CHAPTER 6

PLATFORM INTEROPERABLE SOLUTION TOOL USING IIOP .NET

6.1 INTRODUCTION

A heterogeneous computing environment is often unavoidable due to the evolving needs of new platforms, new operating systems, new software, and new networking systems. More and more applications written in different programming languages running on different platforms with different operating systems in a heterogeneous computing environment are required to communicate with each other across different computer networks [64][99].

Conflicts resulting from having heterogeneous computing environments require strategies for application integration support. Software interoperability is a fundamental problem to application integration [13]. There are many aspects related to component interoperability, including syntactic agreements on method names, behavioral specifications of components, service access protocols, business domain knowledge and shared ontology, negotiation of Quality of Service and other non-functional properties [58].

Providing type interoperability consists in ensuring that, even if written by different programmers, possibly in different languages and running on different platforms, types that are supposed to represent the same software
module are indeed treated as one single type. This form of interoperability is crucial in modern distributed programming [82].

In Microsoft's vision, the next generation of distributed systems will communicate with WebServices. WebServices are great when it comes to integrate heterogeneous loosely coupled systems, but they have their limitations too: they have no support for remote object references [67]. In practice, they are stateless and closer to a remote method call than to a distributed object system. Furthermore, SOAP and XML are by no means a compressed format and tend to be quite verbose. Due to the drawbacks of SOAP, IIOP is better for the interoperability. Based on IIOP, a remoting channel IIOP .NET is developed, which provides seamless interoperation between .NET, CORBA and J2EE distributed objects. In this work IIOP .NET is used for providing interoperability between J2EE and .NET.

J2EE and .NET

.NET and J2EE are two similar but disjointed worlds: they currently can interact together only using WebServices. Both platforms offer great mechanisms for building tightly coupled distributed object systems: .NET’s Remoting and Java’s RMI, but sadly these rely on incompatible standards.

Enterprise Java Beans (EJBs) is an established technology for implementing software components on a Java platform [66]. Sun MicroSystem’s definition of Enterprise JavaBeans is [75]:

“The Enterprise Java Beans architecture is component architecture for the development and deployment of component-based distributed applications. Applications written using the enterprise Java Beans architecture
is scalable, transactional and multi-user secure. These applications may be written once, and then deployed on any server platform that supports Enterprise JavaBeans specification”.

Many IT key-players are providing application servers for hosting EBJs, in particular IBM's WebSphere, BEA's WebLogic, and the open source JBoss. EBJs can be used to create distributed object systems, and rely on Java RMI/IIOP to exchange messages; EJBs can also be exposed as Web Services. JavaBeans is a reusable component technology. It can be called by Java Applet, Servlet and JSP. It can also access database and run on any platform [100].

Some of the advantages of EJB are :

- **EJB provides developers architectural independence** - EJB insulates developers from the underlying middleware, since the only environment an EJB developer sees is the Java environment.

- **WORA for server side components** - Since EJB is based on Java technology; both the developer and the user are guaranteed that their components are Write Once, Run Anywhere (WORA).

- **EJB establishes Roles for Application Development** - The EJB specification assigns specific roles for project participants charged with enterprise application development utilizing EJB

- **EJB provides Distributed Transaction support** - EJB provides transparency for distributed transactions.
- *It helps create Portable and Scalable solutions* - Beans conforming to the EJB API will install and run in a portable fashion on any EJB server.

- *It provides vendor specific enhancements* - Since the EJB specification provides a lot of flexibility for the vendors to create their own enhancements, the EJB environment turned out to be feature rich.

For a decade, Component Object Model (COM) of Microsoft was the standard for components those run on Windows machines. In July 2000, Microsoft launched the .Net Components model without offering backward compatibility with COM, though this new model inherits COM infrastructure and runtime services called COM+. These services are available starting with the operating system Windows 2000 and are configurable via the “Component Services” which is an administrative tool of the Windows control panel [41].

The .NET and J2EE platforms can tightly interoperate together, as it is often needed when developing distributed enterprise applications [67]. For this purpose, an open-source custom remoting channel called IIOP .NET can be used.

### 6.2 Why IIOP .NET?

IIOP .NET allows a seamless interoperation between .NET, CORBA and J2EE distributed objects. This is done by incorporating CORBA/IIOP support into .NET, leveraging the remoting framework. IIOP .NET was born on May 2 2003, and grew from a small experimental project to a stable and useful application.
Features of IIOP .NET

- **Tight coupling** between distributed objects in .NET, CORBA and J2EE; components on each platform can act in either client or server role.

- **Transparency**: existing servers can be used unmodified, without wrapping code or adapters.

- **Extensive coverage** of CORBA/.NET type mappings.

- **Native integration** in the .NET framework.

Distributed applications require interchange of data to provide integration and coordination between the various pieces. XML, a standard of the World Wide Web Consortium (W3C), is a tag-based language that can be used to describe information exchanged between senders and receivers and offers a standardized approach for defining these integration schemes. XML also provides the ability to express rules governing the structure of the information [2]. IIOP .NET is the best channel for providing interoperability between the .NET and J2EE platforms. At the same time, while using IIOP .NET for providing interoperability, lots of manual processes are involved for the configuration process.

In order to reduce the manual process, an automated design tool called Platform Interoperable Solution (PIS) is developed using IIOP .NET and XML. The Platform Interoperable Solution (PIS) automates the configuration process for providing interoperability between EJB and .NET. The main objective of the proposed PIS is to provide interoperability between EJB and .NET both on Server side and Client side. PIS uses a Java server as a configuration server. In PIS, with the help of configuration server and
XML files, interoperability solution for EJB and .NET has been obtained. And the configuration processes have been stored once into the XML files and can be loaded whenever interoperability is required in PIS.

### 6.3 PLATFORM INTEROPERABLE SOLUTION

Even though the solutions had been given for providing the interoperability between J2EE and .NET using IIOP .NET, it is too complicated to do the configuration process. Providing the interoperability between .NET and J2EE using IIOP .NET involves more time and more manual configuration process. But PIS provides automatic configuration process for interoperability between J2EE and .NET using IIOP. PIS is a machine independent and component independent tool, which can provide interoperability between the components of any vendors and on any machine. PIS can be used for providing the interoperability between .NET and J2EE both on Server side and Client Side, and hence PIS gives both (Server & Client) the sides’ solution. The two kinds of solution had been included in PIS for providing interoperability between EJB and .NET. They are,

- PIS-Server Side Interoperability (PIS-SSI)
- PIS-Client Side interoperability (PIS-CSI)

In PIS, the configuration server namely Java Server does the process of translating the one form of component into the Interface Definition Language (IDL) form, and from IDL to required form of the client. In PIS, with the help of configuration server and XML files, interoperability solution for EJB and .NET has been obtained both on Server Side and Client Side. In PIS, the configuration processes have been stored once into the XML files and can be loaded whenever interoperability is required.
6.4 PIS-SERVER SIDE INTEROPERABILITY

Normally the stubs of .NET and java clients can be allowed to access their components only through .NET and EJB server respectively. EJB and .NET clients cannot directly access each other’s stubs. There is no solution for providing common stub for both EJB and .NET server. Thus we developed PIS with the solution for accessing the .NET component from Java Client and the EJB from .NET Client. The solutions are,

(i) PIS-EJB Server Side Interoperability (PIS-ESSI)
(ii) PIS-.NET Server Side Interoperability (PIS-dotSSI)

6.4.1 PIS-EJB Server Side Interoperability

As stated earlier EJB Server allows stub of java client and it will not accept .NET client. Figure 6.1 shows that, in PIS-ESSI EJB Server allows the stub of java client as well as with the help of configuration server it allows the .NET client. In PIS-ESSI, EJB Server allows the stub of both .NET and Java client.

![Diagram of PIS-EJB Server Side Interoperability]

**Figure 6.1 PIS-EJB Server Side Interoperability**
6.4.2 PIS-.NET Server Side Interoperability

In PIS-.NET Server Side Interoperability, .NET Server allows the stub of .NET client as well as with the help of configuration server it allows the java client as shown in Figure 6.2. Mainly the configuration server performs the job of translating .NET form into IDL, then translates IDL to java form with the help of XML files in which the configuration process have been stored. In PIS-dotSSI, EJB Server allows the stub of both .NET and Java client.

![Figure 6.2 PIS-.NET Server Side Interoperability](image)

6.5 PIS-CLIENT SIDE INTEROPERABILITY

As the component needs to communicate with the client, they can be implemented as an in-process (read DLL) component and an out-process (read EXE) component.
6.5.1 PIS-Client Side Interoperability through Out-Process

Out processes are in the form of (EXEs) Executables. They run in a different memory space as that of the client application (that's why they are termed as out-process). An out-process component is an .exe file that runs in its own process, with its own thread of execution. If they crash, it doesn't affect the client application as they operate in a different memory space. Creating a generic Platform on the Client Side for .Net and EJB stub is the best solution for integrating .NET and EJB.

![Diagram of PIS-Client Side Interoperability](image)

Figure 6.3 PIS-Client Side Interoperability

In PIS-Client Side Interoperability environment, the interoperability is provided on client side itself, through out-process as shown in Fig.6.3. .NET client will convert the stub into Interface Definition Language (IDL), which is language neutral. And from that IDL, the stub for Java client will be created. Similarly in PIS Client Side environment java clients convert their stub into .NET stubs through out-process.

6.5.2 PIS-Client Side Interoperability through In-Process

In an in-process component (ActiveX DLL component), the communication between the server and the client is implemented in the address space of the client application. Though this makes them faster than ActiveX- EXE components, which need to be loaded in their own address
space, the biggest drawback is that a faulty DLL will crash the client, and in turn, the object. An in-process component, such as a .dll or .ocx file, runs in the same process as the client. In PIS Client Side environment, the .NET component can be accessed from a java client by loading CLR into the java client’s address space through in-process as shown in Figure 6.4. Similarly accessing an EJB from .NET client is enabled through PIS-CSI-IP by loading JVM into the .NET client’s address space.

![Diagram](image)

**Figure 6.4 An In-Process PIS-Client Side Interoperability**

Even though the solution had been given for interoperability between EJB and .NET both on server side and client side, Client side interoperability approach is found to be easy, low in cost and quick to implement.

![Diagram](image)

**Figure 6.5 Layout of the configuration process using PIS**
6.6 CONCLUSION

In this chapter, an automated design tool called Platform Interoperable Solution (PIS) tool is described. PIS involves less development and maintenance cost and it needs less manual work. For the components already developed and registered in the server, PIS-CSI is suitable and for the new components PIS-SSI is suitable. Integration of the EJB and .NET components both on server side and client side are achieved through PIS. PIS does not dependent on any vendor like Janeva and J-integra. In PIS, there are no difficulties in classpath setting, while providing interoperability like J-Integra. Using PIS, .NET services can be called from J2EE and vice versa. PIS is viewed as an efficient automated tool for Component Based Software Development.