

Experiment No.: 5

Use middleware to implement connectors

37_BE-COMP-B_Mihir-Gharat

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

Theory:

What is Middleware ?

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

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

Role of Middleware is :-

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

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

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

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

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

Vertical Connectors :

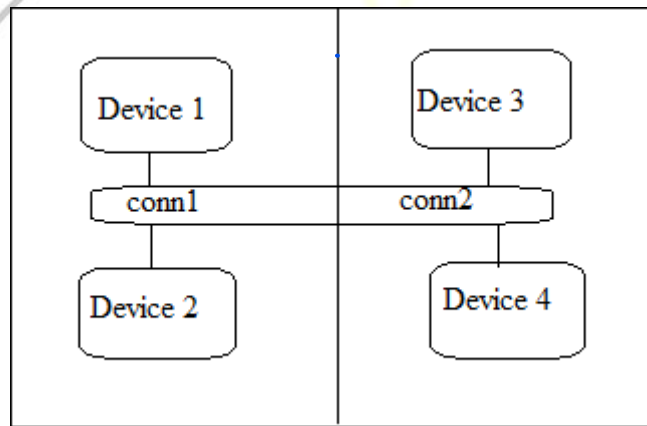


Figure (a)

Horizontal Connectors :

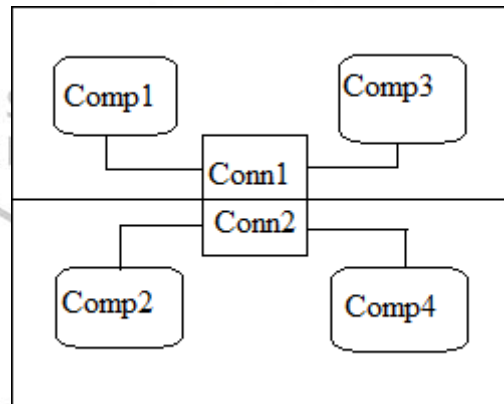


Figure (b)

Linking Ports Across Process Boundaries :

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

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

Linking Connectors Across Process Boundaries :

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

Using Middleware Technologies :

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

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

Simultaneous Use of Multiple Middleware Packages :

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

OUTPUTS:

Estd. 2001

ISO 9001 : 2015 Certified
NBA and NAAC Accredited

eclipse-workspace - middleware/src/middleware/Server.java - Eclipse IDE

```

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer
  CalculatorMul
  CalculatorMulClient
  CalculatorMulEAR
  CalculatorProject
  CalculatorProjectClient
  CalculatorProjectEAR
  middleware
    JRE System Library [JavaSE-17]
    src
      middleware
        Client.java
        interfaceCalculator.java
        Server.java
        module-info.java
  Servers

Server.java
1 package middleware;
2
3 public class Server implements interfaceCalculator {
4     public int add(int a, int b) {
5         return a+b;
6     }
7     public int sub(int a, int b) {
8         return a-b;
9     }
10    public int mul(int a, int b) {
11        return a*b;
12    }
13    public int div(int a, int b) {
14        return a/b;
15    }
16    public String uppercase(String s) {
17        return s.toUpperCase();
18    }
19 }
20
21

Outline
  middleware
    Server
      add(int, int) : int
      sub(int, int) : int
      mul(int, int) : int
      div(int, int) : int
      uppercase(String) : String

Problems Javadoc Declaration Console
<terminated> Client [Java Application] C:\Users\lab314-4\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (01-Aug-2022, 1:23:00 pm - 1:23:00 pm) [pid: 8716]
332
92
25440
1
GANESH
VISHAL
SIDHESH

Type here to search 30°C Light rain 19:5:35 13:23 01-08-2022

```

eclipse-workspace - middleware/src/middleware/interfaceCalculator.java - Eclipse IDE

```

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer
  CalculatorMul
  CalculatorMulClient
  CalculatorMulEAR
  CalculatorProject
  CalculatorProjectClient
  CalculatorProjectEAR
  middleware
    JRE System Library [JavaSE-17]
    src
      middleware
        Client.java
        interfaceCalculator.java
        Server.java
        module-info.java
  Servers

interfaceCalculator.java
1 package middleware;
2
3 public interface interfaceCalculator {
4     public int add(int a, int b);
5     public int sub(int a, int b);
6     public int mul(int a, int b);
7     public int div(int a, int b);
8     public String uppercase(String s);
9 }
10

Outline
  middleware
    interfaceCalculator
      add(int, int) : int
      sub(int, int) : int
      mul(int, int) : int
      div(int, int) : int
      uppercase(String) : String

Problems Javadoc Declaration Console
<terminated> Client [Java Application] C:\Users\lab314-4\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (01-Aug-2022, 1:23:00 pm - 1:23:00 pm) [pid: 8716]
332
92
25440
1
GANESH
VISHAL
SIDHESH

Type here to search 30°C Light rain 8:39:226 13:24 01-08-2022

```

eclipse-workspace - middleware/src/middleware/Client.java - Eclipse IDE

```

1 package middleware;
2
3 public class Client {
4     public static void main(String[] args) {
5         interfaceCalculator c = new Server();
6         System.out.println(c.add(212, 120));
7         System.out.println(c.sub(212, 120));
8         System.out.println(c.mul(212, 120));
9         System.out.println(c.div(212, 120));
10        System.out.println(c.uppercase("ganesh"));
11        System.out.println(c.uppercase("vishal"));
12        System.out.println(c.uppercase("siddhes"));
13    }
14 }
15

```

Problems Javadoc Declaration Console X

```

<terminated> Client [Java Application] C:\Users\lab314-4\p2\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (01-Aug-2022, 1:23:00 pm - 1:23:00 pm) [pid: 8716]
332
92
25440
1
GANESH
VISHAL
SIDDHES

```

Writable Smart Insert 12:49:425 30°C Light rain 01-08-2022

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

```

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

```

Markers Properties Servers Data Source Explorer Snippets Terminal Console X

```

<terminated> ClientClass [Java Application] C:\Users\lab314-2\p2\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe
20

```

Writable Smart Insert 4:7:6

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

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

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

```

1 public interface InterfaceClass{
2
3
4     public int add(int a,int b);
5     public int sub(int a,int b);
6     public int mul(int a,int b);
7     public float div(int a,int b);
8 }
9

```

Outline

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

Markers Properties Servers Data Source Explorer Snippets Terminal Console

<terminated> ClientClass [Java Application] C:\Users\lab314-2\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaz 20

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

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

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

```

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

```

Outline

- ClientClass
 - main(String[]) : void

Markers Properties Servers Data Source Explorer Snippets Terminal Console

<terminated> ClientClass [Java Application] C:\Users\lab314-2\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaz 20

Result and Discussion:

Learning Outcomes: Students should have be able to

LO1: Define Middleware.

LO2: Identify different connectors in middleware.

LO3: Explain middleware implements in connectors.

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

Conclusion:

Viva Questions:

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

For Faculty Use

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

Lagdu Singh Charitable Trust's (Regd.)

THAKKUR COLLEGE OF ENGINEERING & TECHNOLOGY



Estd. 2001

ISO 9001 : 2015 Certified
NBA and NAAC Accredited

Zagdu Singh Charitable Trust's (Regd.)

THAKKUR COLLEGE OF ENGINEERING & TECHNOLOGY



Estd. 2001

ISO 9001 : 2015 Certified
NBA and NAAC Accredited

Lagdu Singh Charitable Trust's (Regd.)

THAKUR COLLEGE OF ENGINEERING & TECHNOLOGY



Estd. 2001

ISO 9001 : 2015 Certified
NBA and NAAC Accredited