Assuring Reliability of Enterprise JavaBean Applications

White Paper

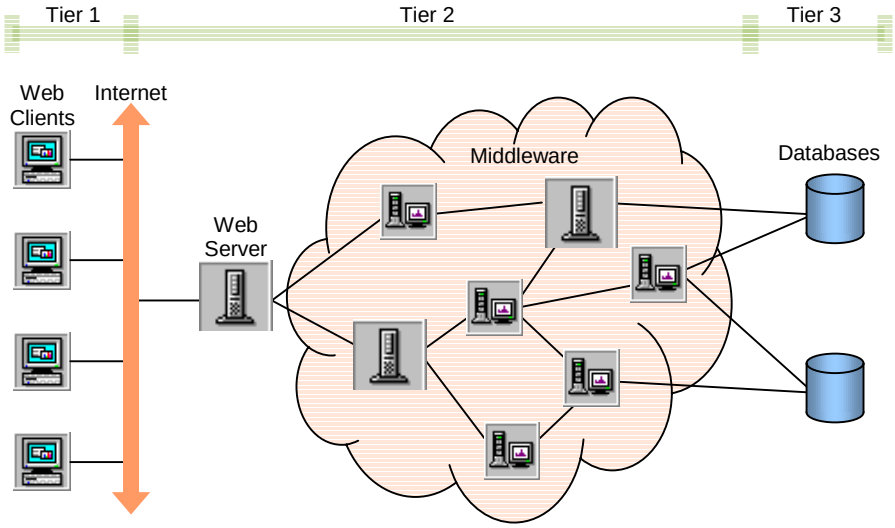
Segue Software

www.segue.com

The *Enterprise JavaBean (EJB)* specification demonstrates the evolution of distributed objects from *middleware* to *application components*. This paper discusses where EJBs fit into the distributed object landscape, and how Segue addresses the reliability of e-business applications built using EJB technology.

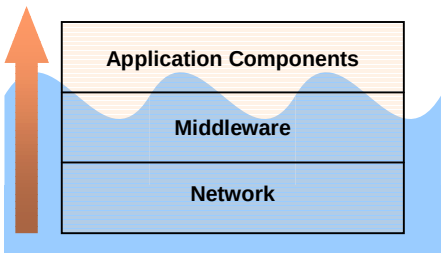
Middleware in the Middle Tier

The three-tier architecture is fundamental to the deployment of applications on an Internet-based infrastructure. A web server fronts the middle tier (Tier 2 in the diagram below), behind which are many systems that comprise the value-add for an e-business application. From a practical perspective, e-business applications are assembled from disparate components, must be accessible to web clients (Tier 1) through the Internet, and reach relevant data repositories (Tier 3) such as financial data or travel reservations.



Middleware is the glue for integrating systems within the middle tier. It plays a vital role in application reliability since integration points among the various systems are potential points of failure. As distributed object technology has become well accepted within mainstream software applications, standards such as CORBA and RMI have emerged as preferred object communication mechanisms. However, from an application perspective these technologies present a rather low abstraction level, leaving the task of defining how components should interoperate to the programmer. This means inventing proprietary programming interfaces that must be adhered to by all parties involved with the creation of the application.

One fact of life concerning technology is that the "waterline" is always rising, making technology more accessible and easier to use. In the Internet era, the waterline corresponds to the higher level of abstraction presented by technologies for assembling web-based applications. More to the point, software frameworks provide a simpler model for creating e-business applications, supporting the development of application components that can easily work together.



A *framework* is a software infrastructure that defines certain rules of engagement among application components. These rules are typically implemented as services with standardized interfaces, allowing components to interoperate without prior knowledge of other components' existence. The ability to cut and paste information between different applications is a classic example. A framework also provides an execution environment for end users and applications. A good example of this is the Microsoft Windows desktop.

Although frameworks and component models offer convenience for quickly assembling applications, only those based on proprietary technologies and captive install bases, such as Microsoft's OLE for desktop integration, have historically enjoyed widespread industry acceptance. But thanks to the Internet, the tide appears to be turning for an open standard framework as the industry embraces EJB.

Relationship of EJB to CORBA and RMI

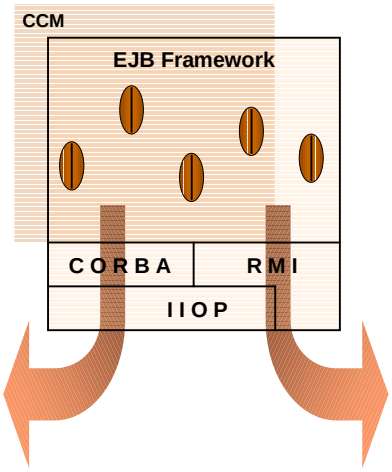
CORBA is an open industry standard that facilitates remote object invocation via published interfaces. However, it does not yet specify a market-accepted component model for plug-and-play interoperability. To illustrate this point, contrast CORBA with its parallel universe, Microsoft:

DCOM is to CORBA as OLE is to _____.

In other words, DCOM and CORBA provide the communication substrate in their respective universes. OLE is the framework that sits on top of DCOM, which you would experience every day as a Windows user. But in the CORBA universe, what is analogous to OLE? What provides the higher abstraction level moving up the waterline? The answer isn't particularly clear yet, but EJB seems to be emerging as a viable solution for e-business applications.

The EJB specification defines an execution and services framework for server-side Java components. *Application servers* house EJB *containers*, which together manage threading, transactions, object lifecycle, and other EJB needs. Commercial products from industry movers like IBM, BEA Systems and Oracle, and increasing adoption by customers illustrate EJB's growing acceptance as a standard component model for Web applications. For an EJB to be accessed from outside the framework, its container must expose a public interface through CORBA or RMI. Since RMI is a Java-only communication mechanism, it can be used over IIOP to speak to non-Java applications.

*The **CORBA** standard defines mechanisms whereby objects implemented in different languages can communicate transparently through method invocations. Server interfaces are described in a neutral language called IDL, which is mapped to native language constructs such as Java, C++, Ada, Cobol and others. IIOP is the underlying wire protocol supporting communications among different ORBs and platforms.*



***RMI** is the Remote Method Invocation facility provided as part of the JDK. Objects can make remote invocations on one another through public interfaces specified in Java, making RMI suitable only for Java programs. However, the JDK now supports RMI over the CORBA wire protocol – IIOP. This allows Java objects to utilize RMI while offering interoperability with CORBA objects implemented in any language.*

It is worth noting that the CORBA Component Model (CCM) defines a framework for application components implemented in any major programming language. CCM extends the EJB model, which will allow Java and other programs to operate on the same CORBA platform.

Reliable EJB

Application servers based on the EJB specification facilitate the creation of web-enabled applications. Such applications operate in the middle tier, and rely on distributed object communication for interoperability. So how do you verify that your EJB implementation will run reliably? The behavior of an EJB server can be validated from outside the framework. Test automation can be employed to exercise functionality and simulate usage models through public interfaces via the CORBA or RMI standards.

SilkPilot from Segue Software allows a user to test the behavior of EJB components through published interfaces. The initial release of SilkPilot supports IDL interfaces and IIOP communication. Additional interface specifications and wire protocols will be added. SilkPilot exploits dynamic invocation capabilities to provide an interactive, graphical user interface representation of object interfaces, which can otherwise only be accessed programmatically. Using SilkPilot, a user connects to one or more servers, views information about live objects within the servers, invokes methods with parameters, and views or modifies attributes. Test cases are generated automatically as Java source code based on object interactions performed during a session. These test cases can then be run to validate that application components are functioning properly.

In addition, Segue's **SilkPerformer** can be used to simulate protocol traffic and measure the capacity and scalability of an application. For example, IIOP communication can be recorded and subsequently replayed. Measurement of an application's performance involves simulation of usage models under various loads. Load testing scripts can be generated from recorded traffic and replayed for multiple virtual clients.

In short, Segue's family of object testing products continues to lead the industry by keeping up with the ever rising technology waterline. As distributed objects evolve from middleware to application components, the need for advanced test automation continues to grow. EJB is an early example of industry consensus around application components, and Segue is right there to assure the reliability of EJB-based applications.