobs</description>
    <direction type="Direction">out</direction>
  </interface>
</component>

<connector type="Connector" id="connector2">
  <description type="Description">Apply for Jobs</description>
  <interface type="Interface" id="interface1">
    <description type="Description">To get filtered Jobs/description>
    <direction type="Direction">in</direction>
  </interface>
  <interface type="Interface" id="interface2">
```

```
    <description type="Description">To Apply for Job</description>
    <direction type="Direction">out</direction>
  </interface>
</connector>

<component type="Component" id="component5">
  <description type="Description">Submit Resume</description>
  <interface type="Interface" id="interface1">
    <description type="Description">To Submit resume for filtered Jobs</description>
    <direction type="Direction">in</direction>
  </interface>
</component>

<component type="Component" id="component6">
  <description type="Description">Schedule Test</description>
  <interface type="Interface" id="interface1">
    <description type="Description">Schedule Test for Filtered Jobs</description>
    <direction type="Direction">in</direction>
  </interface>
</component>

<component type="Component" id="component7">
  <description type="Description">Look for Match Skills</description>
  <interface type="Interface" id="interface1">
```

```
    <description type="Description">To apply for Matched Skill</description>
    <direction type="Direction">in</direction>
  </interface>
</component>

<component type="Component" id="component8">
  <description type="Description">Edit Job Status</description>
  <interface type="Interface" id="interface1">
    <description type="Description">To edit status after Scheduling Jobs</description>
    <direction type="Direction">in</direction>
  </interface>
</component>
```

Learning Outcomes: The student should have the ability to

LO1: identify problems during installation of Eclipse.

LO2: Write components and connectors in xADL

Course Outcomes: Upon completion of the course students will be able to write in xADL

Conclusion:……

For Faculty Use

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

EXPERIMENT NO. 02: VISUALIZATION USING XADL2.0

AIM: Students should be able to implement Visualization using xADL2.0

THEORY:

Architecture is *the set of principal design decisions* made about a system. Recall also that models are artifacts that capture some or all of the design decisions that comprise an architecture. An architectural visualization defines how architectural models are depicted, and how stakeholders interact with those depictions. Two key aspects here:

- Depiction is a picture or other visual representation of design decisions
- Interaction mechanisms allow stakeholders to interact with design decisions in terms of the depiction

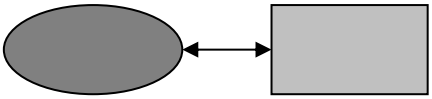
It is easy to confuse models and visualizations because they are very closely related. To make this distinction explicit:

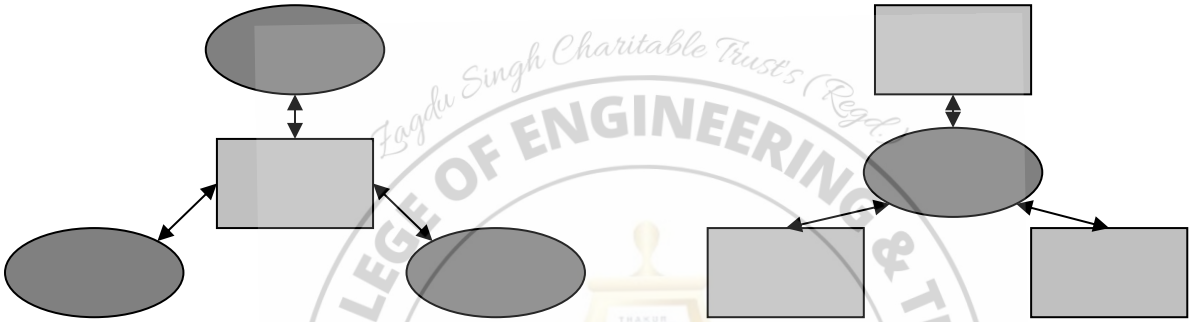
- A model is just abstract information – a set of design decisions
- Visualizations give those design decisions form: they let us depict those design decisions and interact with them in different ways. Because of the interaction aspect, visualizations are often active – they are both pictures AND tools

Canonical Visualization:

Each modeling notation is associated with one or more canonical visualizations. This makes it easy to think of a notation and visualization as the same thing, even though they are not. Some notations are canonically textual eg Natural language, XML-based ADLs or graphical eg PowerPoint-style or a little of both eg UML or have multiple canonical visualizations eg Darwin

Different Relationships





Kinds of Visualizations

1. Textual Visualization: Depict architectures through ordinary text files. Generally conform to some syntactic format, like programs conform to a language. May be natural language, in which case the format is defined by the spelling and grammar rules of the language. Decorative options: Fonts, colors, bold/italics & Tables, bulleted lists/outlines.

   Advantages

   a. Depict entire architecture in a single file

   b. Good for linear or hierarchical structures

   c. Hundreds of available editors

   d. Substantial tool support if syntax is rigorous (e.g., defined in something like BNF)

Disadvantages

e. Can be overwhelming

f. Bad for graph like organizations of information

g. Difficult to reorganize information meaningfully

h. Learning curve for syntax/semantics

2. Graphical Visualizations: Depict architectures (primarily) as graphical symbols like Boxes, shapes, pictures, clip-art, Lines, arrows, other connectors, Photographic images, Regions, shading, 2D or 3D. Generally conform to a symbolic syntax but may also be 'free-form' and stylistic.

   Advantages

   a. Symbols, colors, and visual decorations more easily parsed by humans than structured text

   b. Handle non-hierarchical relationships well

   c. Diverse spatial interaction metaphors (scrolling, zooming) allow intuitive navigation

   Disadvantages

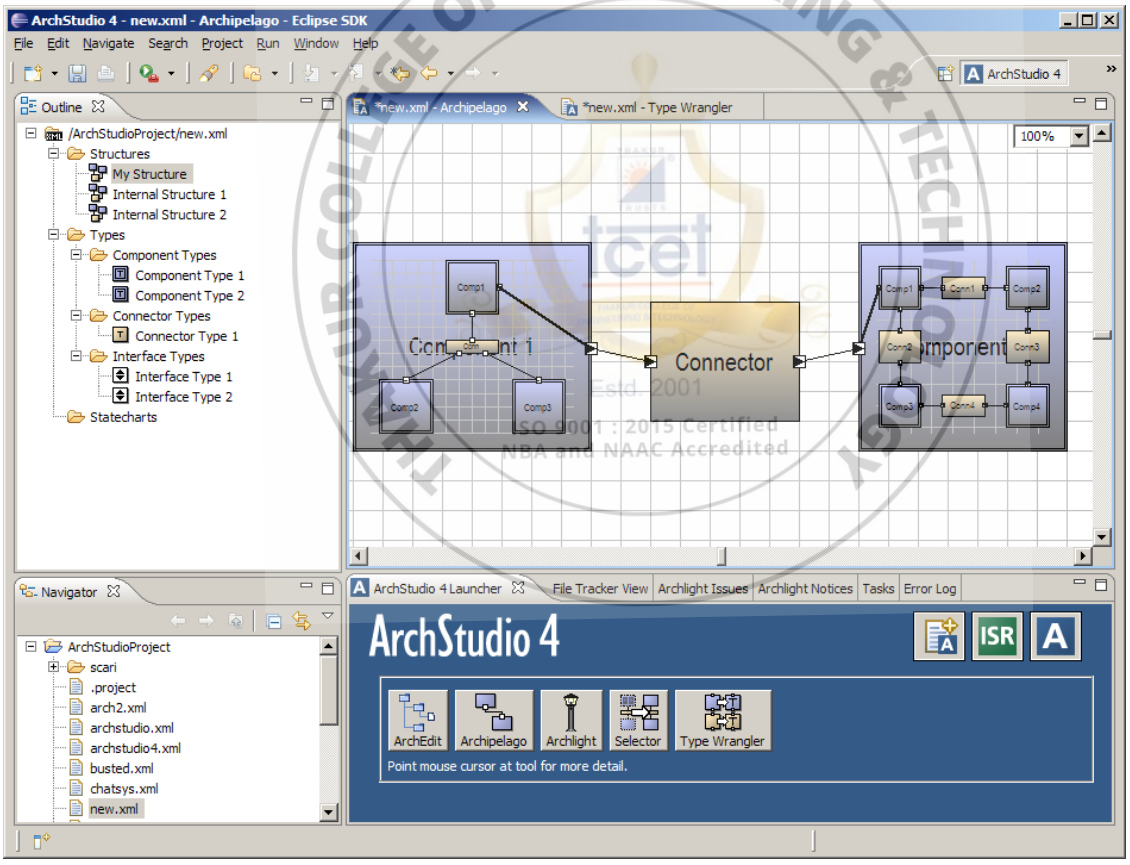
   a. Cost of building and maintaining tool support

      i. Difficult to incorporate new semantics into existing tools

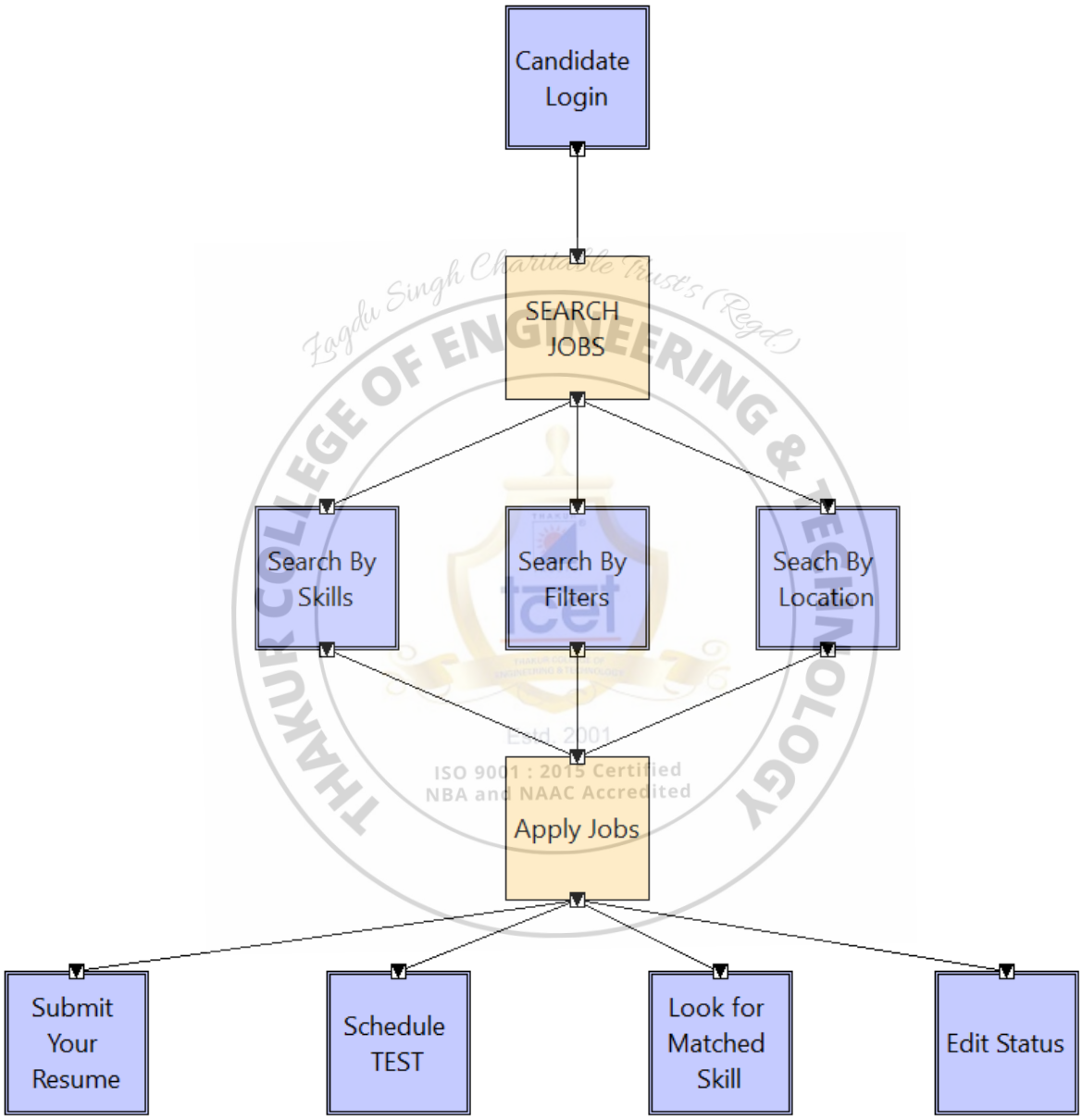
   b. Do not scale as well as text to very large models

3. Hybrid Visualizations: Many visualization are text-only few graphical notations are purely symbolic. Text labels, are at a minimum. Annotations are generally textual as well. Some notations incorporate substantial parts that are mostly graphical alongside substantial parts that are mostly or wholly textual.

Architecture Visualization: The xADL language defines the structure of architecture description data, but it can be depicted and manipulated in many ways. ArchStudio provides several different visualizations for xADL models. Archipelago, ArchStudio's graphical editor, provides visualizations as symbol graphs - the kind of box-and-arrow models common in tools like Microsoft Visio and OmniGraffle. However, unlike PowerPoint or OmniGraffle models, the graphical depictions in Archipelago aren't just pictures - they are a user-editable graphical projection of the underlying architecture model. ArchStudio includes other editors as well, including ArchEdit, a syntax-directed editors that adapts to new xADL schemas automatically with no recoding, and the Type Wrangler, which provides a custom view of an architectural model that makes it easier to achieve type consistency.



Output:


Candidate Login
SEARCH JOBS
Search By Skills
Search By Filters
Seach By Location
Apply Jobs
Submit Your Resume
Schedule TEST
Look for Matched Skill
Edit Status

Learning Outcomes: The student should have the ability to

LO1: create project in ArchStudio.

LO2: Draw components and connectors in ArchStudio

Course Outcomes: Upon completion of the course students will be able to Draw design in Arch Studio.

Conclusion:……

For Faculty Use

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

EXPERIMENT NO. 03: CREATING WEB SETVICES

AIM: Students should be able to create web services.

THEORY :

Web services are client and server applications that communicate over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP)

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.

There are two types of webservices

1. "Big" Web Services

Big web services use XML messages that follow the Simple Object Access Protocol (SOAP) standard, an XML language defining a message architecture and message formats.

Such systems often contain a machine-readable description of the operations offered by the service, written in the Web Services Description Language (WSDL), an XML language for defining interfaces syntactically.

The architecture needs to handle asynchronous processing and invocation. In such cases, the infrastructure provided by standards, such as Web Services Reliable Messaging (WSRM), and APIs, such as JAX-WS, with their client-side asynchronous invocation support, can be leveraged out of the box.

2. RESTful Web Services

RESTful web services, often better integrated with HTTP than SOAP-based services are, do not require XML messages or WSDL service–API definitions.

A RESTful design may be appropriate when the following conditions are met.

The web services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server.

A caching infrastructure can be leveraged for performance.

The service producer and service consumer have a mutual understanding of the context and content being passed along.

Bandwidth is particularly important and needs to be limited. REST is particularly useful for limited-profile devices, such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.

Web service delivery or aggregation into existing web sites can be enabled easily with a RESTful style.

Example of web service

Following is Web Service example which works as a service provider and exposes two methods (add and SayHello) as 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.

FirstService.asmx

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

using System;
using System.Web.Services;
using System.Xml.Serialization;

[WebService(Namespace="http://localhost/MyWebServices/")]
public class FirstService : WebService
{
   [WebMethod]
   public int Add(int a, int b)
   {
      return a + b;
   }

   [WebMethod]
   public String SayHello()
   {
      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's start with configuring the IIS.

- Open Start->Settings->Control Panel->Administrative tools->Internet Services Manager.
- Expand and right-click on [Default Web Site]; select New ->Virtual Directory.
- The Virtual Directory Creation Wizard opens. Click Next.
- The "Virtual Directory Alias" screen opens. Type the virtual directory name and click Next.
- The "Web Site Content Directory" screen opens. Here, enter the directory path name for the virtual directory—for example, c:\MyWebServices—and click Next.
- The "Access Permission" screen opens. Change the settings as per your requirements. Let's keep the default settings for this exercise. Click the Next button. It completes the IIS configuration. Click Finish to complete the configuration.

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

To test our 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.

Testing the Web Service

Let's 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.axpx

```
<%@ Page Language="C#" %>
<script runat="server">
void runSrvice_Click(Object sender, EventArgs e)
{
    FirstService mySvc = new FirstService();
    Label1.Text = mySvc.SayHello();
    Label2.Text = mySvc.Add(Int32.Parse(txtNum1.Text),
              Int32.Parse(txtNum2.Text)).ToString();
}
```

```
</script>
<html>
<head>
</head>
<body>
<form runat="server">
  <p>
     <em>First Number to Add </em>:
     <asp:TextBox id="txtNum1" runat="server"
        Width="43px">4</asp:TextBox>
  </p>
  <p>
     <em>Second Number To Add </em>:
     <asp:TextBox id="txtNum2" runat="server"
        Width="44px">5</asp:TextBox>
  </p>
  <p>
     <strong><u>Web Service Result -</u></strong>
  </p>
  <p>
     <em>Hello world Service</em> :
     <asp:Label id="Label1" runat="server"
        Font-Underline="True">Label</asp:Label>
  </p>

  <p>
     <em>Add Service</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. Thus, the proxy created 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. VS .NET creates a proxy automatically for you when the reference for the Web Service is added. Create a proxy for the Web Service using the

wsdl utility supplied with the .NET SDK. It will create FirstSevice.cs 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:library FirstService.cs
```

- Put the compiled proxy in the bin directory of the virtual directory of the Web Service (c:\MyWebServices\bin). IIS looks for the proxy in this directory.
- Create the service consumer, which we have already done. Note that I have instantiated an object of the Web Service proxy 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).

Windows Application-Based Web Service Consumer

Writing a Windows application-based Web Service consumer is the same as writing any other Windows application. The only work to be done is 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

```
using System;
using System.IO;

namespace SvcConsumer{
class SvcEater
{
   public static void Main(String[] args)
   {
      FirstService mySvc = new FirstService();

      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

References:

1. http://www.tutorialspoint.com/webservices/

Code & output:

Python file:

```python
import requests
BASE_URL = 'https://fakestoreapi.com'

#Get Request
print("========================== GET REQUEST ==========================")
response = requests.get(f"{BASE_URL}/products")
print(response.json())
print("========================== STATUS OF REQUEST ==========================")
print(response.status_code)

#Post Request
print("========================== POST REQUEST ==========================")
new_product = {
    "title": 'test product',
    "price": 13.5,
    "description": 'lorem ipsum set',
    "image": 'https://i.pravatar.cc',
    "category": 'electronic'
}
response = requests.post(f"{BASE_URL}/products", json=new_product)
print(response.json())
print("========================== STATUS OF REQUEST ==========================")
print(response.status_code)
```

Output:

```
webservice
C:\Users\khanu\PycharmProjects\CSS\venv\Scripts\python.exe C:/Users/khanu/PycharmProjects/CSS/venv/webservice.py
========================== GET REQUEST ==========================
[{'id': 1, 'title': 'Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops', 'price': 109.95, 'description': 'Your perfect pack for everyday use and
========================== STATUS OF REQUEST ==========================
200
========================== POST REQUEST ==========================
{'id': 21, 'title': 'test product', 'price': 13.5, 'description': 'This is TEST POST', 'image': 'https://i.pravatar.cc', 'category': 'electronic'}
========================== STATUS OF REQUEST ==========================
200

Process finished with exit code 0
```

Learning Outcomes: The student should have the ability to

LO1: create webservices.

LO2: understand use of webservices

Course Outcomes: Upon completion of the course students will be able to develop webservices

Conclusion:……
…

For Faculty Use

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

Experiment no. 04: Integrate software components using a middleware

AIM: To integrate software components using a middleware

THEORY:

Software architectures promote development focused on modular building blocks and their interconnections. Since architecture-level components often contain complex functionality, it is reasonable to expect that their interactions will also be complex. Modeling and implementing software connectors thus becomes a key aspect of architecture-based development. Software interconnection and middleware technologies such as RMI, CORBA, ILU, and ActiveX provide a valuable service in building applications from components. We have to understand the tradeoffs among these technologies with respect to architectures. Several off-the-shelf middleware technologies are used in implementing software connectors.

The Role of Middleware

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.

Java's Remote Method Invocation (RMI) [24] is a technology developed by Sun Microsystems to allow Java objects to invoke methods of other objects across process and machine boundaries. RMI supports several standard distributed application concepts, namely registration, remote method calls, and distributed objects. Currently, RMI only supports Java applications, but there is indication of a forthcoming link between RMI and CORBA that would remedy this. Each RMI object that is to be shared in an application defines a public interface (a set of methods) that can be called remotely. This is similar to the RPC mechanism. These methods are the only means of communication across a process boundary via RMI. Because RMI is not a software bus, it has no concept of routing, filtering, or messages. However, Java's RMI built-in serialization and deserialization capabilities handle marshalling of basic and moderately complex Java objects, including C2 messages. RMI is fully compatible with the multithreading capabilities built into the

Java language, and is therefore well suited for a multithreaded application. It allows communication among objects running in different processes which may be on different machines. Communication occurs exclusively over the TCP/IP networking protocol. RMI supports application modification at run-time, a capability enabled by Java's dynamic class loading.

Addition of 2 numbers using JAVA RMI:

Define an interface that declares remote methods.

The first file AddServerIntf.java that defines the remote interface remains the same.

```
import java.rmi.*;

public interface AddServerIntf extends Remote {
  double add(double d1, double d2) throws RemoteException;
}
```

Implement the remote interface and the server

The second source file AddServerImpl.java (it implements the remote interface) also remains the same, with one minor variation: it calls the super-class constructor explicitly.

```
import java.rmi.*;
import java.rmi.server.*;

public class AddServerImpl extends UnicastRemoteObject
  implements AddServerIntf {

  public AddServerImpl() throws RemoteException {
  super();
  }

  public double add(double d1, double d2) throws RemoteException {
    return d1 + d2;
  }
}
```

The revised version of the third source file AddServer.java includes the security manager, assumes that you will run the server on jupiter using the port 56789, and uses a slightly modified name "MyAddServer" for registration purposes.

```
import java.net.*;
import java.rmi.*;
public class AddServer {
  public static void main(String args[]) {
```

```
    // Create and install a security manager

    if (System.getSecurityManager() == null) {
           System.setSecurityManager(new RMISecurityManager());
    }
    try {
      AddServerImpl addServerImpl = new AddServerImpl();

        // You want to run your AddServer on jupiter using the port 56789
        // and you want to use rmi to connect to jupiter from your local
machine
        // Note that to accomplish this you have to start on jupiter
        // rmiregistry 56789 &
        // in the directory that contains NO classes related to this server
!!!

      Naming.rebind("rmi://jupiter.scs.ryerson.ca:56789/MyAddServer",
addServerImpl);
    }
    catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            e.printStackTrace();
    }
  }
}
```

Copy files **AddServerIntf.java**, **AddServerImpl.java**, and **AddServer.java** to your directory on jupiter and compile them as usual using javac.

Develop a client (an application or an applet) that uses the remote interface

The revised version of the fourth source file AddClient.java has a few new features.

- It includes also the security manager;
- it requires 4 command-line arguments: the name of the remote server, the port where the naming **rmiregistry** will be running, and two numbers;
- it prints the URL that will be used to connect to the server.

```
import java.rmi.*;

   // USAGE: java AddClient   firstNum secondNum

public class AddClient {
```

```
  public static void main(String args[]) {

    // The client will try to download code, in particular, it will
    // be downloading stub class from the server.
    // Any time code is downloaded by RMI, a security manager must be
present.

    if (System.getSecurityManager() == null) {
          System.setSecurityManager(new RMISecurityManager());
    }

    try {
      String addServerURL = "rmi://" + args[0] + ":" + args[1] +
"/MyAddServer";

     System.out.println("I will try to invoke the remote method from  " +
addServerURL);

      AddServerIntf remoteObj =
                  (AddServerIntf) Naming.lookup(addServerURL);

      System.out.println("The first number is: " + args[2]);

      double d1 = Double.valueOf(args[2]).doubleValue();

      System.out.println("The second number is: " + args[3]);

      double d2 = Double.valueOf(args[3]).doubleValue();

      // Now we invoke from a local machine the remote method "add"

      System.out.println("The sum is: " + remoteObj.add(d1, d2));
    }
    catch(Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Keep this file on your local machine together with the remote interface and the **rmi.policy** file that controls access to your local machine:

```
grant {
  // The simplest (but uncarefull) policy is allow everything:
  //    permission java.security.AllPermission;
  // More secure policy is the following.
  //
  // jupiter.scs.ryerson.ca   This is the rmihost - RMI registry and the
server
  // www.scs.ryerson.ca       This is webhost - HTTP server for stub classes
  permission java.net.SocketPermission
    "jupiter.scs.ryerson.ca:1024-65535", "connect,accept";
  permission java.net.SocketPermission
    "www.scs.ryerson.ca:80", "connect";
```

```
};
```

You also have to copy this policy file to your directory on jupiter that contains all server-related files.

Generate stubs and skeletons

Next, go to the server (jupiter), and change into the directory that contains **AddServerIntf.class** (interface), **AddServerImpl.class** (its implementation), and **AddServer.class** (server itself) and **rmi.policy** file. In that directory, run **rmic** compiler:

rmic AddServerImpl

This command generates two new files: **AddServerImpl_Skel.class** (skeleton) and **AddServerImpl_Stub.class** (stub).

Start the RMI registry

Before you proceed, copy stub and interface classes to a directory, where the web server can access them, e.g. to the world-accessible sub-directory **JavaClasses** of your **public_html** directory:

```
ls -t -l ~mes/public_html/JavaClasses/
total 24
-rw-r--r--   1 mes      mes          205   AddServerIntf.class
-rw-r--r--   1 mes      mes         1612   AddServerImpl_Skel.class
-rw-r--r--   1 mes      mes         3171   AddServerImpl_Stub.class
```

Check that all these files are accessible, e.g., try to download them using Netscape or IE: if you succeeded, then they are accessible.

On jupiter in the directory that does NOT contain any server related classes and assuming that those classes are NOT on you CLASSPATH, start rmiregistry:

rmiregistry 56789 &

For example, create the temporary directory tmpTEST, go to that directory and start there **rmiregistry** naming system. By default, rmiregistry naming system loads stub and skeleton files from directories mentioned in your **CLASSPATH**. But because you want to load them dynamically from your web directory `JavaClasses`, you want to hide these files from **rmiregistry**: this way you force to load required files from the codebase given below as a command-line argument.

Start the server

In the directory that contains all server related classes:

```
-rw-r--r--   1 mes      mes          1075       AddServer.class
-rw-r--r--   1 mes      mes           363       AddServerImpl.class
-rw-r--r--   1 mes      mes          1612       AddServerImpl_Skel.class
-rw-r--r--   1 mes      mes          3171       AddServerImpl_Stub.class
-rw-r--r--   1 mes      mes           205       AddServerIntf.class
-rw-r--r--   1 mes      mes           511       rmi.policy
```

we can run the server:

```
java -Djava.security.policy=rmi.policy
  -Djava.rmi.server.codebase=http://www.scs.ryerson.ca/~mes/JavaClasses/
AddServer &
```

Note that the URL given to "codebase" ends with / and use your own login name, of course.

Run the client

Now, go to the directory of your **local computer** that contains only 3 files:

- **AddClient.class** (client),
- **AddServerIntf.class** (interface),
- **rmi.policy**: your policy file.

For example, you can create a new directory and copy there 3 files mentioned in this section.

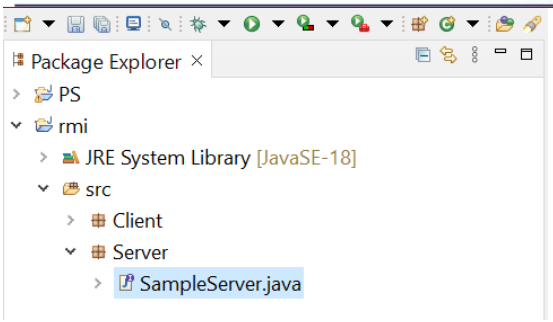
This time, you would like to load the stub class dynamically from the server. Assume that the server side was developed by a different company, you are responsible only for the client application and you do not have access to the server-related files when you start your client. All you know is that the RMI server will be running on jupiter and you can connect to the registry at the port 56789 if you need to invoke remote methods on the server.

Finally, you can run the client application on your local machine:

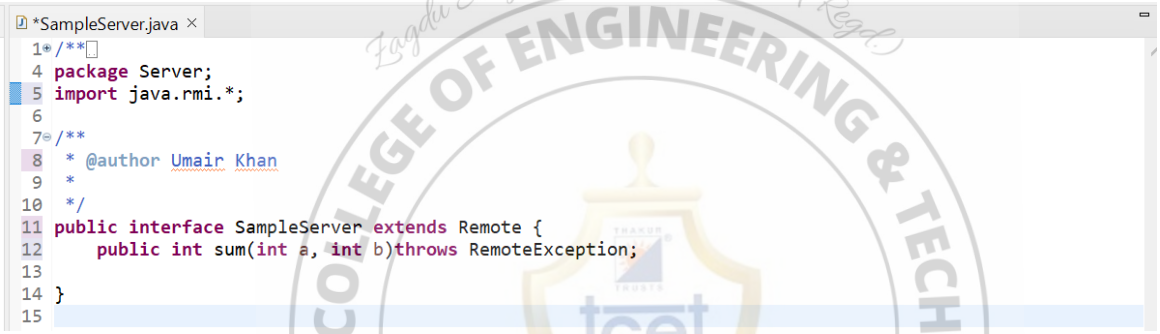
```
java -Djava.security.policy=rmi.policy AddClient jupiter.scs.ryerson.ca 56789
456 544
I will try to invoke the remote method from
rmi://jupiter.scs.ryerson.ca:56789/MyAddServer
The first number is: 456
The second number is: 544
The sum is: 1000.0
```

Output:

1) Create Client & Server packages in New Java project



2) Define the remote interface In Server Package



3) Develop the implementation class remote object in Server package

*SampleServer.java *SampleServerImpl.java

```
1  package Server;
2  import java.rmi.*;
3  import java.rmi.server.*;
4  /**
5   * @author Umair Khan
6   */
7  @SuppressWarnings("serial")
8  public class SampleServerImpl extends UnicastRemoteObject implements SampleServer{
9
10     SampleServerImpl()throws RemoteException
11     {
12         super();
13     }
14     public int sum(int a,int b)throws RemoteException
15     {
16         return a+b;
17     }
18
19     public static void main(String[] args)
20     {
21         // TODO Auto-generated method stub
22         try {
23             SampleServerImpl Server = new SampleServerImpl();
24             Naming.rebind("SampleServer", Server);
25             System.out.println("Server Waiting");
26         }
27         catch(java.net.MalformedURLException me)
28         {
29             System.out.println("malformes url:"+me.toString());
30         }
31         catch(RemoteException re) {
32             System.out.println("remote exception:"+re.toString());
33         }
34
35     }
```


4) Develop the Client Program in Client Package

*SampleServer.java *SampleServerImpl.java *SampleClient.java

```java
/**
 */
package Client;
import Server.SampleServer;
import java.rmi.*;
/**
 * @author Umair Khan
 *
 */
public class SampleClient {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        try {
            System.out.println("Security Manager loaded");
            String url ="//localhost/SampleServer";
            SampleServer remoteobject=(SampleServer)Naming.lookup(url);
            System.out.println("got remote Object");
            System.out.println("1+2="+remoteobject.sum(1, 2));
        }
        catch(RemoteException exc)
        {
            System.out.println("error in lookup"+exc.toString());
        }
        catch(java.net.MalformedURLException exc)
        {
            System.out.println("malformed url"+exc.toString());
        }
        catch(java.rmi.NotBoundException exc)
        {
            System.out.println("notbound"+exc.toString());
        }

    }
}
```

5) Now Compile & Execute Program in terminal

Command Prompt - java AddServer.java

```
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\khanu>cd C:\Users\khanu\OneDrive\Desktop\Lab

C:\Users\khanu\OneDrive\Desktop\Lab>javac AddRemImpl.java

C:\Users\khanu\OneDrive\Desktop\Lab>javac AddServer.java

C:\Users\khanu\OneDrive\Desktop\Lab>start rmiregistry

C:\Users\khanu\OneDrive\Desktop\Lab>java AddServer.java
```

Command Prompt - java AddClient.java

```
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\khanu>cd C:\Users\khanu\OneDrive\Desktop\Lab

C:\Users\khanu\OneDrive\Desktop\Lab>javac AddRem.java

C:\Users\khanu\OneDrive\Desktop\Lab>javac AddClient.java

C:\Users\khanu\OneDrive\Desktop\Lab>java AddClient.java
Enter the 1st Parameter
66
Enter the 2nd parameter
69
The output is:
135
```

Learning Outcomes: The student should have the ability to

LO1: understand middle wares

LO2: write middlewares

Course Outcomes: Upon completion of the course students will be able to develop middleware

Conclusion:……

For Faculty Use

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

Experiment no. 5: Use middleware to implement connectors

AIM: Use middleware JAVA RMI to implement stream connectors

THEORY:

Software connectors have been embraced as a critical abstraction by software architecture researchers. Connectors remove from components the responsibility of knowing how they are interconnected. They also introduce a layer of indirection between components. The potential penalties paid due to this indirection (e.g., performance) should be outweighed by other benefits of connectors, such as their encapsulation of complex intercommunication protocols that can be reused relatively inexpensively across applications. Modeling and implementation of software connectors with potentially complex protocols thus becomes an important aspect of architecture-based development.

Because software connectors provide a uniform interface to other connectors and components within an architecture, architects need not be concerned with the properties of different middleware technologies as long as the technology can be encapsulated within a software connector. Internally, however, connectors based on different middleware technologies have different abilities. Implementors of a given architecture can use this knowledge to determine which middleware solutions are appropriate in a given implementation of an architecture. In this way, encapsulating middleware functionality within software connectors maintains the integrity of an architectural style by keeping it separate from implementation-dependent factors such as how to bridge process boundaries within a single architecture.

Introduction to Java RMI.

The Java RMI (Remote Method Invocation) is a package for writing and executing distributed Java programs. The Java RMI provides a framework for developing and running servers (server objects). The services (methods) provided by those server objects can be accessed by clients in a

way similar to method invocation. I.e. Java RMI hides almost all aspects of the distribution and provides a uniform way by which objects (distributed or not) may be accessed.

Writing Java programs using RMI can be described by the following steps:

- write server interface(s),
- write server implementation,
- generate server skeleton and client stub,
- write client implementation.

Executing distributed Java programs using RMI can be described by the following steps:

- start the rmi registry,
- start the server object,
- run the client(s).

A server (RMIServer.java) will provide the methods String getString() and void setString(String s). A client (RMIClient.java) may use those two methods for retrieving and storing a string in the server, i.e. the client may modify and inspect the local state of the server object.

The following sections will develop this server and a corresponding client.

The server interface.

The server interface is used by the stub/skeleton compiler when generating the client stub and the server skeleton files. This interface thus defines the methods in the server, which may be invoked by the clients.

There are two issues to remember when writing such an interface. First, the interface has to be written as extending the java.rmi.Remote interface. Second, all methods in the interface must throw java.rmt.RemoteException. This exception must thus be caught when the clients are invoking any of the servers methods, thus the clients may have a way to determine if a method invocation was successful.

The complete source code for the server interface (ServerInterface.java) is included below.

```
package examples.rmi;
import java.rmi.*;

public interface ServerInterface extends Remote
{

   public String getString() throws RemoteException;
   public void setString(String s) throws RemoteException;

}
```

The interface is compiled by the javac compiler to generate the file `ServerInterface.class`.

Writing the server object.

The server must be written as a "regular" Java program, i.e. a program with a method public static void main(String argv[]). Through this main method, server objects may be instantiated and registered with the rmi registry.

Each distributed object is identified by a string, specifying the object name. This string is registered with the rmi registry and is used by the clients when requesting a reference to the server object. The following lines of code indicates how an instance of RMIServer can be registered with the rmi registry under the string name "RMIServer". The following code would typically appear in the main method of the server:

```
String name = "RMIServer";
RMIServer theServer = new RMIServer();
Naming.rebind(name,theServer);
```

Server objects must - of course - implement the defined interfaces and in addition typically extend java.rmi.server.UnicastRemoteObject. At the abstract level, this should potentially enable various kinds of distribution schemes (e.g. replication of objects, multicast groups of objects etc.) by extending different classes. However in current implementations of Java, only java.rmi.server.UnicastRemoteObject is available. The only practical advantage of inheriting from java.rmi.sever.UnicastRemoteObject is that it will preserve the usual semantics of the methods hashCode(), equals() and toString() in a distributed environment.

```
package examples.rmi;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.*;

public class RMIServer extends UnicastRemoteObject implements ServerInterface
{
   private String myString = "";
```

```
   // The default constructor
   public RMIServer() throws RemoteException
     {
       super();
     }

   // Implement the methods from the ServerInterface
   public void setString(String s) throws RemoteException
     {
       this.myString = s;
     }

  public String getString() throws RemoteException
     {
       return myString;
     }

   // The main method: instantiate and register an instance of the
   // RMIServer with the rmi registry.
   public static void main(String argv[])
     {
       try
         {
           String name = "RMIServer";
           System.out.println("Registering as: \""+name+"\"");
           RMIServer theServer = new RMIServer();
           Naming.rebind(name,theServer);
           System.out.println(name+" ready...");
         }
       catch(Exception e)
         {
           System.out.println("Exception while registering: "+e);
         }
     }
}
```

The RMIServer.java is compiled using the default javac to generate the file RMIServer.class.

Generating skeleton and stub.

Based on the compiled ServerInterface and RMIServer files, a client stub and a server skeleton can be generated. The server skeleton acts as interface between the rmi registry and the server objects residing on a host. Likewise, the client stub of the server is returned to the client when a reference to the remote object is requested. The exact details (using the skeleton and stub) are all taken care of by the runtime environment.

To generate the skeleton and the stub, the rmic compiler is used. Like the Java virtual machine, the rmic compiler requires fully qualified class names, i.e. to generate the server skeleton and the client stub for the RMIServer, the following command must be invoked:

```
rmic examples.rmi.RMIServer
```

This generates the files `RMIServer_Skel.class` and `RMIServer_Stub.class` .

Writing the client implementation.

When writing a client implementation, three things must be done. First, a "security manager" must be installed. Such a security manager specifies the security policy, i.e. decides which constraints are imposed on the server stubs. An appropriate security manager for these examples can be installed by the following code:

```
System.setSecurityManager(new java.rmi.RMISecurityManager());
```

Second, a reference to the remote object must be requested. To do so, the hostname of the host where the object resides as well as the string name, under which the object is registered, is required. In this example, the object was registered with the string name "RMIServer". Assuming that the server was started on the host "objecthost.domain.com", the following line of code may be used to get a reference to the object:

```
String name = "rmi://objecthost.domain.com/RMIServer"
server = (ServerInterface)Naming.lookup(name);
```

The code above contacts the rmi registry at "objecthost.domain.com" and asks for the stub for the object, registered under the name "RMIServer".

Thus in reality, Naming.lookup() returns an instance of the RMIServer_stub. However the available methods in the server object (and thus in the stub) are defined by ServerInterface. Thus whatever Naming.lookup() returns is typecast into a ServerInterface.

```
package examples.rmi;
import java.rmi.server.*;
import java.rmi.*;

public class RMIClient
{
   public static void main (String argv[])
     {
       // Parse the commandline to get the hostname where
       // the server object resides
       String host = "";

       if (argv.length == 1)
```

```
        {
          host = argv[0];
        }
      else
        {
          System.out.println("Usage: RMIClient server");
          System.exit(1);
        }

      // Install a security manager.
      System.setSecurityManager(new RMISecurityManager());

      // Request a reference to the server object
      String name = "rmi://"+host+"/RMIServer";
      System.out.println("Looking up: "+name);

      ServerInterface server = null;
      try
        {
          // In reality, Naming.lookup() will return an instance of
          // examples.rmi.RMIServer_stub.
          // This is typecast into the ServerInterface, which is what
          // specifies the available server methods.
          server = (ServerInterface)Naming.lookup(name);
        }
      catch(Exception e)
        {
          System.out.println("Exception " +e);
          System.exit(1);
        }

      // Given a reference to the server object, it is now
      // possible to invoke methods as usual:
      try
        {
          server.setString("Foobar");
          System.out.println("String in server: "
                             +server.getString());
        }
      catch(Exception e)
        {
          System.out.println("Exception " +e);
          System.exit(1);
        }
    }
}
```

Running the example.

The first thing to do when running Java programs using RMI is to start the rmi registry:

```
install:~> rmiregistry &
[1] 1449
install:~>
```

Next, the server must be started:

```
install:~> java examples.rmi.RMIServer
Registering as: "RMIServer"
RMIServer ready...
```

Finally, the client may be started and the setup tested:

```
install:~>        java        examples.rmi.RMIClient        install.cs.auc.dk
Looking                 up:                rmi://install.cs.auc.dk/RMIServer
String                   in                  server:                    Foobar
install:~>
```

Result/Output:

1) Hello Class

```java
import java.rmi.*;
import java.rmi.server.*;

public class Hello extends UnicastRemoteObject implements HelloInterface
{
  private String message;

  public Hello (String msg) throws RemoteException
  {
    message = msg;
  }

  public String say() throws RemoteException
  {
    return message;
  }
}
```

2) HelloClient

HelloClient.java > ...

```java
import java.rmi.*;

public class HelloClient
{
  Run | Debug
  public static void main (String[] argv)
  {
    try
    {
      HelloInterface hello = (HelloInterface) Naming.lookup(name: "//localhost/Hello");
      System.out.println(hello.say());
    }
    catch (Exception e)
    {
      System.out.println ("HelloClient exception: " + e);
    }
  }
}
```

3) HelloInterface

HelloInterface.java > ...

```java
import java.rmi.*;

public interface HelloInterface extends Remote
{
  public String say() throws RemoteException;
}
```

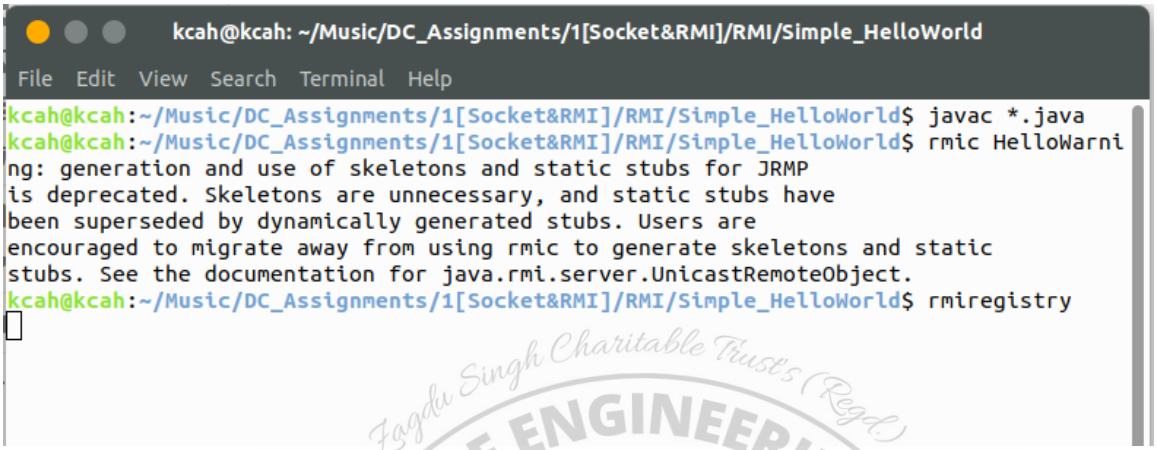
4) HelloServer

HelloServer.java > ...

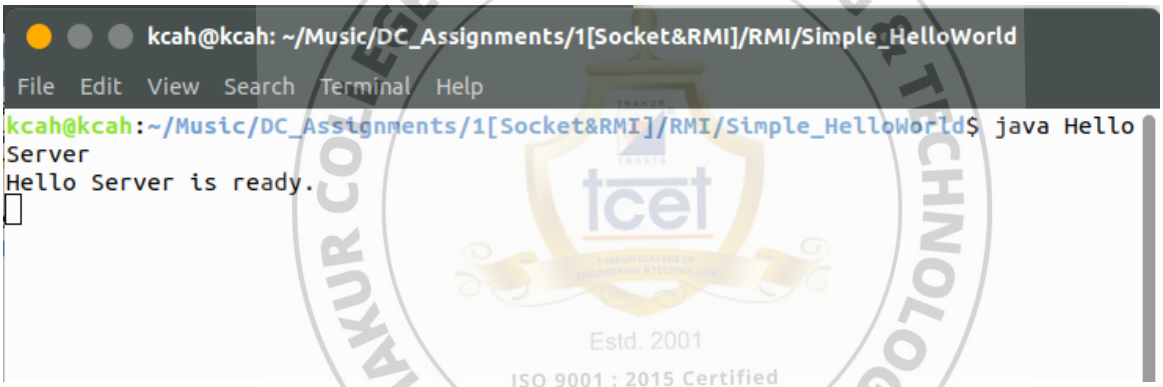
```java
import java.rmi.*;

public class HelloServer
{
  Run | Debug
  public static void main (String[] argv)
  {
    try
    {
      Hello robj = new Hello (msg: "Hello, world!");
      Naming.rebind (name: "Hello", robj);
      System.out.println (x: "Hello Server is ready.");
    }
    catch (Exception e)
    {
      System.out.println ("Hello Server failed: " + e);
    }
  }
}
```

Terminal Results:

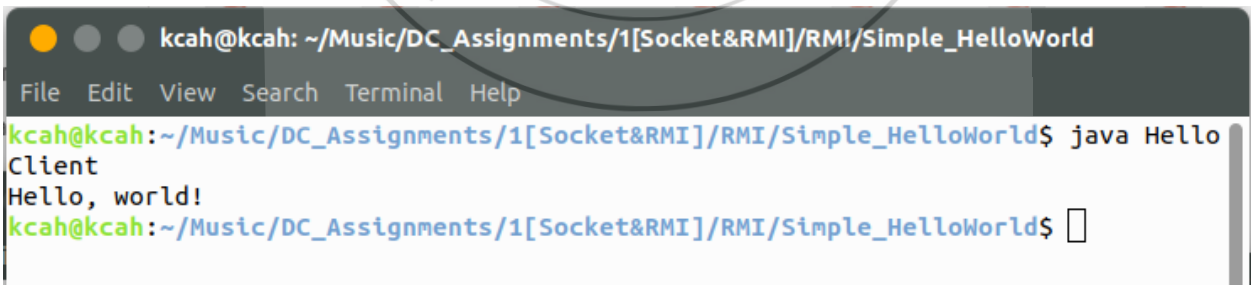
1) Initial Setup



2) Initiate Server



3) Client Side Response



Learning Outcomes: The student should have the ability to

LO1: understand types of connectors

LO2: write connectors using middleware

Course Outcomes: Upon completion of the course students will be able to develop connectors using middlewares

Conclusion:……

For Faculty Use

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

Experiment no. 06: wrappers to connect two applications with different architecture

AIM: Students should be able to implement wrappers to connect two applications with different architecture

Theory : Wrapper classes are used to convert any data type into an object.The primitive data types are not objects; they do not belong to any class; they are defined in the language itself. Sometimes, it is required to convert data types into objects in Java language. For example, upto JDK1.4, the data structures accept only objects to store. A data type is to be converted into an object and then added to a Stack or Vector etc.
a wrapper class wraps (encloses) around a data type and gives it an object appearance. Wherever, the data type is required as an object, this object can be used. Wrapper classes include methods to unwrap the object and give back the data type.

```
int k = 100;
Integer it1 = new Integer(k);
```

```
int m = it1.intValue();
System.out.println(m*m);
```

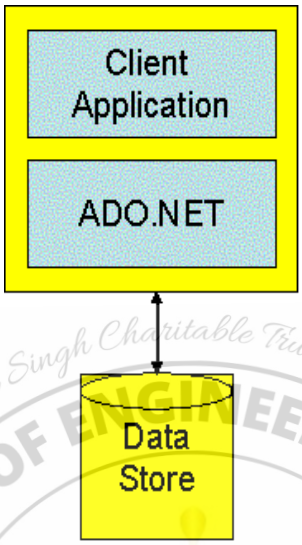
There are mainly two uses with wrapper classes.

1. To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.
2. To convert strings into data types (known as parsing operations), here methods of type parseXXX() are used.
3. So that null values can be there.

Create wrapper classes for connecting two tier and three tier application.

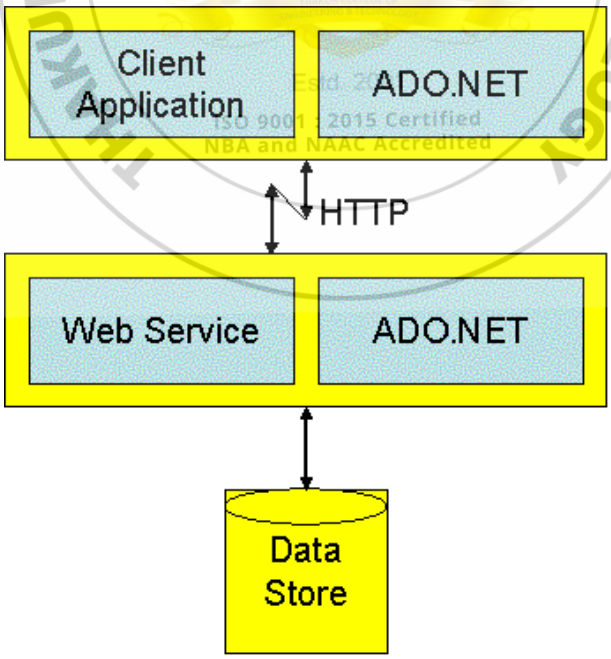
Two-Tier Application Architecture

A typical two-tier application is a client application using ADO.NET communicating directly with a database server, like Microsoft SQL Server™. There are no intervening layers between the client application and the database other than ADO.NET e.g. Client Server applications



Three-Tier Application Using an XML Web Service

Another design option is to use an XML Web service to separate the database's access to another component that returns the data to the front-end application. E.g. Web applications



A three-tier application using an XML Web service is appropriate for either a Web-based or Microsoft Windows® application. This technique comes in handy when you need the richness

of a desktop application, but users connect to it from many different locations and access the data across an HTTP interface.

References:

1. http://msdn.microsoft.com/en-us/library/ms973829.aspx#designnetapp_topic1

Result/output:

Code for Wrapping & Unwrapping:

WrappingUnwrapping.java

```
1  /**
4
5  /**
6   * @author Umair Khan
7   *
8   */
9  public class WrappingUnwrapping {
10     public static void main(String[] args)
11     {
12      char ch = 'a'; // char data type.
13      Character chrobj = new Character(ch);  // Wrapping char type value into Character object.
14
15      byte a = 10; // byte data type value.
16      Byte byteobj = new Byte(a); // Wrapping byte type value into Byte object.
17
18      int b = 20; // int type value.
19      Integer intobj = new Integer(b); // Wrapping int type value into Integer object.
20
21      float c = 18.6f; // float type value.
22      Float floatobj = new Float(c); // Wrapping float type value into Float object.
23
24      double d = 250.5; // double data type value.
25      Double doubleobj = new Double(d); // Wrapping double data type value into Double object.
26
27     // Displaying the values from wrapper class objects.
28       System.out.println("Displaying values of Wrapper class objects:");
29       System.out.println("Character object:  " + chrobj);
30
31       System.out.println("Byte object:  " + byteobj);
32       System.out.println("Integer object:  " + intobj);
33
34       System.out.println("Float object:  " + floatobj);
35       System.out.println("Double object:  " + doubleobj);
36
37          System.out.println("\n");
```

Line: 55

```
        System.out.println("\n");
    // Retrieving primitive data type values from objects.
    // Unwrapping objects to primitive data type values.
        char chr = chrobj;
        byte by = byteobj;
        int in = intobj;
        float fl = floatobj;
        double db = doubleobj;

    // Displaying the values of data types.
      System.out.println("Displaying unwrapped values: ");

      System.out.println("char value: " + chr);
      System.out.println("byte value: " + by);

      System.out.println("int value: " + in);
      System.out.println("float value: " + fl);
      System.out.println("double value: " + db);
      }
}
```

Output:

```
Problems  Javadoc  Declaration  Console
<terminated> WrappingUnwrapping [Java Application] C:\Program Files\Java\jdk-18.0.2\bin\javaw.exe  (Oct 17, 2022, 12:19:20 AM – 12:19:21 AM) [pid: 24472]
================Wrapper Class Experiment By BECOMPA69&66 ==========================
Displaying values of Wrapper class objects:
Character object:  a
Byte object:  10
Integer object:  20
Float object:  18.6
Double object:  250.5


Displaying unwrapped values:
char value: a
byte value: 10
int value: 20
float value: 18.6
double value: 250.5
```

Learning Outcomes: The student should have the ability to

LO1: understand wrappers

LO2: write wrappers to connect application

Course Outcomes: Upon completion of the course students will be able to develop wrappers to connect application

Conclusion:……
…

For Faculty Use

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

EXPERIMENT NO. 07: Design requirements for project - functional requirements are to be added

AIM: Specify the Functional requirements for the project

THEORY:

A Domain-Specific Software Architecture (DSSA) has been defined as:

• "an assemblage of software components, specialized for a particular type of task (domain), generalized for effective use across that domain, composed in a standardized structure (topology) effective for building successful applications"

• "a context for patterns of problem elements, solution elements, and situations that define mappings between them "

The first section describes the domain model 3 that was generated based on scenarios or "operational flows" that reflect the behavior of applications in the domain being analyzed - ticket sales. The domain model consists of:

1. scenarios,
2. domain dictionary,
3. context (block) diagram,
4. entity/relationship diagrams,
5. data flow models,
6. state transition models, and
7. object model.

The second section focuses on the reference requirements. Besides specifying the functional requirements identified in the domain model, the reference requirements also contain:

1. non-functional requirements,
2. design requirements, and
3. implementation requirements.

The third section describes the resulting reference architecture consisting of:

1. reference architecture model,
2. configuration decision tree,
3. architecture schema or design record,
4. reference architecture dependency diagram (topology),
5. component interface descriptions,
6. constraints, and
7. rationale.

The final section provides an analysis of differences between "real world" problems and this "toy" example.

DOMAIN MODEL

One of the insights to be gained from this example is the separation of "problem space" from "solution space" or "design space." The domain model generally tries to characterize fully the former, while the reference architecture addresses a portion (for reasons of practicality) of the latter. Every DSSA starts with an analysis of the application domain. This domain analysis process often involves several domain "experts" who are intimately familiar with legacy systems

of this kind or other aspects of the domain of interest. It also may involve customer inputs as well as inputs from others familiar with various aspects of the application.The purpose of a domain model is to provide to individuals who will develop or maintain applications in a domain an unambiguous understanding of various aspects of the domain.

One important difference between DSSA requirements analysis and traditional systems requirements analysis is the emphasis on the separation of functional requirements from design and implementation requirements. In the customer's mind, these are all "requirements."But from the DSSA perspective, the functional requirements define the (problem) domain, while the design and implementation requirements constrain the design/architecture.

SCENARIOS

The following scenarios consist of a list of numbered, labeled scenario steps or events followed by a brief description.

Ticket Purchase Scenario

1. Ask: The customer asks the agent what seats are available.

2. Look: The agent enters the appropriate command into his/her terminal and relates the results to the customer (cost, section, row number, and seat number).

3. Decide: The customer decides what seats are desired, if any, and tells the agent.

4. Buy: The customer pays the agent for the tickets.Agent gives the tickets to the customer.

5. Update: Tile agent records the transaction.

Ticket Return Scenario

1. Return: The customer gives the agent tickets that are no longer needed.

2. Refund: The agent gives the customer money back.

3. Update: The agent records the transaction.

Ticket Exchange Scenario

1. Ask: The customer asks the agent what seats are available.

2. Look: The agent enters the appropriate command into his/her terminal and relates the results to the customer (cost, section, row number, and seat number).

3. Decide: The customer decides what seats are desired, if any, and tells the agent.

4. Exchange: The customer gives the agent the old tickets, then the agent gives the customer the new tickets. Depending on the price of the new tickets, the agent either collects additional money from the customer or issues a refund.

5. Update: The agent records the transaction.

Ticket Sales Analysis Scenario

1. Stop Sales: The sales manager enters the command to stop the sale of tickets for a particular performance.

2. Tally: The ticket sales program generates a report listing total sales.

Theater Configuration Scenario

1. Performance Logistics: The sales manager enters in the name, time, location, and date of the performance. The following scenarios consist of a list of numbered, labeled scenario steps or events followed by a brief description.

2. Seating Arrangement: The sales manager decides if the performance is "Reserved Seating" or "Open Seating."

3. Theater Logistics: If this performance is reserved seating, then the sales manager enters the number and kind of sections in the theater, what rows are in what sections, and what seats are in

what rows. If this performance is open seating, then the sales manager enters the total number of tickets to be sold.

4. Pricing: The sales manager enters in the price of each ticket, determined by section and seating style. Scenarios are not only a good way of eliciting functional requirements, data flow, and control flow information from a customer but they also allow the analyst to get an idea of what kind of "look and feel" the system should have.

System's requirement

Functional Requirement Id	Requirement Name	Details of requirement
FR01	Authentication	The system will provide the functionality to all the users once they log onto the system with their username and password. Based on their levels, they will be directed to different pages or section
FR02	Login declined	If the user enters the wrong username and password, it will show the error message.

FR03	Admin changes	Once admin login to the system, they can easily see everything about employees and employers, and they can make changes as required
FR04	Job listing	Employers can post jobs and specific details
FR05	Registration for Job	A job seeker can register online to apply for the job or as an employee
FR06	Addition of employee to the list	Admin will have the ability to add job seeker to the employee list
FR07	Reset password	Due to privacy concerns any registered user in the system can change their login password
FR08	Upload CV	Jobseeker can upload their CV including cover letter to apply for a job
FR09	Download CV	Admin can download the CV of the applicant from the system

Learning Outcomes: The student should have the ability to

LO1: understand domains

LO2: write architecture for any domains

Course Outcomes: Upon completion of the course students will be able to develop architectures for specific domains

Conclusion:……

For Faculty Use

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

Experiment 8: Identifying System requirements for an Architecture for any specific domain.

Learning Objective: Students will be able to List various hardware and software requirements, Distinguish between functional and non-functional requirements, indicate the order of priority for various requirements, analyze the requirements for feasibility. Student should be able to understand System requirements for an Architecture for any specific domain.

Tools: IEEE template and MS Word

Theory:

The purpose of a requirements architecture is to structure and organize requirements in such a way that the requirements are stable, usable, adapt to changes, and are elegant. When a requirements architecture is sound, it helps facilitate better design of the system it attempts to describe. When a requirements architecture is faulty, it can cause problems. When the requirements architecture is poor, the following problems result:

No one knows why a requirement was changed

ii. Requirements cannot be reused

iii. Traceability is superficial or unused by other teams

iv. Requirements reviews involve irrelevant information

v. Big picture of the system being built and reasons for building it are not well-understood

vi. It is important to keep in mind that the purpose of a good requirements architecture is to build working software that meets business objectives.

Software requirements must be testable, unambiguous, and concise, a requirements architecture must also possess certain attributes. The above blueprint provides some general guidelines for how to structure requirements, but keeping in mind the following attributes:

a. Maintainable: Whatever choices you make in organizing requirements, ensure that you create a structure that can adapt to changes in requirements.

b. Traceable: Do you know which requirements any given process flow step is traced to?

c. Usable: Consider the stakeholders in the org chart—are the requirements architected in such a way that you could either produce output for each of them or such that they could navigate to the requirements in the tool and find the requirements objects that are relevant to

them? The hierarchies and traces you create should be consistent: Don't create one hierarchy where the FRs are children of the models and another hierarchy for the same project where FRs are not children of the models but are traced to them. The absolute worst thing to do is to list all requirements objects in a flat list or to manage your requirements in word or excel.

d. Scalable: Imagine your requirements architecture with 10 times the number of requirements it has. Now imagine it with 100 times the number of requirements. Architectures should be able to support the addition of new requirements with minimal overhead.

e. Elegant: Are there just enough hierarchies to facilitate use? Are you repeating hierarchies just to make traceability easier? Does your architecture contain duplicate models or requirements?

f. Generalizable: The architecture approach should be repeatable. You ought to be able to go into any project and no matter the domain uses the same approach to requirements architecture.

All architectures are tradeoffs – like in software architecture, you may need to sometimes sacrifice aesthetics for robust traceability or reuse. Or you may sacrifice usability for ease of exporting to external formats. Understand the tradeoffs you are making with your requirements architecture.

All architectures are tradeoffs – like in software architecture, you may need to sometimes sacrifice aesthetics for robust traceability or reuse. Or you may sacrifice usability for ease of exporting to external formats. Understand the tradeoffs you are making with your requirements architecture.

Domain-specific Software Development Various mechanisms of domain-specific software development are under investigation within the projects. Compositional mechanisms facilitate reuse of existing artifacts, including software. Generative mechanisms are used when needed components are not available.1 Constraint-based reasoning systems and module interconnection languages are critical underlying technologies for software composition. Prototyping technologies underlie generation. The TRW project serves as a technology conduit from the prototyping community into the DSSA program.

Following are the system requirement of Domain-specific Software Development:

- Performance
  - How quickly must the system respond to interactive operations of different kinds?
  - Are there different classes of interactive operations that users have different tolerances / expectations for?
  - Is there a batch window? What runs in it?

- Do the batches have their own performance constraints, e.g., to clear the batch windowbefore it closes?
- Does the batch load influence any interactive users running at the same time?
- What is the trade-off between lower averages and wider variations in response time?

- Interoperability
  - What systems will this system interoperate with immediately?
  - What other systems are anticipated?
  - What classes of internal and external systems might later be needed to interoperatewith?
  - What functionality from this system needs to be exposed as a service in a service-oriented architecture?
  - What functionality from this system needs to be exposed as a Web service or via aportal?

- Upgradeability
  - Do the servers need to be upgraded while running?
  - How many client stations need to be upgraded, and what are the costs and mechanismsfor upgrading them? o
  - How often do different kind of fixes need to be distributed? Are there "hot fixes" thathave to go out right away, but others that can wait? How often do each kind occur?

- Auditability / traceability
  - What record of who did what when must be maintained?
  - For how long?
  - Who accesses the audit trails?
  - How? o Is archive to tape or other off-site storage media required?
  - Is "effective dating" required?

- Transactionality
  - What are the important database and application transaction boundaries?
  - Is standard "optimistic" locking appropriate, or is something more complex requiredin some or all cases>
  - Is disconnected operation required by any node?

- Administrability
  - What live usage information needs to be displayed?
  - To who? How? When?
  - What "live" interventions are required?
  - What ability to handle remote configurations are required? O
  - Are there existing application management consoles that will be used to manage thisapplication?

Software Requirements:
Windows: 7 or newer
MAC: OS X v10.7 or higher
Linux: Ubuntu

Software Interface:
I-Front End Client: Html-I-Web Server:
WASCE-I-Data Base
Server: DB2-|-Back End: Java2.

Hardware Requirements:
We strongly recommend a computer fewer than 5 years old.

Processor: Minimum 1 GHz; Recommended 2GHz or more
Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
Hard Drive: Minimum 32 GB; Recommended 64 GB or more
Memory (RAM): Minimum 1 GB; Recommended 4 GB or above
Sound card w/speakers
Some classes require a camera and microphone
Recommended Software

Supported Browsers
People often ask what browser they should use. There is no single answer for this. Use whichever browser works best on your computer. However, we recommend downloading Firefox and/or Chrome in addition to having Internet Explorer or Safari.

Firefox
Chrome

Result and Discussion:

In this Experiment, we have created an SRS document for Job Board App. It specifies the System requirements for an architecture for Job Board App.

Learning Outcomes: Students should have the ability to

LO1: List various hardware and software requirements
LO2: Distinguish between functional and non-functional requirements
LO3: Indicate the order of priority for various requirements
LO4: Analyze the requirements for feasibility Course

Course Outcomes: Upon completion of the course students will be able to prepare an SRS document

Conclusion: ……

For Faculty Use

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

Experiment No. 9: Specify the Non-functional requirements for the project

Learning Objective: Students will be able to List various hardware and software requirements, Distinguish between functional and non-functional requirements, indicate the order of priority for various requirements, analyze the requirements for feasibility. Student should be able to understand System requirements for an Architecture for any specific domain.

Tools: IEEE template and MS Word

Theory:

The purpose of a requirements architecture is to structure and organize requirements in such a way that the requirements are stable, usable, adapt to changes, and are elegant. When a requirements architecture is sound, it helps facilitate better design of the system it attempts to describe. When a requirements architecture is faulty, it can cause problems. When the requirements architecture is poor, the following problems result:

No one knows why a requirement was changed

ii. Requirements cannot be reused

iii. Traceability is superficial or unused by other teams

iv. Requirements reviews involve irrelevant information

v. Big picture of the system being built and reasons for building it are not well-understood

vi. It is important to keep in mind that the purpose of a good requirements architecture is to build working software that meets business objectives.

Software requirements must be testable, unambiguous, and concise, a requirements architecture must also possess certain attributes. The above blueprint provides some general guidelines for how to structure requirements, but keeping in mind the following attributes:

a. Maintainable: Whatever choices you make in organizing requirements, ensure that you create a structure that can adapt to changes in requirements.

b. Traceable: Do you know which requirements any given process flow step is traced to?

c. Usable: Consider the stakeholders in the org chart—are the requirements architected in such a way that you could either produce output for each of them or such that they could navigate to the requirements in the tool and find the requirements objects that are relevant to

them? The hierarchies and traces you create should be consistent: Don't create one hierarchy where the FRs are children of the models and another hierarchy for the same project where FRs are not children of the models but are traced to them. The absolute worst thing to do is to list all requirements objects in a flat list or to manage your requirements in word or excel.

d. Scalable: Imagine your requirements architecture with 10 times the number of requirements it has. Now imagine it with 100 times the number of requirements. Architectures should be able to support the addition of new requirements with minimal overhead.

e. Elegant: Are there just enough hierarchies to facilitate use? Are you repeating hierarchies just to make traceability easier? Does your architecture contain duplicate models or requirements?

f. Generalizable: The architecture approach should be repeatable. You ought to be able to go into any project and no matter the domain uses the same approach to requirements architecture.

All architectures are tradeoffs – like in software architecture, you may need to sometimes sacrifice aesthetics for robust traceability or reuse. Or you may sacrifice usability for ease of exporting to external formats. Understand the tradeoffs you are making with your requirements architecture.

Non-Functional Requirements

1. **Performance Requirement**

- The system should respond fast to any request the user made
- The system should protect individual data & keep it secure
- It must be Portable
- It should be Highly compatible with the user system
- The system must be Reliable Failure rate should be close to zero

2. **Safety Requirement**

- All system data is backed up at regular intervals and the backup is stored on different servers for disaster recovery.
- The admin access is restricted to the college network.
- The data is always logged also to recover from the loss of data

3. Security Requirement

- The access to the administrators is restricted to authorized personnel.
- The administrator accounts are only accessible from the college network.
- The authentication of the user is done by email verification at sign-up.
- Computerized login is prevented using a capuche code for login.

4. Software Quality Attribute

- The system should give the correct results consistently. Product reliability is measured in terms of working of the project under different working environments and different conditions
- Maintenance should be cost-effective and easy. The system is easy to maintain and correct defects or make a change in the software.
- Easy to use for input preparation, operation, and interpretation of the output.
- Provide consistent user interface standards and conventions with our other frequently used systems.
- Easy for new or infrequent users to learn to use the system.

5. Business Rule

- to trigger specific workflows
- to default values
- to validate data entry
- to calculate values
- to propagate data from other fields/tables
- to raise warnings or errors and so on

Learning Outcomes: Students should have the ability to

LO1: List various hardware and software requirements
LO2: Distinguish between functional and non-functional requirements
LO3: Indicate the order of priority for various requirements
LO4: Analyze the requirements for the feasibility Course

Course Outcomes: Upon completion of the course students will be able to prepare an SRS document

Conclusion:

For Faculty Use

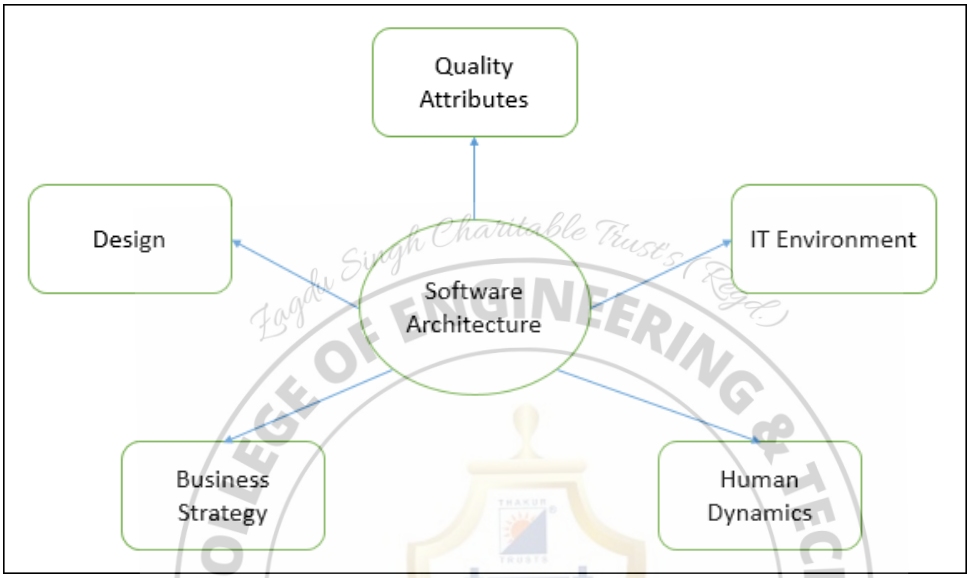
Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment No. 10: Software architecture - component diagram for the project

AIM: Software architecture - component diagram for the project

THEORY:

Software Architecture



Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

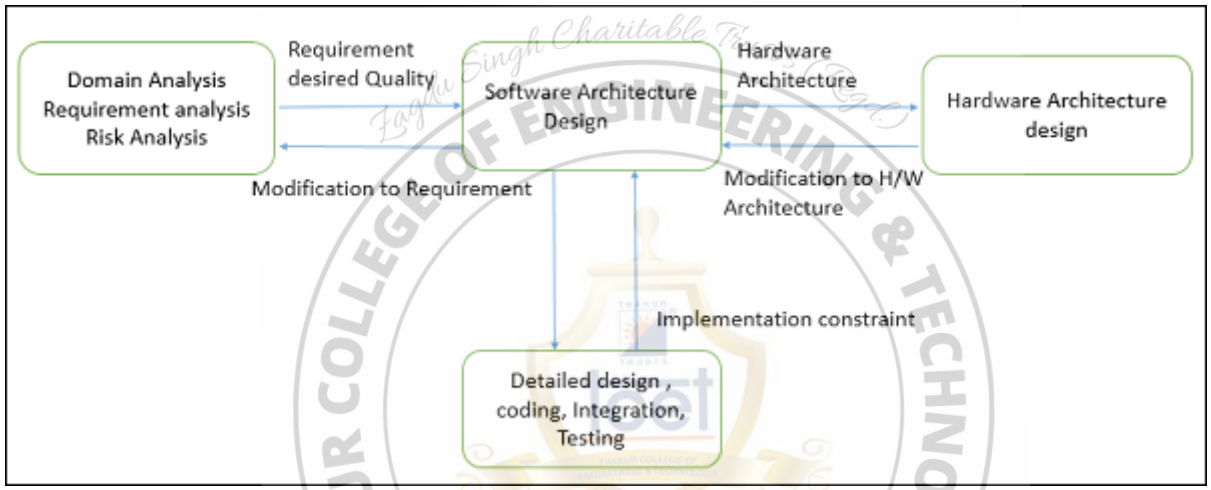
- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −
    - Selection of structural elements and their interfaces by which the system is composed.
    - Behavior as specified in collaborations among those elements.
    - Composition of these structural and behavioral elements into large subsystem.
    - Architectural decisions align with business objectives.
    - Architectural styles guide the organization.

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.



Goals of Architecture

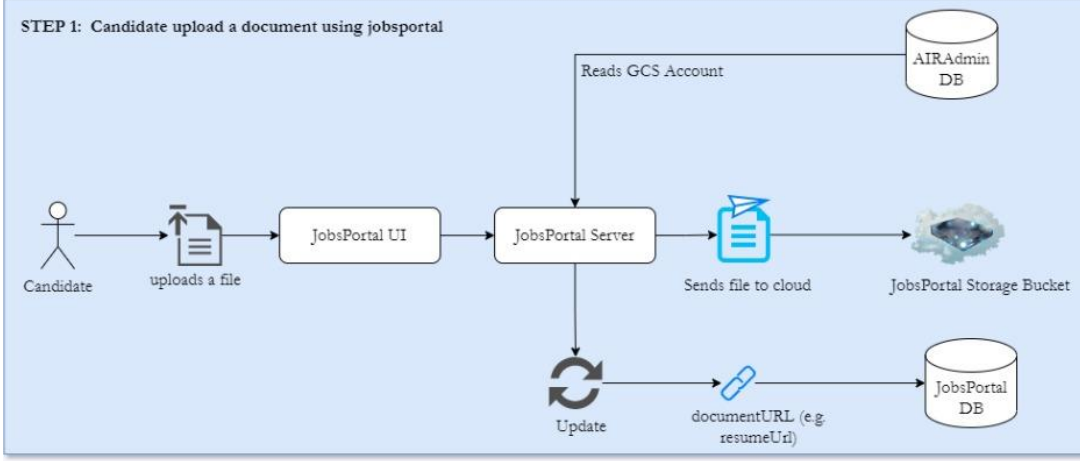
The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows −

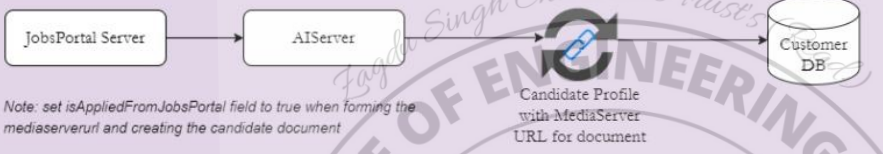
- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization’s market position.
- Improve quality and functionality offered by the system.

Accessing Files between JobsPortal and AIMediaServer components

STEP 1: Candidate upload a document using jobsportal

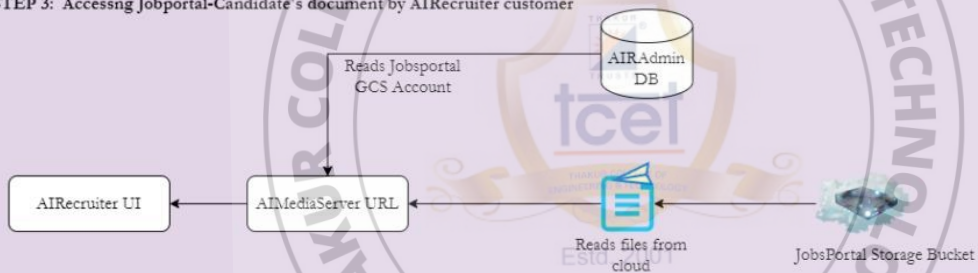


STEP 2: Candidate applying for a job (published by AIRecruiter customer)

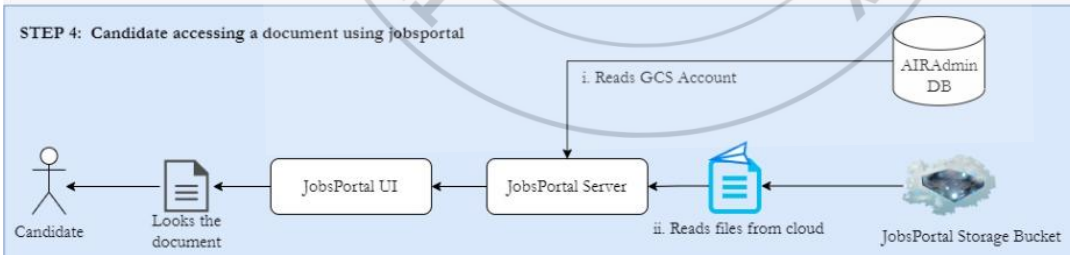


Note: set isAppliedFromJobsPortal field to true when forming the mediaserverurl and creating the candidate document

STEP 3: Accessng Jobportal-Candidate's document by AIRecruiter customer



STEP 4: Candidate accessing a document using jobsportal



Result and Discussion:

In this Experiment, we have created an SRS document for Job Board App. It specifies the System requirements foran architecture for Job Board App.

Learning Outcomes: Students should have the ability to

LO1: List various hardware and software requirements

LO2: Distinguish between functional and non functional requirements

LO3: Indicate the order of priority for various requirements

LO4: Analyze the requirements for feasibility Course

Course Outcomes: Upon completion of the course students will be able to prepare an SRS document

Conclusion:

……

For Faculty Use

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