**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](https://digitalthinkerhelp.com/what-is-operating-system-and-its-types-uses-tutorial/) 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 crucial tool in building software connectors, as it can bridge thread, process, and network boundaries, provide pre-built protocols for data exchange, and include features like filtering, routing, and broadcasting of messages or data. Signal interactions involve one-way interactions between initiating and responding objects, while operation interactions involve interrogations or announcements. Flow interactions are ordered sets of one-way communications from producer to consumer objects. These interactions are a generalized metamodel for communication styles between objects and can be implemented using middleware like Remote Procedure Call (RPC) and Remote Method Invocation (RMI) and message queues. Connectors are the primary vehicle for interprocess communication, and a single connector can be broken up vertically or horizontally for this purpose.

Vertical Connectors :

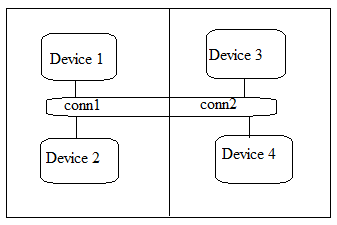


Figure (a)

Horizontal Connectors :

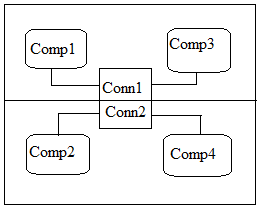


Figure (b)

**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 singleprocess 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 :**

The authors developed two connector-based approaches to implement a multi-process application architecture. The first approach, called "lateral welding," involves implementing a single conceptual software connector using multiple actual connectors linked across process or network boundaries. Each connector becomes a segment of a single "virtual connector," encapsulating all access to the underlying middleware technology within the connector abstraction. This allows for vertical and horizontal slice of an application, with messages sent to each segment filtering and forwarding it to components in its process. The second approach, "horizontally slice," breaks a conceptual connector into top and bottom segments, each with the same properties as a single-process connector to components attached above and below it. These approaches can be used with middleware technology that supports dynamic change at run-time to build applications where processes can join and leave a running application.

Using Middleware Technologies :

The study explores the use of OTS middleware with software connectors using four technologies: Q, Polylith, RMI, and ILU. The researchers integrated two middleware technologies simultaneously in the same application, encapsulating middleware functionality within connectors. This allows architects and developers to use middleware-enhanced connectors like normal, in-process C2 connectors, enhancing the overall functionality of the system.

**Simultaneous Use of Multiple Middleware Packages :**

Middleware technologies offer unique benefits when combined in a single application. Combining multiple middleware technologies within software connectors can widen the range of applications that can be implemented within an architectural style like C2. The KLAX application, a real-time video game application, was used to combine ILU-C2 and RMI-C2 connectors without modification to the middleware-enhanced C2 framework or connectors. 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 supports both ILU and RMI, but this would require changes to the framework.

**Result and Discussion:**  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.

The following topics describe how to perform this task:

1. Building and Deploying a Web Service Created From Java
2. Building and Deploying a Web Service Created From WSDL
3. Deploying the Web Service to a Web Container
4. Verifying Deployment
5. Building and Deploying a Web Service Created From Java

To build and deploy the web service, open a terminal window, go to the *<INSTALL>*/wsit-enabled-fromjava/ directory and type the following:

ant server

This command calls the server target in build.xml, which builds and packages the application into a WAR file, wsit-enabled-fromjava.war, and places it in the wsit-enabled-fromjava/build/war directory. The ant server command also deploys the WAR file to the web container.

The ant command calls multiple tools to build and deploy the web service. The JAX-WS annotation processing tool (apt) processes the annotated source code and invokes the compiler itself, resulting in the class files for each of the Java source files. In the wsit-enabled-fromjava example, the ant target build-server-java in build.xml handles this portion of the process. Next, the individual class files are bundled together along with the web service's supporting configuration files into the application's WAR file. It is this file that is deployed to the web container by the deploy target.

During execution of the server target, you will see a warning message. The message refers to "Annotation types without processors". The warning is expected and does not indicate an abnormal situation. The text is included here for reference:

build-server-java:

  [apt] warning: Annotation types without processors:

    [javax.xml.bind.annotation.XmlRootElement,

     javax.xml.bind.annotation.XmlAccessorType,

     javax.xml.bind.annotation.XmlType,

     javax.xml.bind.annotation.XmlElement]

  [apt] 1 warning

1. Building and Deploying a Web Service Created From WSDL

To build and deploy the web service, open a terminal window, go to the *<INSTALL>*/wsit-enabled-fromjava/ directory, and type the following:

ant server

This command calls wsimport, which takes the WSDL description and generates a corresponding Java interface and other supporting classes. Then the Java compiler is called to compile both the user's code and the generated code. Finally, the class files are bundled together into the WAR file. To see the details of how this is done, see the build-server-wsdl and create-war targets in the wsit-enabled-fromwsdl/build.xml file.

1. Deploying the Web Service to a Web Container

**Deploying to GlassFish**

For development purposes, the easiest way to deploy is to use the autodeploy facility of the GlassFish application server. To do so, you simply copy your application's WAR file to the /autodeploy directory for the domain to which you want to deploy. If you are using the default domain, domain1, which is set up by the GlassFish installation process, the appropriate directory path would be <*AS\_HOME*>/domains/domain1/autodeploy.

The build.xml file which accompanies this example has a deploy target for GlassFish. To invoke that target, run the following command in the top-level directory of the respective examples, either wsit-enabled-fromjava or wsit-enabled-fromwsdl, as follows.

ant deploy

**Deploying to Apache Tomcat**

Apache Tomcat also has an autoDeploy feature that is enabled by Tomcat's out-of-the-box configuration settings. If you are not sure whether the autoDeploy is enabled, check <*TOMCAT\_HOME*>/conf/server.xml for the value of autoDeploy. Assuming autoDeploy is enabled, you simply copy your application's WAR file to the <*TOMCAT\_HOME*>/webapps directory

The build.xml file which accompanies this example has a deploy target for Tomcat. To invoke that target, run the following command in the top-level directory of the respective examples, either wsit-enabled-fromjava or wsit-enabled-fromwsdl, as follows. You need to use the -Duse.tomcat=true switch to make sure that the application is deployed to Tomcat, and not to the default server, which is GlassFish.

ant -Duse.tomcat=true deploy

1. Verifying Deployment

A basic test to verify that the application has deployed properly is to use a web browser to retrieve the application's WSDL from its hosting web container. The following URLs retrieve the WSDL from each of the two example services. If you are running your web browser and web container on different machines, you need to replace localhost with the name of the machine hosting your web service.

**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:** Implementing connectors using middleware offers a versatile and efficient approach to building software systems. It simplifies connectivity, data transformation, and communication between components while enhancing security and scalability. Middleware provides a valuable abstraction layer that promotes interoperability and future-proofing in complex software architectures.

For Faculty Use

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