

# **Component Based Distribution Model**

June 2000

## **Component Based Distributed Model**

Until recently, the focus for developers in distributed computing community was between COM/DCOM from Microsoft, or CORBA standards from OMG. But, with acceptance of server-side Java programming, popularity of Java Application Servers is on rise. The idea is to insulate business logic from underlying distributed object model. In this paper, we examine some of the latest technology invents surrounding Java platform and its acceptance in different school of thoughts.

### **Distributed Object Debate:**

Till mid 1998, it was commonly agreed that between COM/DCOM or CORBA, developers have to choose between one of the two, rendering serious limitations in truly distributing objects. Middleware design had serious limitations, no matter which way you want to distribute the objects. Components will be tied to the underlying operating system.

The debate between COM-versus-CORBA was not paving for a reusable component library development environment. The ground level reality is both the technologies are actually tied to philosophies of their creators. Although both school believed in superiority of one over the other, an abstract view of both made it very clear that a "pure" strategy around COM implies something fundamentally different than a "pure" strategy around CORBA. Industry believes no exclusive strategy is likely to win in practice, illustrating more logical reason for distributed development around Java Application Server framework. We will examine this in-depth in next section.

### **CORBA Scenario:**

The promise of CORBA has been to produce an "intergalactic" standard for distributed object computing, an environment where all objects can live in harmony. A complete utopia for object distribution and communication. In its purest form, CORBA objects are supposed to be platform-neutral, vendor-neutral, and even protocol-neutral. The idea is to allow vendors develop CORBA library and compete in an open market, based on specification. Easily said than done, vendors competing implementations of CORBA make interoperability more difficult than necessary, because building real world applications requires more specificity than the CORBA standards provide. This causes trouble in developer community when they have to choose between OMG specific ORB or vendor specific ORB's (Visigenic or Iona). The tradeoffs are high in terms of functionality and portability.

### **Microsoft Scenario:**

In its own world, Microsoft spins a dramatically different story around COM. Tough to buy, when developer wants to write truly portable/reusable component libraries. Microsoft keeps telling developers how easy it to get the object distribution done in "One Microsoft Way", where they can get rid of complexities surrounding them. Howsoever elegant, but a solution possible only on Microsoft

platform. DNA (Distributed Internet Application) model is a perfect example. In simple visualization, to use COM components introduces dependencies on a host of Windows services, which includes Registry, MTS components, and IIS components. Of course you need extra library components from Visual Studio, SQL server, and Active Directory. Critical server-side components (like MTS) cannot run on non-Microsoft platform.

### **The Java Application Scenario:**

What ever we have learned so far is not to discredit either COM or CORBA, but merely to reiterate the well known limitations of an exclusive approach based on either. With acceptance and addition of functionality's in Java and its API's, a third scenario is emerging in the form of Java-enabled Application Servers.

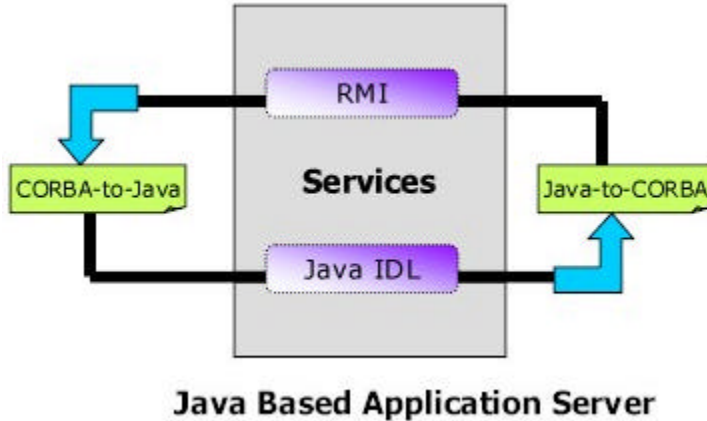
Java application servers unite the technologies of the web, database and networking components. Java application server is functionally a run time environment. It exposes enterprise business logic to set of Java API's to developers. It gives developers controls pertaining to life cycle of application components, and provides services like static management, database connection pooling, and transaction services. Business logic may be organized as a combination of Java Servlets (as an elegant alternative to stateless CGI), Java Beans, and enterprise JavaBeans components.

### **Java Application Server Architecture**

In this paper, it is not possible to cover the complete architecture model of a typical Java Application Server, also known as Java-enabled web application server. We will strictly limit ourselves towards analyzing the current state of distributed model. Java Application server maps remote services and components to a rich local runtime environment in a way that is independent of the underlying middleware and consistent across platform and implementations. Although, the native code of JAS is of course in Java, its possible to map components created with other languages, and component models into the Java environment, and vice-versa.

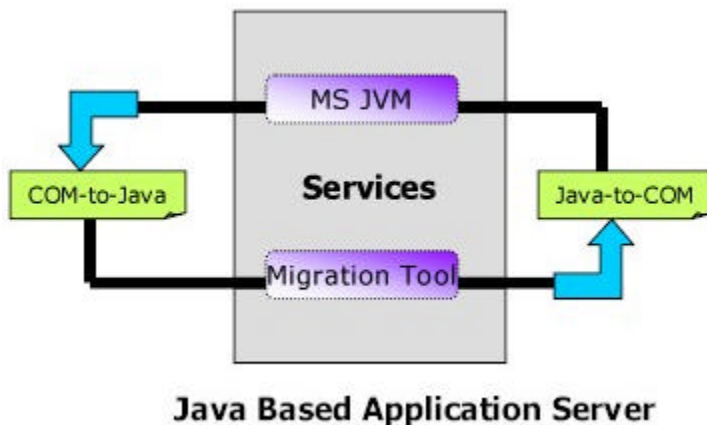
### **?? Mapping between Java and CORBA**

This is pretty straightforward mapping, as Java provides both CORBA-to-Java mapping using Java IDL which ships with Java compiler, and Java-to-CORBA mapping using RMI. Java also provides mapping between various CORBA services, such as, naming and transactions, JNDI and JTS respectively. OMG has also adopted this strategy by incorporating Java RMI protocol (JRMP).



## ?? Mapping between Java and COM

As with CORBA, it is possible to export Java business objects as COM objects or Active-X controls using the Microsoft JVM (JView) and an application server, which can support COM (like WebLogic Tengah). Java Beans and Java objects can be wrapped as COM components that can be imported into Visual Studio. This eliminates the integration barrier between server-side Java and broad array of client environments from web browsers to Java clients to Microsoft desktop. Reverse mapping is also possible by wrapping COM business objects with Java objects, with a limitation that runtime environment of Windows operating systems will become necessary.



## Conclusion

Java application server reduces the complexity of developing enterprise applications. Before the advent of technologies like Enterprise Java Beans, developers had to worry about making server-side business objects persistent, secure, network-aware, shared, and sessioned. Of course issues like scalability and availability were of prime concern. Rich API from Javasoft, which is helping in the

development of Java application servers hide these complexities, so that developer can focus on the business logic.

The idea is to separate business logic layer from middleware. Mixing the two, as is common in traditional client/server programming, complicates application maintenance, portability, and interoperability. With Enterprise JavaBeans and other Java based services, the applications can be developed independently of the underlying middleware.

Along the same lines, IrisLogic has understood the benefits of richness and support of Java API and development in server technology. We have constantly developing and improving our Component Library using Servlets and Java Technology.