CICS and Enterprise JavaBeans

by A. Bainbridge
J. Colgrave
A. Colyer
G. Normington

IBM is supporting Enterprise JavaBeans™ (EJB) across its application server products. Together with related Java™ technologies, EJB provides a standard programming model and set of services across major server platforms. This paper presents an overview of the EJB architecture and describes the technical strategy and design approach for the EJB capability that is being delivered in the latest release of the Customer Information Control System (CICS®). It explores why CICS customers want to use enterprise Java technology, identifies some critical success factors for the CICS support of EJB, and explains its architecture in terms of a portable EJB container executing within the existing CICS run-time infrastructure, focusing on aspects such as transaction management, Java virtual machine reuse, and workload management. Application development tooling is then discussed, together with the strategy for exploiting tools such as VisualAge® for Java. Finally, the paper considers future work and challenges for CICS in this area.

Over the last few years, the IBM Customer Information Control System (CICS*) product[1] has been transformed. Significant investments have been made to develop it as a high-end server for e-business applications, while maintaining and developing its capabilities as the world's leading transaction processing monitor for customers' mission-critical applications. It has been estimated that, worldwide, over 30 billion transactions are executed *each day* on CICS. Some CICS installations support peak workloads well in excess of a thousand transactions per second; some customers execute over 20 million transactions per day.

Java** technology is now a key element of the CICS strategy, and two of the principal new features in the last release of the product (CICS Transaction Server for OS/390* [Operating System/390] Version 1.3) are support for writing CICS transactions in the Java language and for using Common Object Request Broker Architecture (CORBA**) Internet Inter-Orb Protocol (IIOP)[2] to communicate with client applications. These two capabilities provide the foundation for the next major step in the evolution of the product: the addition of support for Enterprise JavaBeans**.[3]

Application servers and Enterprise JavaBeans

Application servers have existed for a long time. For example, CICS is an application server that has been in the marketplace for over 30 years. The term "application server" can be loosely defined as *an execution environment* together with *a set of run-time services* that are made available to applications written to take advantage of those facilities. Application servers are distinct from, and usually run "on top of," operating systems. They offer a specialized environment with desirable characteristics for the application types that they support. One of the principal reasons for the existence of separate middleware application servers is that they provide levels of performance, scalability, and robustness that are beyond those achievable with a simple operating system environment, enabling many hundreds of thousands of concurrent users to be supported with acceptable

0018-8670/01/$5.00 © 2001 IBM

performance and cost. Application servers typically provide a managed environment in which resources such as files, databases, and queues are readied before any application requests are received, thus minimizing response time, and they support specialized programming models, such as the pseudoconversational transaction model, which minimize contention for resources. With CICS, the execution environment may span multiple operating system address spaces and even multiple operating system images.

Recently, we have seen the emergence of the "Web application server" category. Web application servers typically provide HTTP (HyperText Transfer Protocol) handling, connection management, request dispatching, support for servlets[4] and JavaServer Pages** (JSP**),[5] and other facilities related to Web application serving. Another type of application server is a "component transaction server." A component transaction server, in common with transactional servers such as CICS, provides ACID[6] (atomic, consistent, isolated, durable) transaction properties for the applications that run within it. Those applications are typically composed of *components*. A component can be thought of as an application building block that adheres to the conventions of a given *component model*. By following a component model, components can interoperate, take advantage of services defined as part of the component model (for example, life-cycle management and transaction management) and be deployed on any server that complies with the component model. The piece of the application server that implements the component model is often referred to as a *container*.

Component-based application servers provide enhanced application development capabilities and greater reuse of the resulting applications (and application parts) than noncomponent-based servers. The Enterprise JavaBeans (EJB) component model is a cross-industry collaboration coordinated by Sun Microsystems, Inc. under a joint development model known as the Java Community Process. It is supported by a wide range of products, including members of IBM's WebSphere* product family. The COM+ component model was developed by Microsoft Corporation and is supported by the "Microsoft Transaction Server."[7] A third component model, the CORBA Component Model[8] is under development by the Object Management Group (OMG). Other component models may emerge in the future.

The EJB component model provides a vendor and platform-neutral standard for server-side components written in the Java programming language. A piece of business logic or function can be written in the Java language and encapsulated to become a reusable object. These components are known as "enterprise beans." The model allows a developer to focus solely on writing the business logic of an enterprise bean. The way in which the bean interacts with its environment—for transaction services, security services, persistence management, and so on—is separated from the business logic and instead specified via a "deployment descriptor." For instance, a deployment descriptor can specify the transaction management style to be applied to a given method of a bean. Deployment data are typically captured by tooling provided by an EJB vendor in conjunction with its server run time. For example, IBM allows production of enterprise beans and specification of deployment attributes from within the VisualAge* for Java[9] development environment. Deployment data can be changed independently of the business logic, and it is the responsibility of the EJB container to honor the contract specified by the deployment descriptor.

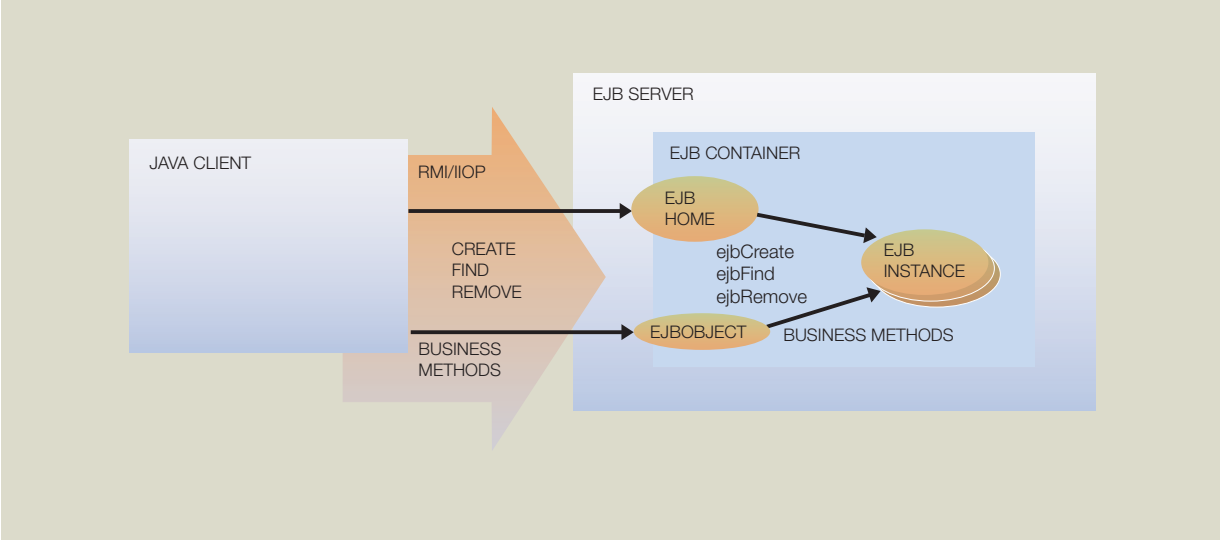
The key elements of deployment data to be specified for an enterprise bean include:

- The type of the bean (entity or session)
- The transaction management to be applied when the methods of the bean are invoked (run under existing transaction, start a new transaction, never run under a transaction, and so on)
- Whether the container is responsible for the persistence of the state data of the bean or whether the bean will undertake that responsibility itself
- Access control for the methods of the bean

The way in which an EJB container honors the contract specified in the deployment descriptor is via *interposition*. That is to say, the container "interposes" (comes between) a client and the enterprise bean logic to ensure that the specified run-time properties (such as the form of transaction management to be used) are maintained. A developer writing an enterprise bean defines its remote interface—the set of business methods that the bean will service, and its home interface—the means to create, find, and remove instances of the bean. He or she then writes the business logic in an enterprise bean that implements the remote interface, following the requirements of the EJB component model.

When an enterprise bean is deployed into an EJB container, the deployment tools provided by the con-

Figure 1 Container interposition on client requests



tainer vendor generate additional classes (a "home" class and a remote or "EJBObject" class) that implement the home and remote interfaces defined by the bean developer. A client of an enterprise bean does not have direct access to that bean but is forced to access it through the generated classes. The generated classes work in conjunction with the container to perform processing before and after delegating to the business method provided by the enterprise bean itself. This *pre-* and *post-invoke* processing is used to bracket the bean call with the necessary transaction, security, and persistence management.

This is a big step forward from the precomponent world where a developer had to learn and master the various application programming interfaces (APIs) needed to interact with run-time services (such as the CORBA services APIs) and explicitly call those services from their application.

Figure 1 depicts the relationship between an EJB server (an application server supporting the EJB component model—for instance, CICS), its EJB container, and client access to enterprise beans deployed within those containers.

Clients typically access enterprise beans using Remote Method Invocation/Internet Inter-Orb Protocol (RMI/IIOP). Java RMI is the programming model; IIOP is the communications protocol. IIOP is an implementation of the General Inter-Orb Protocol (GIOP) based on Transmission Control Protocol/Internet Protocol (TCP/IP). Both IIOP and GIOP are standard CORBA protocols. Clients may also access enterprise beans using the CORBA IDL (interface definition language) programming model and IIOP.

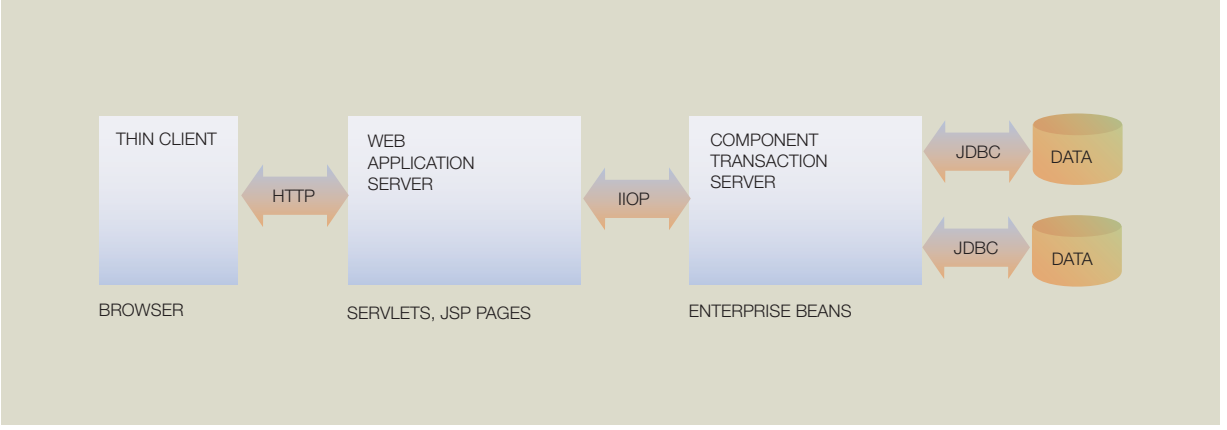
To create, find, or remove an enterprise bean, the client looks up a remote reference to the home object for that bean in the naming service (using JNDI,[10] the Java Naming and Directory Interface**). The client can then invoke the create, find, and remove methods on the EJB home, and these are delegated to corresponding *ejbCreate*, *ejbFind*, and *ejbRemove* methods on the enterprise bean, which contain the supporting business logic.

If an enterprise bean is created or found, a remote reference to the EJBObject is returned to the client. The client invokes methods on the EJBObject, which are in turn delegated to the business implementations provided by the enterprise bean.

Types of enterprise beans. There are two types of enterprise beans: session beans and entity beans.

Session beans are for use by a single client and enable a pseudoconversational interaction style (application data are not locked while control is in the client or is passing between the client and the serv-

Figure 2 Enterprise Java multitier Web application serving configuration



er). A *stateful* session bean contains state particular to a given client, whereas a *stateless* session bean has no client-specific state. Session beans are not recoverable.

Entity beans are shared by multiple clients and provide an object rendering of application data. Entity beans are indexed by a primary key and are analogous to records in a database. Entity bean instances are recoverable. A bean-managed persistence entity bean manages its own state, whereas a container-managed persistence entity bean has its state managed by the EJB container.

An enterprise Java programming model. Enterprise beans are a key component of an enterprise Java programming model. Sun Microsystems, Inc. defines a platform called J2EE** (Java 2 Platform, Enterprise Edition)[11] that combines several technologies for developing enterprise applications in Java. The J2EE platform encompasses technologies such as Enterprise JavaBeans, Java Transaction Service,[12] Java Message Service,[13] servlets, and JSP. IBM is a key supporter and provider of many of these technologies and continues to support them in its products.

Of special interest is the construction of solutions providing Web access to enterprise applications. This gives rise to a multitier architecture as depicted in Figure 2.

In this model, thin clients, such as Web browsers or pervasive computing devices, interact via HTTP with the Web application server. The Web application server provides a container for servlets and JSP pages. Servlets and JSP pages call out to enterprise beans within a component transaction server to execute business logic. The aim of this architecture is to separate model, view, and controller (MVC) into individual components providing better maintainability and serviceability. The MVC approach to building applications splits responsibility into three areas: a model encapsulates the data and business logic, while one or more views provide renderings of those data (for example, a graphical user interface view, a textual user interface view, or a Web-based view). Controllers coordinate between users and the model: user actions are translated by the controller into requests against the model, and any subsequent change in the model state is reflected by the views.

In our case, the servlet is the controller. It parses an incoming HTTP request and extracts the request data. The servlet processes the request by invoking methods on one or more enterprise beans. The enterprise beans are the model; they contain the business logic and reside within the component transaction server so that they run in a secure transactional environment. Enterprise beans within the component transaction server interact with back-end data servers to fulfill the request. Results returned by the enterprise bean(s) may be encapsulated as JavaBeans**.[14] The servlet then passes control to a JSP (the view). The JSP constructs a response (e.g., formats an HTML [HyperText Markup Language] page, or XML [Extensible Markup Language][15] document) for sending back to the client, extracting any dynamic parameters from JavaBeans passed to it by the servlet.

Graphic designers and Web editors can work on presentation (JSP pages) independent of code to manage business logic (enterprise beans) or protocol (servlets). Likewise, programmers do not have to litter servlet code with presentation logic. Those who understand the business can focus on business logic without being concerned as to how it is accessed or presented.

A complete enterprise Java solution often involves multiple products combining to offer the full range of Java services. For example, the Web application server could be WebSphere Standard Edition running on the Windows NT**, AIX* (Advanced Interactive Executive), or z/OS (the operating system for the IBM z Series servers) platforms; the component transaction server could be CICS or WebSphere Enterprise Edition running on z/OS.

CICS strategy for Enterprise JavaBeans

In developing the technical strategy and design approach for EJB support in CICS, we have worked with a number of customers and have made extensive use of the "user-centered design" methodology. We have often asked customers who are keen to use enterprise beans in CICS what makes this an attractive proposition. Existing CICS customers see two principal motivations, both involving the operational reuse of existing CICS applications and data: (1) Web access to existing applications, and (2) representation of applications and data as components to enable rapid and easy assembly of new business functions.

In both cases, the most pressing requirement is to provide access to existing applications or data, or both, from enterprise beans (a process sometimes known as "wrappering").

Of course, there are other motivations regarded by our customers as important for using enterprise Java technology, including:

1. Cross-platform exploitation of new technology. Many of our customers are already starting to take advantage of enterprise Java technologies. Among the reasons most frequently cited are: productivity gains, openness of the technology, and anticipated availability of third-party components. Such customers typically have a range of software platforms, indicating a strong requirement for EJB server implementations that are consistent and that have *common tooling*. Particularly when used as part of Web-based solutions, the workload of such applications may be unpredictable, significant, and growing rapidly. Key requirements here include *scalability* and *manageability*.
2. Evolutionary development of existing systems. The vast majority of customers who will use the CICS support for EJB will do so in conjunction with existing applications, data, and information technology infrastructure. New EJB-based applications will have to *coexist and interoperate* with those applications. In many cases, enterprise beans that are deployed in CICS will need to make use of CICS-specific services. And, of course, the requirement for good *performance* will be no different than for any other CICS transaction.
3. Availability of skills. Another factor that is increasingly important to many customers is the ability to attract and retain highly skilled application developers. Being able to use a leading-edge technology such as EJB should make it much easier to maintain and grow an application development community that is keen to work on a platform such as CICS. A key requirement is to make the process of developing applications for CICS more attractive through the use of workstation-based *common tooling*.
4. High-end growth. Customers who have developed applications on Windows NT, AIX, or other UNIX** platforms are increasingly deciding to move the application to z/OS. This movement may be to improve scalability and performance, to reduce the number of systems being managed, or simply to roll out across an enterprise an application that was piloted on the other platform. Such migration to the mainframe computer may increase with EJB-based applications. Given the EJB aim of "write once, run anywhere"**, an EJB-based application could be prototyped, and perhaps piloted, on one platform, but deployed, for enterprise-wide use, on a high-end server such as CICS or WebSphere Enterprise Edition. The critical requirement in such cases is for *portability* of enterprise beans.

In considering how best to address these diverse requirements, we have identified five critical success factors for CICS EJB support:

1. Maintaining the CICS value proposition. That is to say, the qualities of service traditionally associated with CICS and associated products (robustness, data integrity, reliability, etc.) must continue to apply to the CICS EJB server capabilities. The

EJB support must capitalize, and build, upon the existing strengths of the product.
2. Exploitation of existing CICS resources. Given that the most immediate requirement appears to be for the "wrappering" of existing applications and data, we need to provide effective and efficient mechanisms for achieving this result.
3. Fully compliant support for "any" session bean. It should be possible to deploy standard session beans developed by using any application development tool within CICS. (Although it will certainly be possible for enterprise beans to make direct use of CICS services via the Java CICS [JCICS][16] classes, they are not required to do so.)
4. Performance and scalability consistent with CICS customer expectations. The key to achieving satisfactory performance will be to minimize the overhead (particularly in initialization) associated with Java transactions. Although the cost of executing Java transactions will inevitably remain somewhat greater than for languages such as COBOL, our aim is to reduce the overhead to a small proportion of the total cost of the transaction. In any event, the additional overhead will be more than offset by the improved development efficiency of building new applications in the Java language. In addition to the performance of individual transactions, we must also ensure that the scalability of the CICS EJB environment is not constrained in any way.
5. Integration within existing operational infrastructures. The deployment and management of enterprise beans within the CICS environment should not require a separate systems management infrastructure; in other words, it must be possible to achieve this using current CICS System Manager tools.

These success factors present some interesting technical challenges, including:

- Effective integration of the transaction and recovery management styles of EJB and CICS
- Isolation of distinct transaction invocations, without incurring significant Java initialization costs
- Management of enterprise bean workload within the CICS and MVS* (multiple virtual storage) system structure
- Enabling the use of application development tools that are not specific to CICS and that typically run on a different platform

If, as we believe will be the case, we are successful in addressing these challenges, CICS and other members of the WebSphere product family will offer the most robust and scalable platform in existence for the deployment of enterprise beans.

Enterprise JavaBeans in CICS. In the first release of its implementation of the EJB architecture, CICS transaction services for z/OS will provide first-class, fully integrated support for session beans. Recall that session beans are for use by a single client and support a pseudoconversational programming style; pseudoconversational transactions are the core of the CICS business and, hence, session beans are an excellent and natural fit in the CICS environment. A session bean deployed within CICS may be completely new and use Java Database Connectivity (JDBC**)[17] to access data from a relational database. Alternatively, the session bean may access data via entity beans deployed in other members of the WebSphere family such as WebSphere Enterprise Edition.

As noted, we expect that one of the most common uses of session beans within CICS, at least to begin with, will be to provide client access to existing CICS transactions, programs, and resources via IIOP. These session beans may use the JCICS API already provided within CICS to execute CICS transactions, link to existing CICS programs, and access resources managed by CICS such as transient data queues and VSAM (virtual storage access method) files. It is expected that access to CICS resources will be modeled by *dependent objects* behind session beans. A dependent object is one whose life cycle is fully managed by its enclosing bean and which cannot be accessed except through that bean.

The next section describes how the EJB architecture relates to the CICS programming model. We then outline the elements of the CICS EJB design that will enable our goals to be achieved.

Architecture for EJB support on CICS

In describing the approach that has been taken for the design and implementation of support for the EJB architecture in CICS, it is worth restating two of the key technical challenges, namely,

- Effective integration of the EJB infrastructure with existing facilities of CICS
- Achieving satisfactory levels of performance *and* scalability

The first of these points requires that the integration of an EJB container into the CICS run-time in-

frastructure be done in such a way as to exploit the functions of the existing CICS domains (such as recovery management); the CICS qualities of service would not be maintained simply by running a "standalone" EJB implementation within a CICS region. Similarly, in order to achieve the desired level of scalability (and transaction throughput), it is necessary to be able to distribute and replicate the elements of the CICS EJB server support across multiple regions, taking advantage of CICS and MVS workload

We made the decision to reuse the WebSphere Advanced Edition EJB container as a basis for the EJB server support in CICS.

management techniques, in a way that does not restrict the ability to balance a client's requests across a whole system image or sysplex. And the performance of individual transactions (as opposed to the overall transaction rate and throughput that will be achievable) will depend, among other factors, upon reducing the initialization costs typically associated with Java programs.

About two years ago, we made the decision to reuse the WebSphere Advanced Edition (WSAE) EJB container as a basis for the EJB server support in CICS. But, for the reasons outlined above, the approach we have taken is to provide the services required by mapping onto and integrating the container with CICS facilities. So, although the effort to implement support for enterprise beans in CICS has obviously required significant additions to the CICS run-time infrastructure, we have also achieved significant reuse of pre-existing CICS function (as well as reusing the WSAE container itself). By doing so, we hope to achieve a level of robustness that would have been almost impossible to attain as quickly had we chosen to implement a brand new set of services to underpin the container.

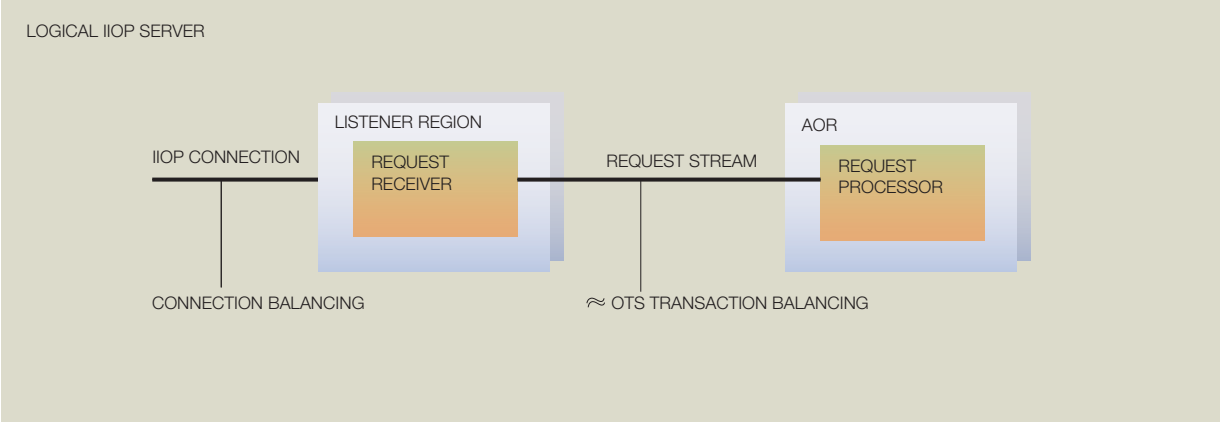
The architecture for supporting EJB within CICS is based on the notion of an "IIOP server." It is the CICS IIOP server that provides the run-time environment in which the container and, in turn, deployed enterprise beans execute and from which they may interact with other CICS services and resources. A key element of the IIOP server design is the provision of a transaction service, based on the OMG Object Transaction Service (OTS) specification.[18] When an enterprise bean that is deployed within CICS is operating as part of an OTS transaction, it will execute within a CICS unit of work that is coordinated by the OTS transaction. The transaction service component of the CICS IIOP server implements the OTS protocols that provide transactional interoperability with other OTS-based enterprise Java servers, and interfaces within the CICS recovery manager. It is via the CICS recovery manager that transactional coordination of local resource managers (such as DB2*, DATABASE 2*) and other systems that do not support OTS interoperability (but which do implement another distributed two-phase commit protocol, such as that of LU 6.2) is achieved.

Although some components of the CICS IIOP server are written in a language called PL/X, the bulk of the run-time environment, including the container itself, is written in the Java language. And, of course, so are enterprise beans themselves. So the performance of Java code executing in CICS is of paramount importance in meeting the aim of achieving "satisfactory" performance for enterprise beans executing in CICS. Moreover, since Java virtual machine (Jvm) instances may accumulate side-effects from applications, we are providing mechanisms that allow Jvm instances to be reused across multiple CICS transactions, with the option of "resetting" (without fully reinitializing) the Jvm after each use.

One of the principal mechanisms that will enable the combined aims of performance and integrity to be met is what we refer to as the "persistent reusable" Java virtual machine. This version of the Jvm has been optimized for the execution of high-volume, short-running transactions on the S/390* platform; it enables the bulk of Java initialization costs to be avoided (or, to be more precise, amortized across multiple transactions) by reusing Jvm instances for multiple transaction invocations. From the perspective of an application, a persistent reusable Jvm will appear no different from any other short-lived Jvm instance, but system objects and classes will be maintained (and reset, when necessary) across multiple invocations. The lifetime of application objects, however, will correspond to that of the relevant transaction, and the application heap will be reset after each invocation, thereby maintaining isolation between transactions.

The design of the CICS IIOP server, together with the transaction service that operates (in conjunction with

Figure 3 CICS IIOP server



the container) within the IIOP server infrastructure, the persistent reusable Jvm, and the approach that has been taken to managing the workload associated with the execution of enterprise beans within CICS, are described in greater detail in the following subsections.

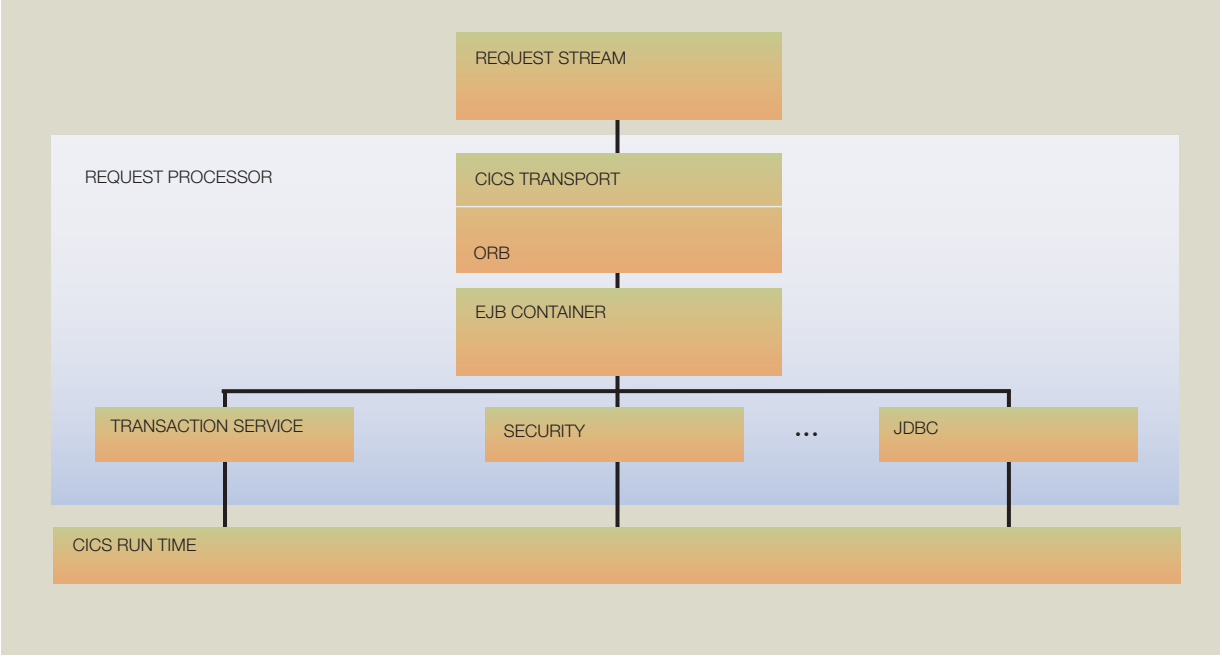
CICS IIOP server design. CICS supports enterprise beans using an IIOP server constructed from transaction instances known as *request receivers* and *request processors* that communicate with each other via *request streams*, as shown in Figure 3. When a client uses an enterprise bean in CICS, the client object request broker (ORB) forms a TCP/IP connection to a port in CICS. The TCP/IP connection may be workload-balanced across a set of cloned listener regions (prior to any workload balancing that is performed within the CICS environment), but ultimately the connection is made to a particular listener region, and a request receiver is assigned to the connection. The client's method requests are converted into IIOP messages by the client ORB and flow across the connection to CICS.

The request receiver forwards each message to a request processor via a request stream. The processor instantiates the object identified by the request in memory as a servant, calls the servant to process the method, and sends a reply to the client via the request stream, the receiver, and the TCP/IP connection. The client ORB converts the IIOP reply message into a return from the method request. The receiver and processor are active and communicate via the request stream, which is essentially passive. Note that the receiver may detach when it is not waiting for a reply from a request processor. It issues an asynchronous socket receive before detaching that results in a notification and an equivalent receiver being attached when data arrive at the socket. The IIOP server also supports stateless CORBA objects introduced in CICS Transaction Server Version 1.3. Stateless CORBA objects are similar to stateless session beans, but lack standardization.

Internally, CICS is composed of encapsulated components known as "domains." Enterprise JavaBeans support is provided principally by six domains: Enterprise Java, Object Transaction Service, Jvm Management, IIOP, Request Streams, and Sockets. Although domains are not implemented in Java, some of them have related Java components: EJB container, transaction service, and ORB. There are also Java components to support JNDI, JDBC, and SQLJ (Structured Query Language for Java).[19]

Request streams. The request streams domain is responsible for transmission of opaque requests and replies between transaction instances running in possibly distinct regions. The request streams domain provides workload balancing and attach processing of a transaction instance that receives requests from a particular request stream. Workload balancing occurs when a request stream is created and is typically based on workload and availability data relating to a cloned set of application owning regions (AORs). Once a request stream has been created, multiple requests may be sent to the corresponding request processor without further workload balancing. The request stream and its processor terminate

Figure 4 Request processor



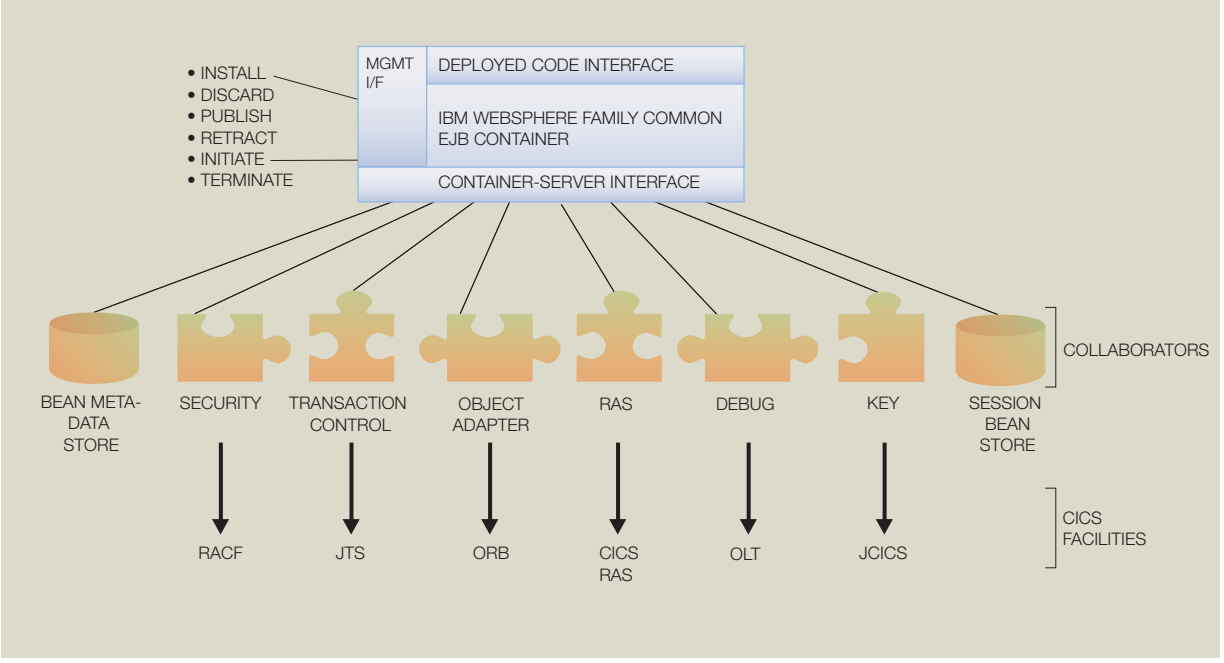
as soon as possible (i.e., at the end of a method request or of an OTS transaction; see below) so that there is the maximum opportunity for workload balancing.

An existing request stream may be used from the creating receiver or from another receiver or processor possibly on another region. "Inbound" request streams support intra-region and inter-region transport mechanisms, the latter being based on CICS multiregion operation (MRO) protocols. "Outbound" request streams forward requests unmodified to a remote TCP/IP socket and receive replies to such outbound requests. Both inbound and outbound request streams support an asynchronous form of receive. The caller listens for receipt of data and is notified when data arrive. The caller may wait until it is notified and then receive data, or it may cancel listening. The Request Streams domain was separated from the IIOP domain because it is IIOP-neutral and is intended to be suitable for use by components that support other protocols. So far the request processor (shown in Figure 4) has been treated as a black box. Now we look inside.

The request processor is concerned with creating "servant" objects and processing method requests on them. A servant object is an in-memory representation of a remote object. The request processor has three components that manage servants: the container, which manages enterprise bean servants, the transaction service, which manages its own servants, and the ORB, which, apart from its other responsibilities, manages stateless CORBA object servants. The request processor sends and receives requests via the ORB, which understands the IIOP protocol. The ORB converts request messages into method invocations on a servant and converts method results into reply messages. The ORB also converts method requests on stubs into request messages to a remote server and converts reply messages into results from method requests.

The ORB processes a request by constructing a servant, if necessary, and invoking the servant to process the method specified in the request. Servant managers isolate the ORB from the complexities of managing servants; they are created using the ORB during initialization of the request processor and are called by the ORB to construct servants corresponding to object keys. Various types of servant are supported: enterprise beans and their homes, stateless CORBA objects, and transaction service objects (which have state but are nontransactional).

Figure 5 EJB container integration in CICS



EJB container. The EJB container provides the run-time environment for servants of enterprise beans and their homes. The container is initialized by the ORB during request processor initialization and creates a servant manager using the ORB. When the ORB calls the servant manager of the container to construct a servant for a given object key, the container constructs the appropriate environment around the servant. The servant manager of the container returns the servant to the ORB. When the ORB invokes the servant to process a method request, the servant can use the environment that was established earlier.

The EJB container used within the CICS solution is shared with other members of the WebSphere family. This sharing provides a consistent interpretation of the EJB specification across the WebSphere products and facilitates common application development tooling and systems management for enterprise Java technologies. Figure 5 shows how the IBM WebSphere family EJB common container fits into the CICS environment.

At the left side of the figure is the (systems) *management interface* exported by the container. This interface is used by CICS Resource Definition On-line commands to install and discard deployed Java archive (JAR) files from the container, and to publish and retract the homes of enterprise beans installed in the container. It is also used to initialize and terminate an EJB container instance within the request processor.

The *deployed-code interface* is the interface provided to the WebSphere EJB deployment utilities. These utilities are tools such as VisualAge for Java and the WebSphere Application Assembly Tool. The home and remote (EJBObject) classes generated by these tools use the deployed-code interface to perform the required container interposition on enterprise bean method invocations.

The *container-server interface* is the means by which the container obtains the run-time services it needs. For each service that the container requires, a Java interface is defined, and the server hosting the container (in this case CICS) provides a *collaborator* that implements the interface. The collaborators map from the Java service definition to the underlying CICS facilities. The core collaborators are:

- The Bean Meta-data Store, which provides a persistent store for bean meta-data (deployment attributes) for beans installed within the container. The Bean Meta-data Store is maintained in shared storage inside the Enterprise Java domain.
- The security collaborator, which provides security role checking and access to user principal information. The security collaborator will, in a later release, be backed by definitions stored in the Resource Access Control Facility (RACF*).
- The transaction control collaborator, which performs transaction management interposition for bean- and container-managed transactions. The transaction control collaborator uses the transaction service to achieve its end.
- The object adapter/servant manager collaborator, which provides the interface to the ORB.
- The RAS (reliability, availability, serviceability) collaborator, which maps tracing, logging, and dump information issued by the container into the existing CICS RAS facilities.
- The debug collaborator, which provides object-level trace (OLT) capabilities for remote debugging.
- The stateful session bean key generating collaborator, which creates server-unique primary keys for stateful session beans.
- The session bean store, which holds passivated stateful session beans in a shared CICS file.

Outbound IIOP. A method running in a request processor may call remote objects such as enterprise beans, OTS transaction coordinators, or other types of CORBA objects. The processor passes such requests via an "outbound" request stream to a TCP/IP connection to a remote server.

The ORB (in collaboration with the IIOP domain) determines whether the method request can be processed locally and, if not, whether it could be processed by the current IIOP server. If the method request is to be processed locally, the object is invoked locally similarly to the way in which its calling object was invoked. If the method request is to be processed by the current IIOP server, the same mechanisms employed in the receiver are used to determine the target request stream and, in the case of a new request stream, the target AOR. If the target AOR is not the calling AOR and is not connected by MRO to the calling AOR, the method request must use an outbound TCP/IP connection.

There is a separation of concerns. The processor deals with inbound and outbound request streams that in turn deal with transport mechanisms such as MRO and TCP/IP. In particular circumstances, an outbound method request may be processed in the calling request processor. This is possible when the called object belongs to the same CICS IIOP server as the calling object and when the caller's unit of work does not need to be "suspended" and "resumed" during the call since this is not supported within a single CICS transaction instance.

Support for OTS transactions. As we mentioned previously, an important part of the IIOP server is the transaction service, which is based on the protocols of the OTS specification. Combining the transaction service with the underlying transaction and recovery management facilities of CICS has been one of the key challenges in achieving an effective integration of the EJB infrastructure with CICS itself. The basic approach we have taken is that when a method of an enterprise bean executes in CICS as part of an OTS transaction, it executes with a CICS unit of work that is coordinated by the OTS transaction. Whether a method of an enterprise bean should be run under control of an OTS transaction is determined by the "transaction attribute" in the deployment descriptor; depending upon the setting of this attribute, the method may run under the caller's OTS transaction (if there is one), under a new OTS transaction that is created specifically for the duration of the method, or under no OTS transaction at all.

When a method is run under control of an OTS transaction, the transaction service has to interact with a number of other components. As is the case in any interoperable EJB server, the transaction service uses the OTS protocols to communicate with any other OTS-based systems that may be involved in the transaction. To control the two-phase commitment processing of any non-OTS resources that may be involved in the transaction, the transaction service makes use of the CICS recovery manager to drive the necessary two-phase commit flows. (For example, if the enterprise bean were to invoke a CICS transaction that communicated with a remote system using the LU 6.2 protocol, resources on that remote system would be committed or rolled back in this way, via the CICS recovery manager.) A further consideration is the possibility that enterprise beans may make use of JDBC services to access a local DB2 system. The implementation of JDBC services within the CICS IIOP server makes use of the CICS resource manager interface to provide access to DB2 via what is known as the CICS DB2 task-related user exit; the resource manager interface also handles registration with the CICS recovery manager, so that when the transaction

service commits or rolls back a transaction, the recovery manager will drive the appropriate action through to the local DB2 system (as well as to any remote non-OTS systems).

One advantage of reusing the CICS infrastructure in this way is that the fundamental two-phase commitment and resynchronization mechanisms are not dependent upon the Jvm. The use of a Jvm may be terminated when it is not required (for example, during an extended wait for the resolution of a transaction, such as may occur after a communications failure to a superior coordinator system). This allows resources such as Jvms and open task control blocks to be used by other transactions, thereby improving the overall throughput of the system.

The "persistent reusable" Java virtual machine. Another important aspect of the design of the CICS IIOP server is the way in which the efficiency of Jvm operation is improved. As we have already outlined, a significant improvement in performance has been realized by exploiting the "persistent reusable" Jvm technology. (This enhancement of the Jvm is, in fact, not specific to CICS. It is being developed as a standard part of the Java Development Kit [JDK**] that is shipped with the z/OS platform, although we have worked closely with the IBM Centre for Java Technology on its design and implementation.)

Typically, one of the most significant execution-time costs of running Java programs is the initialization of Jvm instances. To avoid this cost being incurred each time a request processor is started, Jvms are serially reused across multiple request processors. So, when a request processor is initialized, a pre-existing Jvm is acquired from a pool; the Jvm then remains allocated to the request processor until the request processor terminates. To prevent interference between application instances and ensure the integrity of the data, a distinction is made between the system and middleware heaps (the main areas of storage allocated to system and middleware components, which contain the system objects and classes that need to be long-lived) and the application heap (the main area of storage allocated for objects relating to a particular application instance). It is this Jvm technology that enables request processors to serially reuse the long-lived contents of the system and middleware heaps by resetting them after use and reinitializing them before subsequent reuse; this avoids the significantly greater cost of initializing a Jvm instance from scratch each time it is used. By contrast, the application heap may be discarded on termination of a request processor. A further reduction in initialization cost has been achieved by making part of the ORB, EJB container, and transaction service state reusable across multiple request processor instances. Reuse is achieved by allowing these parts to be held in the middleware heap. Additionally, we hope to optimize the garbage collection mechanisms of the persistent reusable Jvm by virtue of the fact that there is a clean separation of short-lived application objects from long-lived system objects.

Workload management. Workload may be balanced across listeners and AORs to achieve performance objectives such as response time. This is achieved in cooperation with the MVS workload manager (WLM), as shown in Figure 6. The MVS WLM gathers workload and availability data for each of the CICS regions comprising a logical EJB server, which may be used to balance workload across the regions of the logical server.

Incoming TCP/IP connections to the logical server may be balanced using either a "dynamic" domain name server, an IP router, or a combination of the two. Each connection may be directed to an appropriate request receiver, based on the workload and availability of the listener regions.

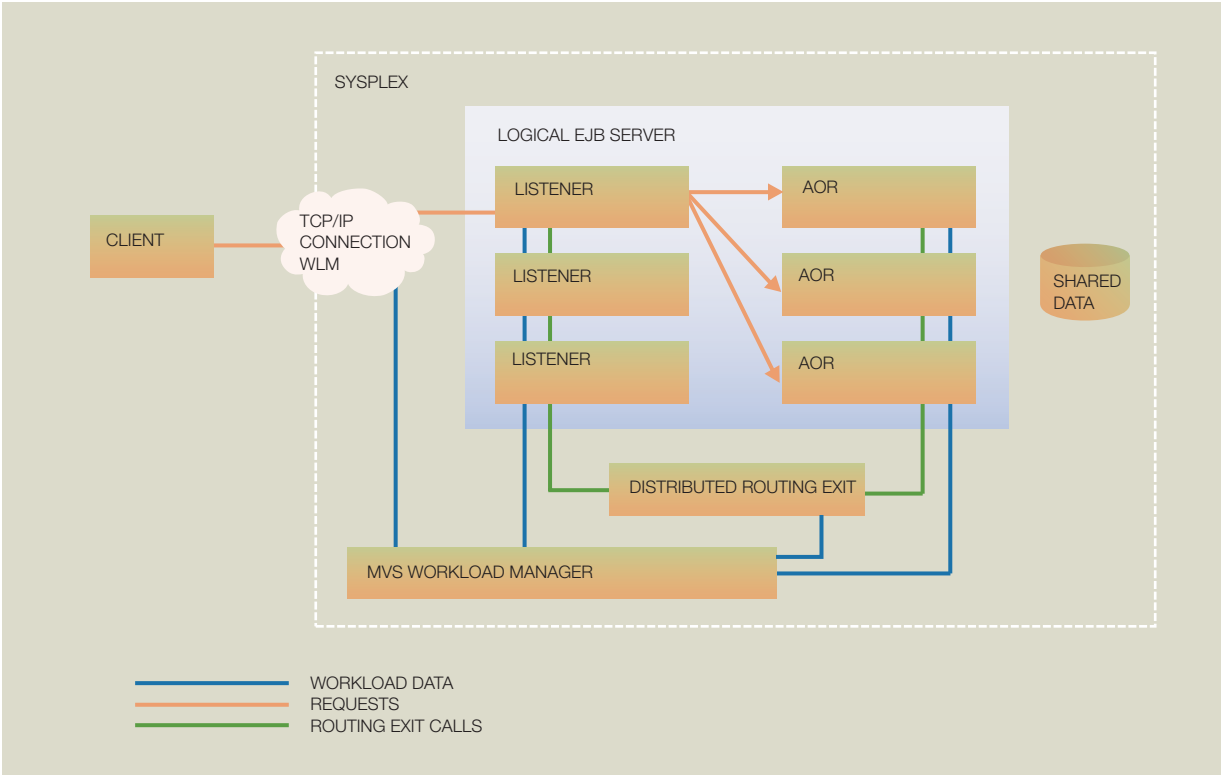
Once a connection has been made, requests from listeners to AORs can be workload-balanced using the CICS distributed routing exit. The routing exit implementation supplied with CICS, as part of the CICSPlex System Manager (CPSM) component, uses the workload and availability data, collected by CPSM from the MVS WLM, to select an AOR.

Tooling for CICS EJB servers

A significant benefit of the introduction of support for Enterprise JavaBeans into CICS is that it enables a whole new set of software development tools and practices to be used to write software for the CICS environment. This is not to say that the bulk of the work to support Enterprise JavaBeans in CICS has been in the area of tools. With respect to application development, our primary goal has been to minimize the CICS-specific tooling that was required, and where CICS-specific tooling was required, to integrate it as seamlessly as possible with the relevant IBM application development tools product.

There are a number of areas (for example, the capabilities for debugging enterprise beans deployed

Figure 6 Workload management



in CICS) in which we will deliver improved functionality in subsequent releases, so this section describes both the longer-term strategy for EJB-based application development for CICS and also elements of the tooling that will be shipped for use with the first release of the CICS EJB support.

One of the main reasons for basing the support for EJB in CICS on the WebSphere common container was to allow the IBM WebSphere tooling to be used to develop enterprise beans that can be deployed, without change, into CICS. Session beans that use JDBC to access data in DB2 can be developed and tested in the WebSphere Test Environment before being deployed into a CICS region for final testing on z/OS. Nevertheless, it is not necessary to use WebSphere tools to develop enterprise beans for CICS; CICS supports the stand-alone tools that can be used with WebSphere Advanced Edition, and these tools can be integrated with third-party development environments.

In addition to supporting "generic" session beans without the developer having to be aware that CICS may be the ultimate deployment platform, CICS-specific session beans can be developed using the JCICS API, either directly or via tools, and the specific support for CICS will be integrated into the IBM tools. An extra challenge is to minimize the impact of the fact that, typically, enterprise beans and applications will be developed on a workstation platform, not z/OS, and therefore support for remote application development is a necessary part of the CICS application development strategy.

We can divide the tooling support into three pieces, targeted at different aspects of the overall application development process: development, deployment, and debugging. The following subsections describe the support in each of these areas.

Development. As mentioned above, the main focus for development tools is the set of tools products that

support WebSphere Advanced Edition, but standalone tools are also supported for those developers who prefer to work with a text editor and the JDK or software developer kit, and for those independent software vendors (ISVs) who intend to produce CICS EJB server-based application development solutions. WebSphere extensions to the standard Enterprise JavaBeans programming model, such as access beans and associations, are fully supported. The same deployed Java archive (JAR) file that is supported by WebSphere Advanced Edition is supported and, indeed, required by CICS. The same WebSphere-specific generated code (stubs, ties, implementation classes for the home and remote interfaces, etc.) is used in the CICS environment.

The following subsections describe the programming models that we expect developers to use, and describe in more detail the tool or API support available. The programming model that we envisage being used with the support for EJB in CICS is that session beans will be developed that access resources, either data or CICS programs. The most common types of data we expect to be accessed are held in DB2 and in VSAM. The most common types of programs we expect to be accessed are CICS COBOL programs. The support for accessing each of these resource types is described in the subsections.

Accessing data in DB2. CICS will support the new Type 2 JDBC driver provided by DB2 for z/OS, including SQLJ support. It is possible to write code that uses the JDBC API explicitly that runs both on z/OS and on workstation platforms. In addition, we will support the core (nonvisual) beans provided as part of the data access beans feature with VisualAge for Java. These data access beans can be used programmatically in any environment, but there are SmartGuides supplied with VisualAge for Java that make them even easier to use. There are three such beans:

1. Select Bean—The Select Bean executes SQL queries and returns a modifiable result set, similar to the JDBC 2.0 RowSet. This result set can be used to insert, update, or delete a row in the database without further explicit SQL calls.
2. Modify Bean—The Modify Bean executes SQL INSERT, UPDATE, or DELETE statements without first running a query.
3. ProcedureCall Bean—The ProcedureCall Bean executes a stored procedure and returns one or more modifiable result sets.

The data access beans use a common SQL syntax so that they work both on z/OS and on the workstation platforms. An SQL Assist SmartGuide is provided that allows for the visual composition of an SQL statement.

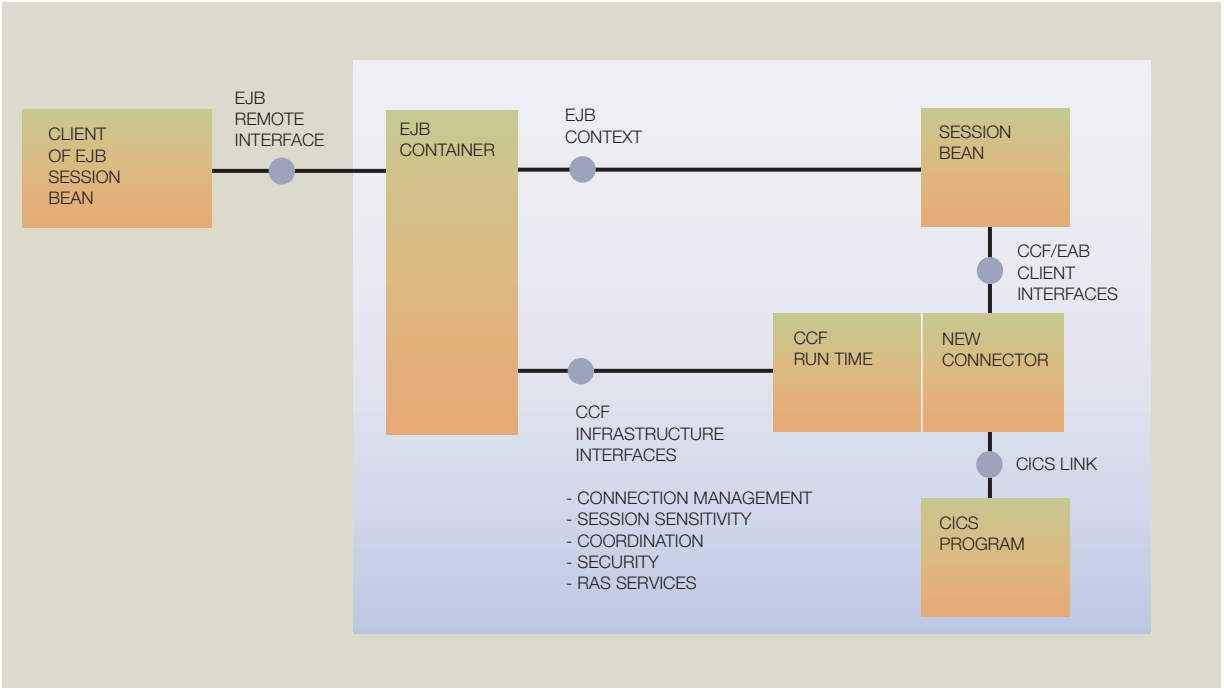
Accessing data in VSAM. The last release of CICS (CICS Transaction Server Version 1.3) includes the JCICS API, which provides a Java version of a subset of the CICS server API, including File Control. This support will, in a future release, be extended by providing full support for the Java Record Framework in the File Control part of the JCICS API. In the current release, the application developer is responsible for invoking the conversion to and from byte arrays provided by the code generated by the Java Record Framework tools that are supplied with VisualAge for Java Enterprise Edition. The JCICS File Control API will be enhanced so that the beans or classes generated by the Java Record Framework tools can be passed directly to the JCICS API, and CICS will manage the conversion to or from byte arrays.

Accessing CICS programs. We expect that many of the initial enterprise beans that are deployed into CICS will wrap existing programs, typically COBOL programs. There is support in the JCICS API to invoke a CICS program, which could be in any programming language supported by CICS, including support for the Java Record Framework to automate the conversion of data from Java programs to, typically, COBOL.

The relevant IBM tools are the Common Connector Framework (CCF) and the Enterprise Access Builder (EAB) for Transactions, both of which are delivered as part of VisualAge for Java Enterprise Edition. The CCF provides a common programming model for users of connectors and common infrastructure for connector providers. Over time, it is anticipated that the IBM CICS connectors will be migrated to the J2EE Connector Architecture.[20]

The EAB provides tooling that simplifies the use of connectors and provides support for higher-level functions such as composition of commands and automatic generation of session beans. The fundamental EAB concept is that of a *command*. A command encapsulates a single interaction with a system. A command consists of a connector, the input of the interaction and the output of the interaction. The definition of the connector consists of the ConnectionSpec, which defines the system to be connected to and the means of communication, and the Inter-

Figure 7 Use of a connector by a session bean



actionSpec, which defines the operation to be performed in the host system. The input and output can be *records* generated by the Java Record Framework and tools or other connector-specific objects. Each input/output bean or object can have an associated *mapper* that maps properties of the generated bean or object to properties of the application object. Typically, records are used, and these have an interface very similar to the COBOL copybook from which they were generated, for example. A mapper might be used when the new domain object model being defined is significantly different from the existing COBOL data definition.

Once a command has been defined, other EAB tools can be used to aggregate a set of commands into a navigator, which represents a more complex interaction with a host system, including sequences of interactions, branches, loops, etc. These navigators appear as commands and so can themselves be composed into other navigators, etc. Another option is to generate a session bean from a set of commands or navigators. Each method of the session bean invokes a command or navigator. For maximum portability of code that uses the External Call Interface (ECI) connector, CICS will support code running inside CICS that uses the ECI connector, although a small number of changes to the ConnectionSpec or the InteractionSpec, or both, may be necessary. This support uses the CICS Transaction Gateway Java classes and reimplements the native code layer to issue an EXEC CICS LINK call instead of an ECI call. Figure 7 shows the run-time architecture of a session bean using this "local" connector to wrap a CICS program.

The use of a command to encapsulate the invocation of a CICS program from a session bean makes it easy to begin to develop session beans using WebSphere Advanced Edition by initially using the CICS ECI connector in conjunction with the CICS Transaction Gateway and then migrating the commands to the new CICS connector that can be used in the CICS server environment. The session beans should not need to change to run in CICS itself. The record produced from the COBOL copybook, for example, will need to be regenerated for MVS as part of the migration of the command, but the COBOL code itself would not need to be parsed again.

Deployment. There are many different interpretations of the word deployment. For the purposes described here, deployment means taking a deployed JAR file produced by the tools and producing the necessary CICS resource definitions to make the beans in the JAR file available. For session beans, this deployed JAR file will contain:

1. The classes or interfaces defined by the developer for each enterprise bean, and any associated classes
2. The WebSphere-specific generated classes
3. The information defined by the EJB 1.1 XML Deployment Descriptor
4. WebSphere-specific meta-data, if appropriate
5. CICS-specific meta-data, if appropriate

There are two different styles of deployment into CICS: unit test deployment, that is, deployment by an application developer into a unit test region, ideally directly from the application development environment; and production deployment, by which we mean deployment by (for example) a system programmer into a system test or production region.

The goals for unit test deployment are to minimize the CICS-specific information that needs to be supplied and to make it as easy to deploy enterprise beans into CICS as it is to deploy them into the WebSphere Test Environment. Whenever possible, defaults will be supplied by the server components of the unit test deployment tools. The only CICS-specific information that can be supplied by the application developer is the CICS transaction identifier to be used when a method request is dispatched. This might be important where an existing CICS program is being called, or even replaced, by an enterprise bean method.

The basic approach for unit test deployment is to provide a client/server system, with the client running in the application development environment and the server running partly on a Web server and partly inside CICS on z/OS. The client will take a deployed JAR file, as described above, and then ship this JAR file to the server using HTTP. The server will allow a CICS system programmer to influence the deployment of the JAR file by the user and will allow defaults to be supplied when the application programmer has chosen not to supply the information, which could be the default case when generic beans are being deployed into CICS. The necessary CICS resource definitions are produced automatically by a set of enterprise beans running in the target CICS system. The JAR file is transferred to the target CICS system using File Transfer Protocol (FTP).

Figure 8 shows the approach that we anticipate using with the first release of the CICS support for Enterprise JavaBeans.

In the medium term, our approach for remote deployment is to use WebDAV.[21] WebDAV (Web-based Distributed Authoring and Versioning) is an open standard, defined by the Internet Engineering Task Force (IETF) Request for Comment (RFC) 2518. WebDAV defines extensions to the HTTP 1.1 protocol to support a kind of distributed repository system using Web technology.

Figure 9 shows the WebDAV-based solution that we aim to support in future releases. With support for WebDAV, any client that supports WebDAV will be able to "publish" a deployed JAR file to CICS transparently.

Regardless of the technology used to transport the JAR file to the CICS server, the process is similar:

1. Develop the enterprise beans and the standard deployment descriptor. Any EJB 1.1 tool can be used for this development.
2. Supply any necessary WebSphere-specific meta-data. Any WebSphere-aware tool can be used for this operation.
3. Supply any necessary CICS-specific meta-data. Any CICS-aware tool can be used for this operation.
4. Invoke the appropriate client, either a WebDAV client or the CICS unit test deployment client, identifying the JAR file and the CICS system to which the JAR file should be deployed.

The goals for production deployment are to provide a style of interaction that is appropriate to a CICS system programmer, rather than an application developer, and to provide full control of how the enterprise beans are deployed into CICS. During production deployment, all of the necessary resource definitions are produced under the control of the system programmer using the production deployment tool. If an application developer has supplied any WebSphere-specific or CICS-specific information, that information is available to the system programmer, but it can be changed by the system programmer if desired. The JAR file and the generated resources are transported manually to the target CICS system(s).

Figure 8 CICS transaction server application development infrastructure for EJB

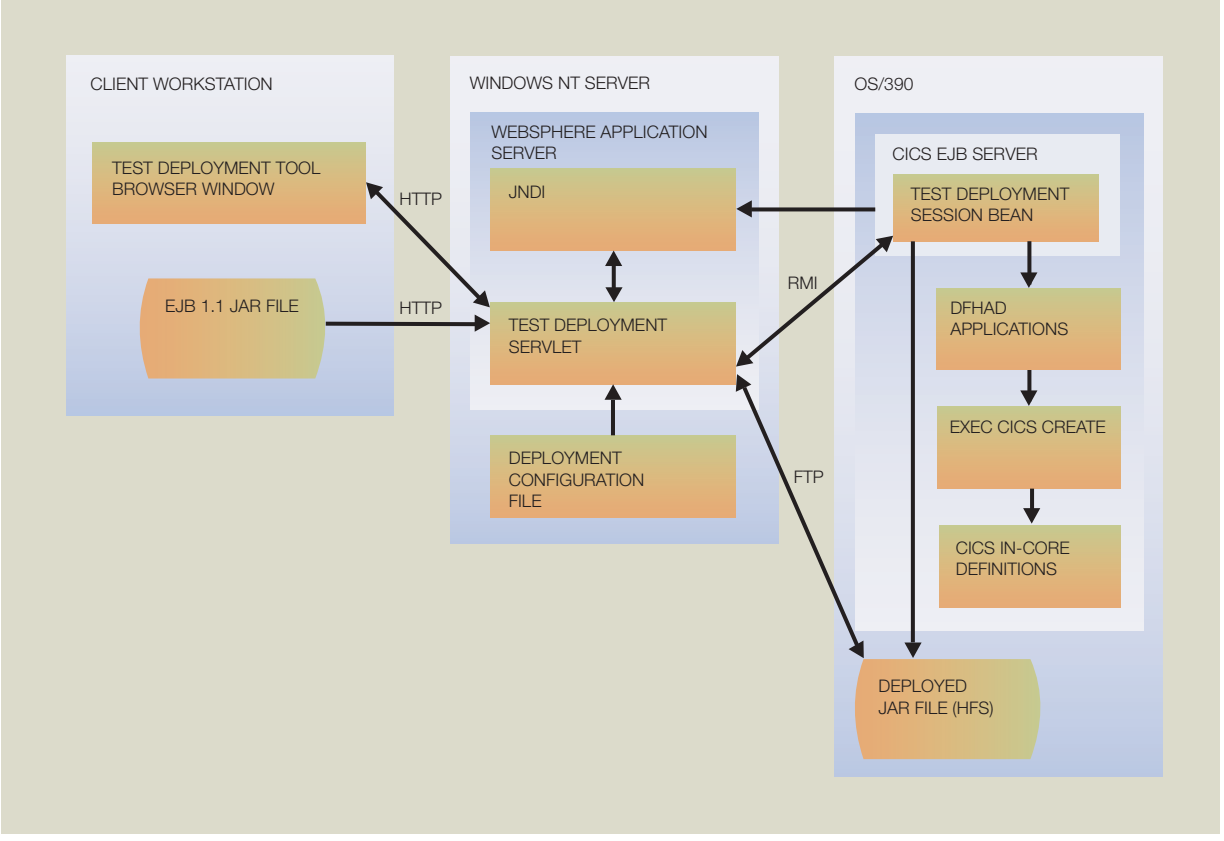


Figure 9 Future application development infrastructure

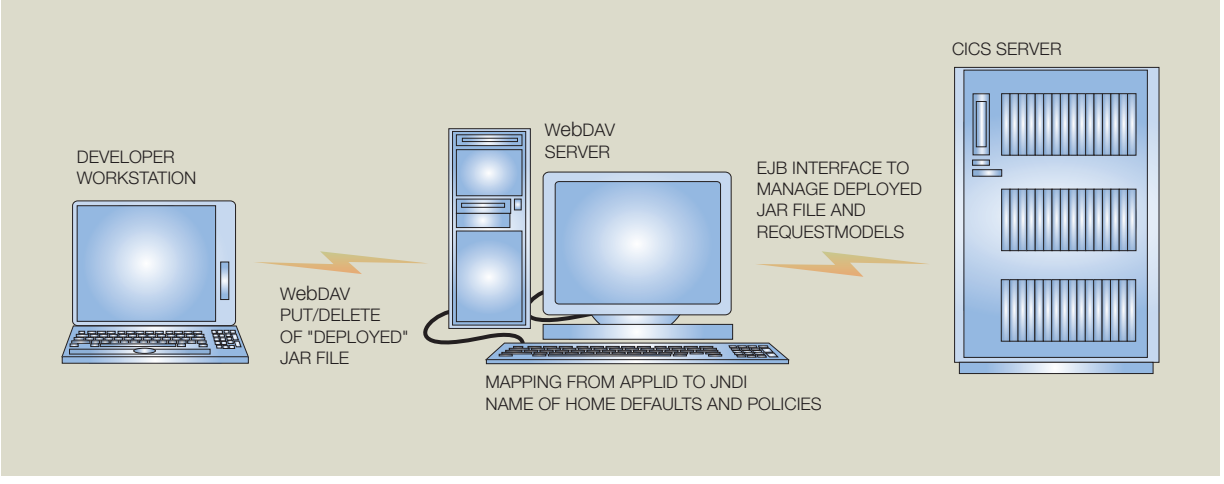
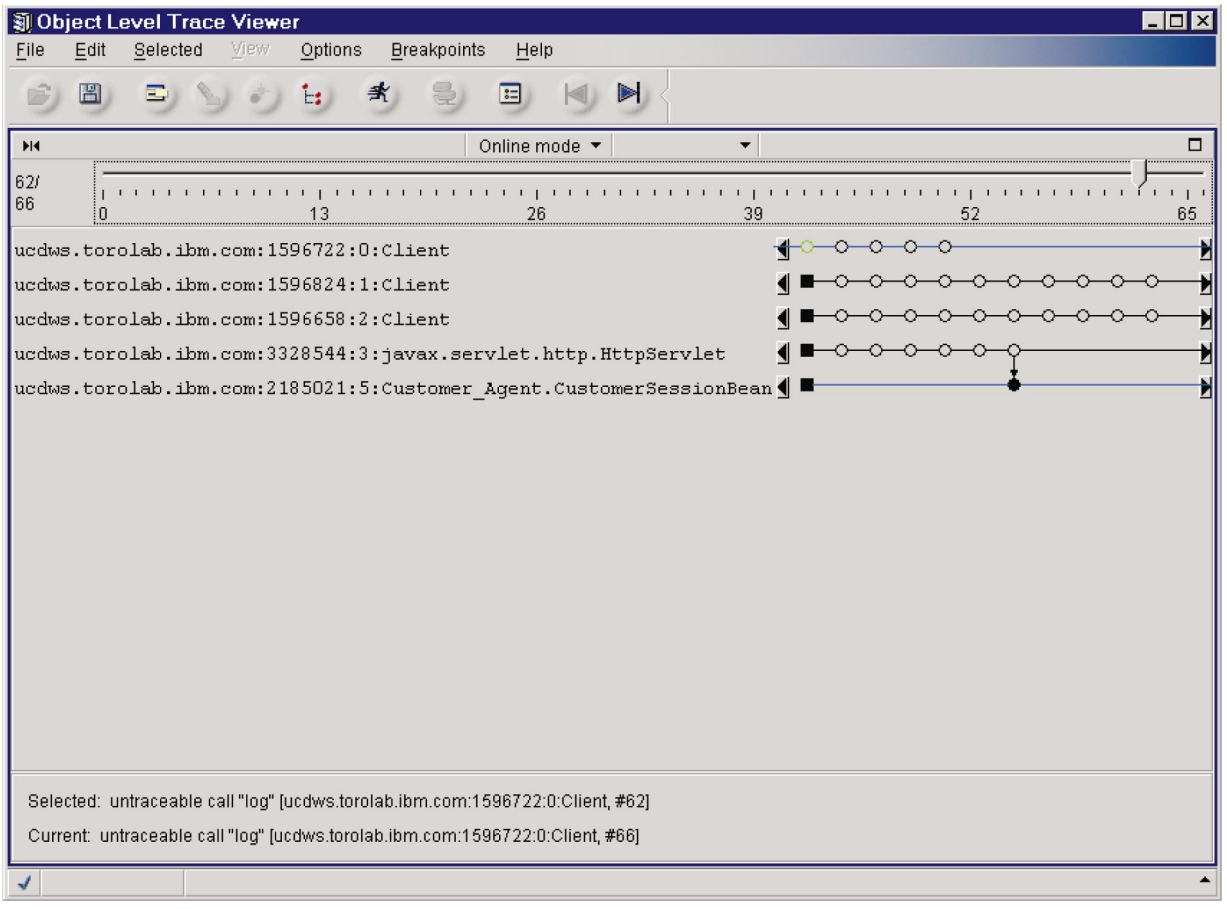


Figure 10 Sample OLT user interface



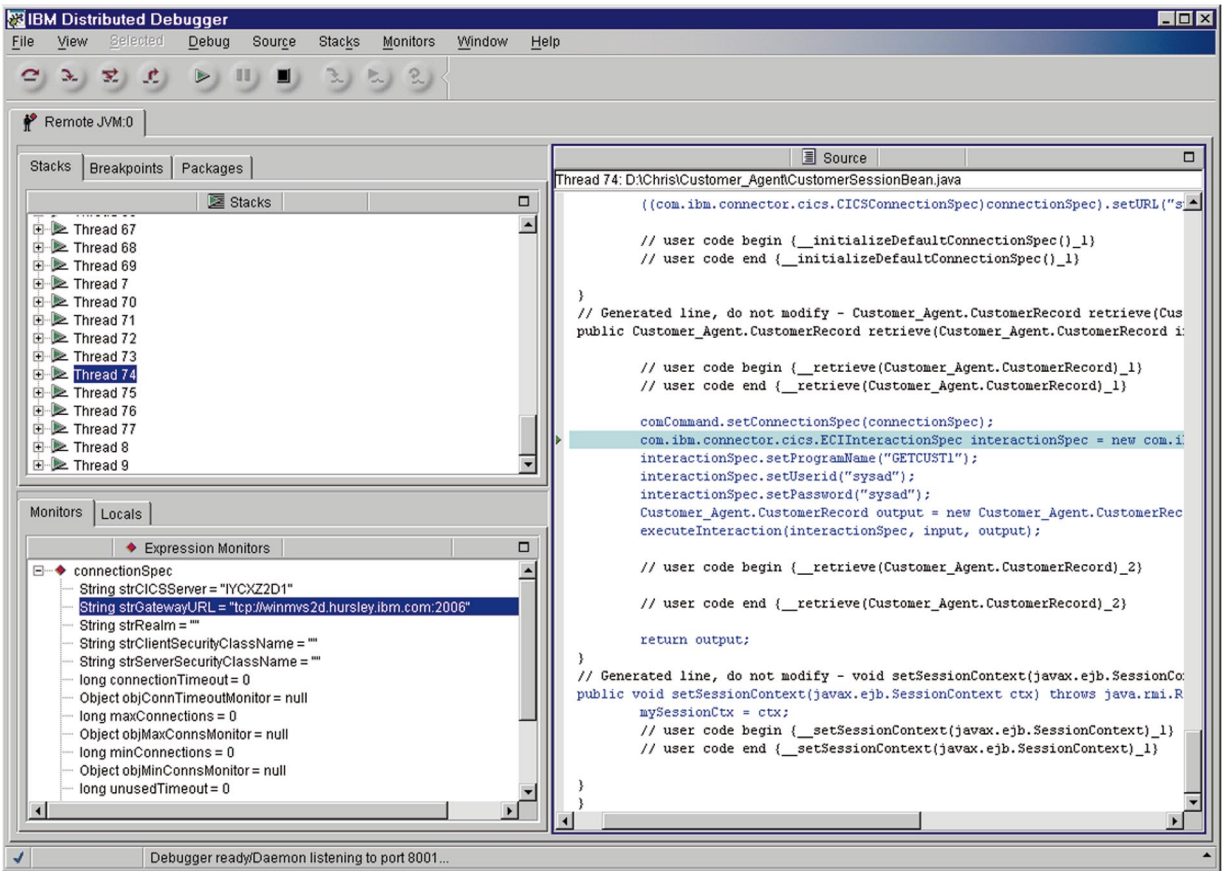
Debugging. The strategic IBM debugging tools for the CICS EJB server are object-level trace and the IBM Distributed Debugger. In addition to support for these IBM tools, CICS will provide appropriate "hooks" to allow for equivalent third-party products to be integrated.

The initial release of EJB support in CICS will offer basic support for debugging using the standard Java debug architecture, Java Platform Debugger Architecture,[22] with the full support for debugging described here being delivered subsequently.

Object-level trace. The object-level trace (OLT) tool allows a developer to follow the flow of an application from the client to a server and from one server to another server. The client and the servers send information about the inbound and outbound method calls to an OLT server, and a graphical tool presents a view of the distributed thread of control made up of all of the (remote) method calls. In addition to visualizing the flow of a distributed application, the OLT tool can be used to control the points in the distributed application at which the IBM Distributed Debugger should be invoked to remotely debug a particular method invocation. This provides much better control of the debugger than the typical server-oriented configuration schemes such as DTCN, a CICS transaction that configures the z/OS debug tool. Figure 10 shows a sample of the OLT user interface.

The information controlling the debugger flows together with an object invocation over IIOP as an extra service context. This information is read and written by request interceptors running in the ORB. The interfaces to allow such request interceptors to be

Figure 11 Debugging a session bean



registered with the ORB are documented so ISVs can register their own request interceptors to flow their own additional information on method calls and support tools similar to OLT.

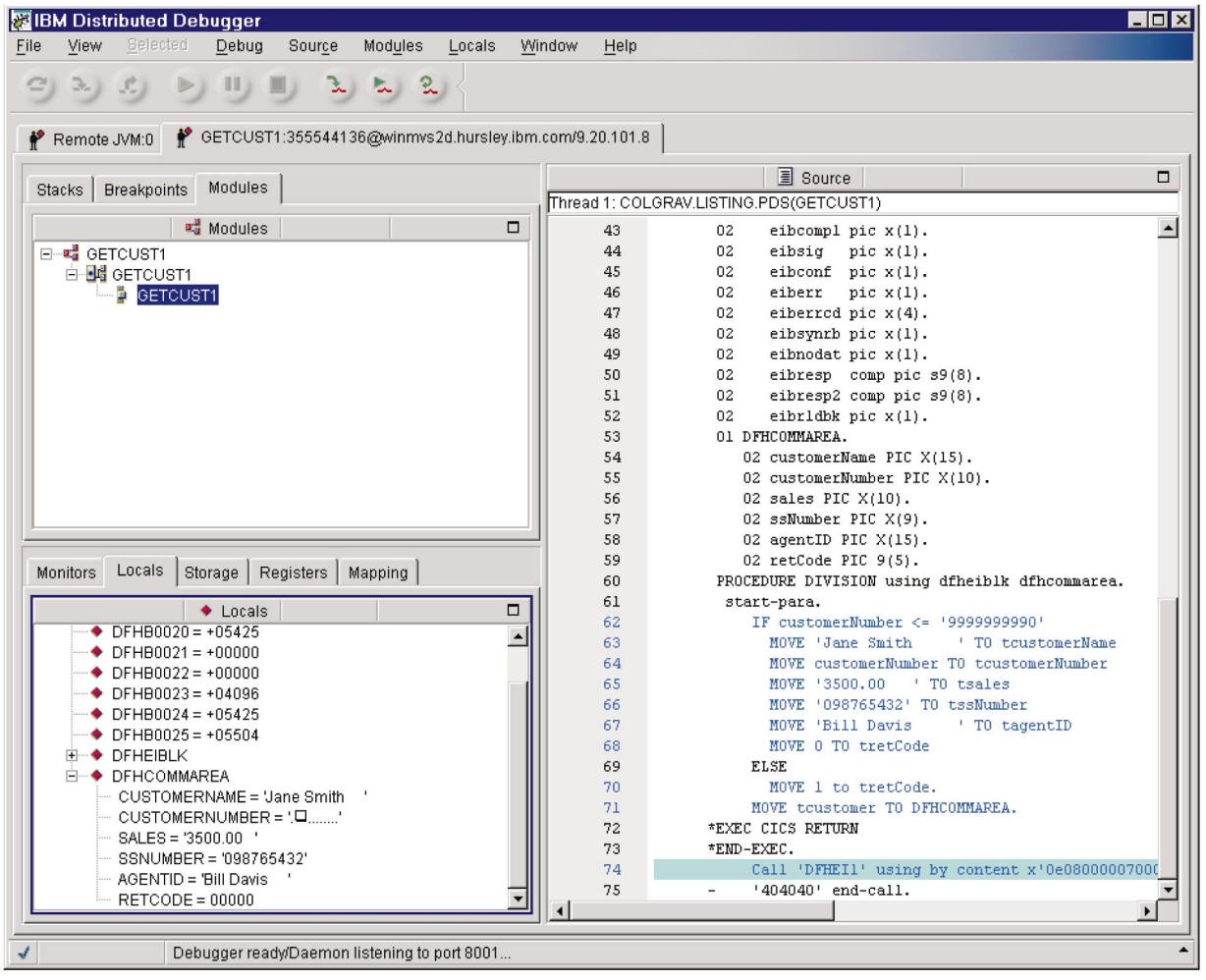
IBM Distributed Debugger. The IBM Distributed Debugger consists of a graphical user interface that communicates with a (remote) debug engine that interacts with the application being debugged. The user interface can display multiple windows, each of which corresponds to a particular debug session with a particular system. This facility can be used to debug, say, a servlet or JSP file in addition to an enterprise bean invoked by the servlet or JSP. The debugger is not limited to debugging Java, it supports debugging code written in C, C++, and COBOL. Figure 11 shows the debugger debugging a session bean.

Figure 12 shows the debugger (running on Windows NT) debugging a CICS COBOL program. Note that the COBOL program is shown in a second window, and the first window showing the session bean is still available so that the developer can switch between the Java code and the COBOL code.

Future work and challenges

One key to the success of the EJB architecture will be the extent to which business logic and the underlying system functions can be separated. By integrating support for enterprise beans into CICS in the manner described, a developer can depend upon the capabilities of a world-class, proven application server platform, without needing detailed knowledge of the facilities of CICS *per se*. The flexibility of the product is such that both "standard" enterprise beans and those that make use of CICS services will be equally well supported. These capabilities are available to customers now and, from day one, will un-

Figure 12 Debugging a CICS COBOL program



doubtedly help make z/OS one of the best platforms for the deployment of enterprise beans.

Nevertheless, there is ample opportunity to further develop the function that will be delivered in the first release so that it evolves: to expand the range of enterprise Java technologies that can be used from enterprise beans deployed in CICS; to add support for broader container-based services that will enrich and complement the "basic" EJB capabilities (such as messaging); and to continually improve the performance and scalability that can be achieved when deploying enterprise beans in CICS. Much of this will be achieved by enabling CICS applications to capitalize on the full range of J2EE services that will be provided by the WebSphere Enterprise Edition product; in a number of cases, it will be possible for CICS to exploit a function that will already have been implemented by WebSphere for the z/OS platform as a whole, instead of having to implement the function in a manner that is specific to CICS itself. By adopting this approach, we will be able to focus future work in CICS on those aspects of J2EE (as we have done with the support for session beans implemented in the current release) that fit most naturally within the architecture of the CICS product, and for which there are clear customer requirements.

A good example of function being implemented by WebSphere, that can be used from CICS, is entity

beans. In future releases, our intention is to integrate the products in such a way as to make the entity bean capability of WebSphere and DB2 available seamlessly to applications running in CICS. Doing so will present a number of challenges, not the least of which is the implementation of a suitably efficient connection between the two environments. Another consideration that will be particularly important when implementing support for entity beans that will, in effect, be distributed across a number of products, will be the loading and managing of persistent state information (including, for example, issues of lazy loading of data, aimed at minimizing the initial cost of accessing an entity bean). And, as with any implementation of entity beans, care must be taken to acquire appropriate locks (which, in turn, depends upon understanding when it is safe to use a read lock as opposed to one that permits data updating).

Another area for continued focus is, of course, performance and scalability. There is no doubt that achieving a level of performance that meets the expectations of today's high-end application server customers will be one of the most important factors in determining the ultimate success of the Java language as an enterprise-class technology for use in high-volume, mission-critical applications. The persistent reusable Jvm technology that is being used by CICS is just one of the results of the significant investments that IBM is making to help ensure that these expectations are met. There are a number of further enhancements that we are already planning to make to this Jvm, subsequent to the first release of the technology that will be used with the first release of the CICS EJB support. For example, one aim is to significantly reduce Jvm initialization time by maintaining a cache of pre-prepared and just-in-time-compiled classes that are referenced when a Jvm is started (instead of being loaded from a file system). Another planned improvement will significantly reduce the amount of "below the 16-megabyte line" virtual storage required by individual Jvms, which will greatly increase the number of concurrent EJB transactions that can be supported concurrently within a single CICS region.

In conclusion, we believe that the combination of what we plan to deliver in CICS, complemented by the full range of J2EE function that will be provided by WebSphere Enterprise Edition and the outstanding application development capabilities of the VisualAge for Java product, will make z/OS *the* premier platform for enterprise Java technologies.

Acknowledgments

The project to implement support for the EJB architecture in CICS has been a major undertaking and would not have been possible without the efforts of many of our colleagues (too numerous to list here) in the CICS organization in Hursley. Two key collaborations have been with the IBM Centre for Java Technology on the persistent reusable Jvm, and with the WebSphere and VisualAge development organizations on the common container and related tooling. Finally, we would acknowledge the work done by our colleagues Ian Brackenbury, Don Ferguson, and Tony Storey on the development of the EJB specification itself, and in establishing the strategy for its exploitation in IBM products.

Cited references

1. IBM CICS Transaction Server, IBM Corporation, http://www.ibm.com/software/ts/cics.
2. CORBA/IIOP 2.3.1 Specification, Object Management Group, http://www.omg.org/technology/documents/formal/corba_2.htm.
3. Enterprise JavaBeans Specification, Sun Microsystems, Inc., http://java.sun.com/products/ejb/docs.html.
4. Java Servlets Specification, Sun Microsystems, Inc., http://java.sun.com/products/servlet.
5. JavaServer Pages (JSP) Specification, Sun Microsystems, Inc., http://www.javasoft.com/products/jsp.
6. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, San Francisco, CA (1992).
7. Microsoft Transaction Server, Overview, Microsoft Corporation, http://www.microsoft.com/NTServer/appservice/exec/overview/Trans_Overview.asp.
8. CORBA Component Model RFP, Object Management Group, http://www.omg.org/techprocess/meetings/schedule/CORBA_Component_Model_RFP.html.
9. IBM VisualAge for Java, IBM Corporation, http://www.ibm.com/software/ad/vajava.
10. Java Naming and Directory Interface (JNDI), Sun Microsystems, Inc., http://java.sun.com/products/jndi.
11. Java 2 Platform, Enterprise Edition, Sun Microsystems, Inc., http://java.sun.com/j2ee/.
12. Java Transaction Service, Sun Microsystems, Inc., http://java.sun.com/j2ee/transactions.html.
13. Java Message Service, Sun Microsystems, Inc., http://java.sun.com/products/jms/.
14. JavaBeans Specification, Sun Microsystems, Inc., http://java.sun.com/beans/docs/spec.html.
15. Extensible Markup Language (XML), World Wide Web Consortium, http://www.w3.org/XML/.
16. JCICS Programing Interface, IBM Corporation, http://www.4.ibm.com/software/ts/cics/library/manuals/jcics/.

17. Java Database Connectivity (JDBC) API, Sun Microsystems, Inc., http://java.sun.com/products/jdbc.
18. CORBA Transaction Service Specification, Object Management Group, http://www.omg.org/technology/documents/formal/transaction_service.htm.
19. SQLJ, SQLJ standards information site, http://www.sqlj.org.
20. J2EE Connector Architecture 1.0 Specification, Sun Microsystems, Inc., http://java.sun.com/j2ee/connector/.
21. Web-based Distributed Authoring and Versioning, WebDAV community site, http://www.webdav.org/.
22. Java Platform Debugger Architecture, Sun Microsystems, Inc., http://java.sun.com/products/jpda/.

Accepted for publication September 22, 2000.

Andrew Bainbridge *IBM UK Laboratories, Hursley Park, Winchester, Hampshire, SO21 2JN, United Kingdom (electronic mail: Andrew_Bainbridge@uk.ibm.com).* Mr. Bainbridge is a Senior Technical Staff Member in the IBM Software Group at the Hursley Development Laboratory in the United Kingdom. He is currently manager for design and new technology development in the Transaction Processing Products organization. He joined IBM as a systems engineer in 1984 and has worked on a wide range of software development projects and standards initiatives in the areas of transaction processing and open systems networking; over the last few years, he has focused on the integration of transaction processing and object technologies, working on the CICS and WebSphere products. He studied mathematics and computer science at Cambridge University, graduating in 1984.

John Colgrave *IBM UK Laboratories, Hursley Park, Winchester, Hampshire, SO21 2JN, United Kingdom (electronic mail: colgrave@uk.ibm.com).* Mr. Colgrave is a senior software engineer in the Transaction Processing Products Design and New Technology department at the IBM Hursley Development Laboratory. For over ten years, he has worked as an architect and designer on CICS on various platforms; for several years he has focused on the use of Java in CICS and, most recently, the associated requirements for application development tooling. He holds a B.S. degree in electronic and electrical engineering and an M.S. degree in computer science, both from Manchester University.

Adrian Colyer *IBM UK Laboratories, Hursley Park, Winchester, Hampshire, SO21 2JN, United Kingdom (electronic mail: adrian_colyer@uk.ibm.com).* Mr. Colyer received his degree in computer science from the University of Southampton, England, in 1992. He is a Chartered Engineer and Corporate Member of the Institution of Electrical Engineers (IEE) and Honorary Membership Secretary of the UK Research and Development Society. Since joining IBM in 1992 he has worked on a number of middleware-related development projects involving CICS and MQSeries, and also on emerging technology projects in e-business and pervasive computing. In his current position Mr. Colyer is a senior software engineer working on enterprise Java support for CICS.

Glyn Normington *IBM UK Laboratories, Hursley Park, Winchester, Hampshire, SO21 2JN, United Kingdom (electronic mail: norm@uk.ibm.com).* Mr. Normington is a senior software engineer working on CICS. He received a B.A. degree in mathematics from Oxford University in 1981 and, after joining IBM, worked initially on graphics software and print spooling. Later he joined the CICS area and worked on the restructuring of some of the central components of CICS. He then led the Component Broker workload management design and implementation before he returned to CICS to lead the design of the support for EJB.