SYBASE®

Feature Guide

EAServer

6.0

DOCUMENT ID: DC38033-01-0600-01

LAST REVISED: July 2006

Contents

About This Book

Subject

This book describes EAServer, which is an integrated set of application servers that you use to deploy Web applications that support high-volume traffic, dynamic content, and intensive online transaction processing. The EAServer product set includes Adaptive Server® Anywhere.

Note Products described in this manual may not be available on some UNIX platforms. See the release bulletin and the installation guide for a list of products included in your EAServer edition.

Audience

This book is for new users of EAServer.

How to use this book

The following chapters are included in this book:

- Chapter 1, "Introducing EAServer," includes a description of EAServer features and use.
- Chapter 2, "Developing an EAServer Application," explains some of the basic concepts and terminology associated with developing component-based EAServer applications.
- Chapter 3, "EAServer Components," discusses developing EJB, Java, and C/C++ components that run on EAServer.
- Chapter 4, "Web Applications," describes the components of Web applications.

Related documents

Core EAServer documentation The core EAServer documents are available in HTML and PDF format in your EAServer software installation and on the SyBooks™ CD.

What's New in EAServer 6.0 summarizes new functionality in this version.

The *EAServer API Reference Manual* contains reference pages for proprietary EAServer Java classes and C routines.

The *EAServer Automated Configuration Guide* explains how to use Ant-based configuration scripts to:

- Define and configure entities, such as EJB modules, Web applications, data sources, and servers

- Perform administrative and deployment tasks

The *EAServer CORBA Components Guide* explains how to:

- Create, deploy, and configure CORBA and PowerBuilder™ components and component-based applications
- Use the industry-standard CORBA and Java APIs supported by EAServer

The *EAServer Enterprise JavaBeans User's Guide* describes how to:

- Configure and deploy EJB modules
- Develop EJB clients, and create and configure EJB providers
- Create and configure applications clients
- Run the EJB tutorial

The *EAServer Feature Guide* explains application server concepts and architecture, such as supported component models, network protocols, server-managed transactions, and Web applications.

The *EAServer Java Message Service User's Guide* describes how to create Java Message Service (JMS) clients and components to send, publish, and receive JMS messages.

The *EAServer Migration Guide* contains information about migrating EAServer 5.*x* resources and entities to an EAServer 6.0 installation.

The *EAServer Performance and Tuning Guide* describes how to tune your server and application settings for best performance.

The *EAServer Security Administration and Programming Guide* explains how to:

- Understand the EAServer security architecture
- Configure role-based security for components and Web applications
- Configure SSL certificate-based security for client connections
- Implement custom security services for authentication, authorization, and role membership evaluation
- Implement secure HTTP and IIOP client applications
- Deploy client applications that connect through Internet proxies and firewalls

The *EAServer System Administration Guide* explains how to:

- Start the preconfigured server and manage it with the Sybase Management Console
- Create, configure, and start new application servers
- Define database types and data sources
- Create clusters of application servers to host load-balanced and highly available components and Web applications
- Monitor servers and application components
- Automate administration and monitoring tasks with command line tools

The *EAServer Web Application Programming Guide* explains how to create, deploy, and configure Web applications, Java servlets, and JavaServer Pages.

The *EAServer Web Services Toolkit User's Guide* describes Web services support in EAServer, including:

- Support for standard Web services protocols such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI)
- Administration tools for deployment and creation of new Web services, WSDL document creation, UDDI registration, and SOAP management

The *EAServer Troubleshooting Guide* describes procedures for troubleshooting problems that EAServer users may encounter. This document is available only online; see the EAServer Troubleshooting Guide at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_5.2.eastg/html/eastg/title.htm.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ 6.0.5 driver to allow JDBC access to Sybase database servers and gateways. The *jConnect for JDBC 6.0.5 Programmer's Reference* is available on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.jconnjdbc_6.05.prjdbc/html/prjdbc/title.htm&toc=/com.sybase.help.jconnjdbc_6.05/toc.xml.

Sybase Software Asset Management User's Guide EAServer includes the Sybase Software Asset Management license manager for managing and tracking your Sybase software license deployments. The *Sybase Software Asset Management User's Guide* is available on the Getting Started CD and in the EAServer 6.0 collection on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_6.0/title.htm.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
commands and methods	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text • Property names in the raw format, as when using Ant or jagtool to configure applications rather than the Management Console
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: `Server.log` (*Server* in italics) • File names • Names of components, EAServer packages, and other entities that are registered in the EAServer naming service
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates “select Save from the File menu.”
`package 1`	Monospace font indicates: • Information that you enter in the Management Console, a command line, or as program text • Example program fragments • Example output fragments

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com/nav/base.do.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list and click Go.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Accessibility features

EAServer has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in Eclipse help formats, which you can navigate using a screen reader.

The Web console supports working without a mouse. For more information, see "Keyboard navigation" in Chapter 2, "Management Console Overview," in the *EAServer System Administration Guide*.

The Web Services Toolkit plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired, or have other special needs. For information about these features see the Eclipse help:

1 Start Eclipse.

2 Select Help | Help Contents.

3 Enter `Accessibility` in the Search dialog box.

4 Select Accessible User Interfaces or Accessibility Features for Eclipse.

Note You may need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1 Introducing EAServer

This chapter describes the EAServer features.

For a description of the features that are new in this version, see *What's New in EAServer*.

Overview

EAServer is an application server, which includes an integrated set of development tools that you use to deploy Web applications that support high-volume traffic, dynamic content, and intensive online transaction processing. The EAServer product set includes Adaptive Server Anywhere.

EAServer provides a framework for deploying the middle-tier logic of distributed component-based applications.

EAServer simplifies the creation and administration of Internet applications that service thousands of clients simultaneously. EAServer components execute on the middle-tier between end-user client applications and remote databases. EAServer provides efficient management of client sessions, security, threads, third-tier database connections, and transaction flow, without requiring specialized knowledge on the part of the component developer.

EAServer scalability and platform independence allow you to develop your application on inexpensive single-processor machines, then deploy the application on an enterprise-grade multiprocessor server.

EAServer provides:

- A modular, lightweight, high-performance J2EE container based on a dynamic Java-components framework for executing standard J2EE components
- Dispatch and stub/proxy support for all major component models, including JavaBeans, PowerBuilder, Java, and C/C++
- Dynamic HTML support using Java servlet and JavaServer Pages
- Java 2 Enterprise Edition (J2EE) platform support
- Graphical administration with the Management Console, including component interface browsing, declarative role-based security, password, and required SSL session characteristics, server and user certificate management, and transaction and runtime monitoring
- Tight integration with the PowerBuilder development environment
- Transparent client-session and component-lifecycle management
- Connection caching to allow reuse of remote database connections
- Industry-standard naming services to resolve components using logical names rather than server addresses
- Transaction management to simplify the design and implementation of an application’s transactions

- Transparent thread-safety features to simplify use of shared data and resources
- Result-set support to enable efficient retrieval of tabular data in client applications
- Declarative, role-based security to restrict client connections and the components that can be invoked by a specific client session
- Identity-based security to restrict intercomponent calls
- Asynchronous messaging support
- Asynchronous processing support
- Web server redirector plug-in forwards client requests directly to a Web server
- EAServer provides plug-ins for Eclipse and Borland JBuilder version 10. The EAServer plug-ins conform to Section 508 guidelines, and are internationalized and localizable.

The following sections explain these features in detail.

EAServer architecture

The EAServer J2EE container is based on a dynamic Java-components framework, which provides a modular, high-performance container with a small footprint for executing standard J2EE components. The container permits the execution of J2EE Web applications, including EJB components and the JMS API, on mobile devices such as a handheld or tablet PC running Windows XP. The core packages are 100% Java and portable to all JDK 1.4+ platforms. Native code is used only for optional modules; for example, operating system login modules, and for supporting CORBA and PowerBuilder components.

EAServer provides a scalable and platform-independent environment for the execution of Web and component-based applications. EAServer is scalable because it is multithreaded and multiprocessor safe.

The EAServer execution environment is the same across all platforms.

EAServer provides the following services:

- Network listeners for the connections on which clients send remote component invocations. EAServer core network server technology is based on Sybase Open Client/Server™ technology. “Network protocol support” on page 7 details the supported application protocols.
- An execution environment for middle-tier components. See “Server-side component support” on page 27.
- A built-in HTTP server. You can use EAServer HTTP support to deploy your application’s Java applets and HTML pages.
- Ability to run with different Java virtual machines.
- Declarative security. You can use roles to authenticate and authorize users. See the *EAServer Security Administration and Programming Guide*.
- Data source connection caching. You can define data source connections for interacting with remote databases from EAServer components. See “Data source connections” on page 17.
- Task scheduling. You can schedule tasks to be performed automatically, based on either time or the completion of another task.

In addition to these built-in services, you can install *service components* that run in the background and provide customized services to clients or other components. See “Services tab” in Chapter 3, “Creating and Configuring Servers,” in the *EAServer System Administration Guide*.

Component support

Components are reusable modules of code that combine related tasks (methods) into a well-defined interface. EAServer components are installed on an EAServer application server and contain the methods that execute business logic and access data sources. You or your administrator install the component’s executable code on the server. Components can be distributed throughout a network, including the Internet or an intranet, on different servers. Installed components can be used by any number of independent applications.

Since EAServer components reside on the server, components do not contain methods to display graphics or user interfaces—that is, EAServer components are inherently nonvisual.

User-interface developers or other component developers can browse a component’s interface in the Management Console; in their code, they use a client stub or proxy to invoke the component’s methods. The stub or proxy acts as a local surrogate for the remote component, providing the same method signatures as the component and hiding the details of remote server communication.

The EAServer server-side component support and client-side stub or proxy support are independent. Any EAServer client can execute any type of component. A component of any model can execute components of another model using intercomponent calls without the use of additional gateway software. Additionally, since EAServer uses standard CORBA IIOP as its core network protocol, you can use CORBA client runtimes from other vendors to invoke components installed on EAServer.

All clients and components share a common interface repository. Component interfaces are stored in standard CORBA interface definition language (IDL). Component developers can define, edit, and browse interfaces in the Management Console, which allows you to edit interfaces graphically or as raw IDL. You can also define interfaces by importing compiled Java classes and standard-format EJB-JAR files.

For more information about the component models that EAServer supports, see Chapter 3, “EAServer Components.”

J2EE platform support

EAServer implements the Java 2 Enterprise Edition (J2EE) 1.4 specification, with support for EJB 2.1 components, J2EE applications, J2EE Web applications, object caching, the JavaMail 1.3.2 electronic mail API, the connector architecture, Java API for XML Parsing, and Java Authentication and Authorization Services.

“Component support” on page 4 describes EAServer-supported component models, including EJB.

A Web application is a unit of deployment for interrelated Web content, JavaServer Pages (JSPs), and Java servlets. The Web application contains static files, servlet and JSP implementation classes, and a deployment descriptor that describes how the files, servlets, and JSPs are configured on the host server. See Chapter 1, “Defining Web Applications,” in the *EAServer Web Application Programmer’s Guide*.

J2EE applications allow you to group related EJB 2.1 components and Web applications into a single entity. This enables you to deploy related business logic components, Java servlets, JavaServer Pages, and Web pages as a single unit between servers. Using the J2EE Application Client model, you can create clients that call the components and Web pages in the application. For more information, see these chapters in the *EAServer EJB User's Guide*:

- Chapter 3, "Developing EJB Clients"
- Chapter 5, "Interoperability"

EAServer supports EJB 2.1 container-managed persistence (CMP) for EJB entity bean components, as well as supporting an automatic persistence model for components of other types. EAServer also supports entity instance and finder-query caching, which can improve performance by minimizing the number of database select queries required to execute business logic. For information on entity components, automatic persistence, and object/query caching, see Chapter 2, "Deploying and Configuring EJB Components," in the *EAServer EJB User's Guide*.

The JavaMail API provides a standard Java interface to the most widely-used Internet mail protocols—see Chapter 5, "Creating JavaMail," in the *EAServer Web Application Programmer's Guide*.

The J2EE connector architecture enables you to write portable Java applications that can access multiple transactional enterprise information systems. A connector is a specialized connection factory that provides connections for EJBs, Java servlets, JSPs, and CORBA-Java components. For more information, see Chapter 4, "Database Access," in the *EAServer System Administration Guide*.

EAServer can host Web applications in popular Web servers such as Apache, iPlanet, and Netscape—see the *EAServer Installation Guide* for your platform.

EAServer implements J2EE version 1.4 security requirements including Java and C++ ORB support and CORBA Secure Interoperable version 2 protocol (CSIv2). See the *EAServer Security Administration and Programming Guide*.

EAServer includes support for the Java API for XML Parsing 1.1. You can configure the parser and transformer implementations for servers, components, Web applications, and application clients.

Java Authentication and Authorization Services (JAAS) provide a framework and standard programming interface for authenticating users and assigning privileges. JAAS is based on the Pluggable Authentication Module standard, which extends the access-control architecture of the Java 2 platform to support user-based authentication and authorization. See the *EAServer Security Administration and Programming Guide*.

Network protocol support

EAServer supports the following protocols:

- **Internet Inter-ORB Protocol (IIOP)** IIOP is the standard protocol for communication between CORBA ORBs over TCP/IP networks. All EAServer client models use IIOP or IIOP tunnelled inside of SSL (referred to as IIOPS). IIOP connections can also be tunnelled inside of HTTP to allow connections through firewalls that do not allow passage of IIOP traffic, as discussed in “HTTP tunnelling support” on page 8.
- **Sybase Tabular Data Stream™ (TDS)** TDS is a proprietary protocol used in two-tier database applications that connect to Sybase database servers or gateways. If you have converted an Open Server™ application to run in EAServer, legacy clients for the application connect to EAServer using TDS. You must define an Open Server listener to support legacy Open Server clients.
- **Hypertext Transfer Protocol (HTTP)** HTTP is used by Web browsers for file downloads and uploads. EAServer provides HTTP support to allow you to deploy HTML pages and Java applets on EAServer itself.
- **Secure Sockets Layer (SSL)** The SSL protocol allows connections to be secured using public-key encryption and authentication algorithms. “SSL support” on page 8 describes EAServer SSL support.

To enable support for each protocol, you must define a *listener* in the Management Console—see “Configuring listeners” in the *EAServer System Administration Guide*. The listener configuration specifies a server address (host name and port number) as well as the network protocol and security settings to be used by clients that connect to that listener. SSL support requires installation of a server certificate. See the *EAServer Security Administration and Programming Guide*.

HTTP 1.1 support

EAServer supports HTTP/1.1, and complies with the features required for *origin server* and *client*. See the HTTP/1.1 specification at http://www.w3.org/Protocols/rfc2616/rfc2616.html.

HTTP tunnelling support

Almost all network firewalls allow HTTP traffic to pass, but some reject IIOP packets. When IIOP traffic is tunnelled inside of HTTP, your clients can connect to EAServer through a firewall that does not allow IIOP traffic to pass.

The EAServer Java client ORB performs HTTP tunnelling automatically using the designated IIOP port. No additional configuration or proxies are required. When connecting, the EAServer client-side ORB first tries to open an IIOP connection to the specified address and port. If the IIOP connection fails, the ORB tries an HTTP-tunnelled connection to the same address and port. The default behavior is appropriate when some users connect through firewalls that require tunnelling and others do not; the same application can serve both types. If you know HTTP tunnelling is always required for a Java client, you can set the ORBHttp property to cause the ORB to use HTTP tunnelling without trying plain IIOP connections first.

The C++ client ORB supports tunnelling when clients explicitly request it by setting the ORBHttp property.

SSL support

The SSL protocol allows connections to be secured using public-key encryption and authentication algorithms that are based on digital certificates. SSL is a *wrapper* protocol: packets for another protocol are secured by embedding them inside of SSL packets. For example, HTTPS is HTTP-secured by embedding each HTTP packet within an SSL packet. Likewise, IIOPS is IIOP embedded within SSL. HTTPS and IIOPS are also commonly called *secure HTTP* and *secure IIOP*, respectively.

EAServer provides native SSL protocol support. Specifically, the EAServer built-in SSL driver supports dynamic negotiation, cached and shared sessions, and authorization for client and server using X.509 Digital Certificate Support.

The Java Secure Socket Extension (JSSE) is a set of Java packages that implements SSL and Transport Layer Security (TLS). SSL and TLS enable data encryption, server authentication, message integrity, and client authentication.

In the Management Console, you can configure a secure IIOP or HTTP port by defining an IIOP or HTTP listener, then associating a security profile with the listener. The security profile designates a server certificate that is sent to clients to verify that the connection ends at the intended server. The security profile also specifies the connection’s required security settings, such as:

- Whether a client certificate is required to open connections. The client certificate serves as proof of the client user's identity.
- What data security options, such as the encryption algorithm, are used to secure data transmitted over the connection.

For detailed instructions on configuring secure ports, see the *EAServer Security Administration and Programming Guide*.

On the client-side, the following types of clients can open SSL connections to EAServer:

- Java applets hosted by SSL-capable Web browsers.
- Java applications
- C++ clients
- PowerBuilder clients

Dynamic Web content

You can use Java servlets and JSPs to dynamically create HTML pages and interactive HTML forms. Java servlets can call methods in EAServer components.

Java servlets are Java classes that use the standard javax.servlet API to respond to HTTP requests. EAServer can host Java servlets natively, and you can use the Management Console to associate logical paths in an HTTP URL with a Java servlet. JavaServer Pages extend the HTML page description language, allowing you to embed Java scriptlets within HTML tags. See the *EAServer Web Application Programmer's Guide* for more information:

- Chapter 2, "Creating Java Servlets"
- Chapter 4, "Creating JavaServer Pages"

Using the PowerBuilder integrated development environment (IDE), you can deploy JSP-based Web applications from PowerBuilder to EAServer. See *Working with Web and JSP Targets* in your PowerBuilder documentation set.

Web server redirector plug-in

A Web server redirector plug-in enables communication between a third-party Web server and an EAServer HTTP or HTTPS listener. A redirector plug-in forwards requests that need to access servlets, JSPs, and so on, to EAServer. EAServer processes the requests and returns the results back to the Web server. You can also use a redirector plug-in to implement load balancing and failover. EAServer includes redirector plug-ins for a number of Web servers, such as Apache and Microsoft Internet Information Services (IIS). See the *EAServer Installation Guide* for information about how to install and configure the Web server redirector plug-ins.

Administration and development tools

The Management Console is a Web-based graphical user interface that provides administration facilities for EAServer, including support for deployment and runtime monitoring of applications.

For detailed instructions on running the Management Console, see the *EAServer System Administration Guide*.

Development support

PowerBuilder has been integrated with EAServer. This allows you to develop, deploy, and debug EAServer components entirely within the development environment. You can also generate the proxies required for client application development. For more information, see the *Application Techniques* manual in the PowerBuilder documentation.

Application developers using other tools can use the Management Console to view the method definitions for any installed component in EAServer. You can view and edit method definitions graphically, or you can directly edit the IDL datatype and interface definitions with the Management Console IDL editor. You can import interface definitions from existing Java classes or from standard CORBA IDL files. See “Defining Component Interfaces” in the *EAServer CORBA Components Guide* for more information.

Stubs are generated automatically and compiled when first used, and regenerated and recompiled if a component’s remote interface changes.

Deployment support

To simplify application deployment, the Management Console defines the following basic, middle-tier application units:

- **Clusters** A cluster represents a set of servers that share configuration information and run the same set of components. For applications with thousands of clients, clusters provide support for load balancing and high availability. When you create a cluster, two export configurations are created automatically, one that allows you to replicate component files and configuration information from a primary server to other servers in the cluster, and another that allows you to replicate security information.
- **Servers** A server represents one EAServer runtime process. Each server has its own network addresses for client session connections and for HTTP (HTML) connections. All servers on one host machine share the same configuration repository. For administration purposes, you can connect to any server on the host machine to configure other servers on the same host.
- **Customized deployment** You can create export configurations to bundle a set of files, Java classes, and entities (application clients, applications, connectors, EJBs, and Web applications) into a single unit for easy deployment between servers.
- **Applications** Applications allow you to group packages (groups of related components) and Web applications (bundled static Web content, servlets, and JSPs) into a single unit for easy deployment between servers.
- **Packages** A package organizes components into cohesive, secure units that can be easily deployed on another EAServer installation.

  Roles attached to packages control access to components in a package. For more information about roles and package security, see the *EAServer Security Administration and Programming Guide*.
- **Components** A component definition consists of the component's method signatures and other properties, such as component type, transaction support, threading model, and the name of the Java class or executable library that implements the component.
- **Web applications** Web applications allow you to group static Web content, servlets, and JSPs into a single unit for easy deployment and configuration.
- **Web components** You can install servlets and JSPs in a Web application as Web components. This allows you to configure request path mappings and other useful settings within the Web application properties.

Before a client application can execute a component, the component must installed in the server to which the client connects.

Hot refresh support

The Management Console includes an option that allows you to refresh components, packages, and servers. This enables you to test and debug component implementation changes without restarting the server.

Performance threshold monitoring support

EAServer includes a threshold monitor feature that can help to prevent degradation of server performance under extreme load conditions. You can configure the threshold monitor settings to heuristically govern the processing requests to prevent performance degradation due to overuse of available resources. For details, see “Threshold monitor settings” in Chapter 2, “Server Tuning,” in the *EAServer Performance and Tuning Guide*.

Runtime monitoring support

The Management Console allows you to remotely view server log files and to monitor statistics for component execution, transactions, data sources, and thread activity.

You can use the Management Console to connect to EAServer and view the contents of:

- EAServer log file, which records errors related to component execution.
- HTTP request log, which records successful and unsuccessful file requests.
- RMI-IIOP log, which traces remote method invocations over IIOP.
- JMS log, which traces Java Message Service (JMS) operations in the server. All public and protected JMS provider methods are traced.
- SQL log, which traces JDBC driver activity in the server, including JDBC prepared statement operations, parameter and result information for container-managed persistence operations (queries and updates), and transaction commit/rollback operations.

Runtime monitoring allows you to view statistics on component and network activity. You can view counts of active client sessions, components, and transactions.

See the *EAServer System Administration Guide* for more information.

Certificate management support

The Management Console enables you to manage the server and user certificates that are required for SSL-protocol support, allowing you to:

- **Install server certificates** Server certificates are required to establish secure IIOP and HTTP ports. The certificate is presented to the client application as proof that the application has connected to the server that the user intends to interact with.

- **Install certification authority (CA) certificates** CA certificates, also called **signing certificates**, are attached to client and server certificates to validate the origin of the certificate. For example, if you obtain a certificate from VeriSign, your certificate includes a copy of the VeriSign CA certificate. Using the Certificates folder, you can install CA certificates into EAServer and indicate which CAs are trusted. When client applications present certificates to EAServer, the signing certificate must be present in the list of trusted CAs or the connection fails.
- **Issue certificates for testing purposes** The command line tool keytool allows you to manage key pairs and certificates for EAServer, for use in testing your applications. To deploy Internet applications, you must obtain server certificates from a well-known CA such as VeriSign or Thawte.

See the *EAServer Security Administration and Programming Guide* for more information on SSL certificates.

Ant based configuration

EAServer includes built-in Ant support and uses Ant build files to perform many administrative and deployment tasks. You can run Ant configuration scripts to define and configure entities such as EJB modules, Web applications, data sources, servers, and so forth.

EAServer generates Ant configuration scripts when you deploy J2EE archives. The generated script applies settings from the archive's deployment descriptor to the new EAServer entities. You can also embed a user configuration file in J2EE archives to configure settings beyond those that can be specified by the deployment descriptor.

Entity types that are not deployed in J2EE archives can also be created or managed with configuration scripts. For example, you can create data sources, restart servers, and manage security roles and domains.

For more information, see Chapter 2, "Ant-Based Configuration," in the *Automated Configuration Guide*.

jagtool and jagant

jagtool is a command line interface that allows you to automate some EAServer development and deployment tasks. jagant lets you run jagtool commands from Jakarta Ant build files.

Ant is similar to make, but is platform-independent, and allows you to incorporate jagtool commands into build files. This powerful feature allows you to write build files that automate many development and deployment tasks. See Chapter 6, "Using jagtool and jagant," in the *Automated Configuration Guide*.

Command line tools

Command line tools allow you to perform many administrative tasks in a command window; for example, starting and stopping a server, setting passwords, configuring and compiling components, and deploying EJB modules. See Chapter 12, “Command Line Tools,” in the *EAServer System Administration Guide*.

Client-session and component life cycle management

EAServer client sessions are established internally by the client stubs and proxies that applications use to invoke EAServer component methods. A component’s lifecycle determines how instances are allocated, bound to client sessions, and destroyed. EAServer manages both client sessions and component life cycles without requiring any specialized knowledge on the part of the application developer.

Client-session management

Client applications use a stub or proxy object to invoke methods on EAServer. Internally, the stub or proxy object establishes a network connection between EAServer and a remote client. All the stub/proxy models discussed in “Client stub and proxy support” on page 28 require user-authentication parameters to instantiate a stub or proxy object. The communication protocol is also determined when the stub or proxy object is instantiated. Once the stub or proxy object exists, all details of network communication are hidden from the application developer.

The user name determines which components a client session can access. If a component does not allow access to the user, attempts to instantiate a proxy or stub fail. The *EAServer Security Administration and Programming Guide* describes security support in detail.

All stubs and proxies use IIOP to communicate with EAServer. “Network protocol support” on page 7 discusses client protocols in detail.

Component life cycle management

A component’s lifecycle determines how instances of a component are allocated, bound to specific client sessions, and destroyed. In the simplest case, an instance is allocated for each stub or proxy created by the client and is destroyed when the client explicitly requests destruction or when it disconnects, whichever happens first.

You can code more sophisticated components to support instance pooling. Instance pooling allows EAServer to maintain a cache of component instances and bind them to client sessions on an as-needed basis. Instance pooling requires the following changes to your component:

- The component must provide activate and deactivate methods. EAServer calls the activate method immediately before an instance is bound to a client session. activate must be able to reset the component to an as-allocated state. EAServer calls deactivate immediately before an instance is unbound from a client session (that is, made idle again).
- Methods in the component must use the EAServer transaction state primitives to request early deactivation.

For components that support EAServer transactions, the time between EAServer activate and deactivate calls coincides with the beginning and end of that instance's participation in an EAServer transaction.

You can increase the scalability of your application by using components that support instance pooling. Instance pooling eliminates execution time and memory consumption that would otherwise be spent allocating unnecessary component instances.

Chapter 2, "CORBA Component Life Cycles and Transaction Semantics," in the *EAServer CORBA Components Guide* describes how components participate in instance pooling and EAServer transactions.

Load balancing and failover

In the simplest scenario, your application's components are deployed to one server, and all clients connect to that server to execute the component's business logic. This scenario works well when the number of simultaneous clients is not too large.

For applications with thousands of clients, you can define an EAServer cluster with several redundant servers to run the application's components. The cluster allows load balancing and failover as follows:

- **Load balancing** EAServer can use load balancing, which optimizes performance for your EAServer cluster by adjusting the load across the servers, based on load metrics and a distribution policy, or based on a random algorithm. When a client resolves the name of an EAServer component, the name server returns several candidate server addresses. When you allow the algorithm to randomly distribute the processing load over servers in the cluster, the client ORB tries the addresses in random order. When you specify the load metrics and distribution policy, the load is distributed according to each server's current load.
- **Failover** You can configure components to allow automatic failover that is transparent to client applications. If a component allows automatic failover, the client ORB automatically reconnects to another server within the cluster when a previously connected server goes offline.

In-memory failover support allows component state to be maintained on a pair of servers, without incurring the overhead of using a remote database to store component state. For more information, see:

- Chapter 8, “Load Balancing, Failover, and Component Availability,” in the *EAServer System Administration Guide*
- *EAServer CORBA Components Guide*
- *EAServer EJB User’s Guide*

Naming services

When multiple servers are involved in your application, the naming service allows you to specify logical server names rather than server addresses. For example, instead of connecting to your finance component server at host badger using port 9000, you can specify the initial naming context for that server, such as *USA/MyCompany/FinanceServer*. Components are identified by specifying an initial server name context plus the package and component name. For example:

```
USA/MyCompany/FinanceServer/FinancePackage/PayrollAdmin
```

This layer of abstraction allows you to move a server to another host without affecting deployed client applications.

Each instance of EAServer operates its own independent name service. In an EAServer cluster, each server in the cluster knows the connection information for all the servers in the cluster. Therefore, an EAServer client can choose one or more cluster members as name servers.

For detailed instructions, see Chapter 5, “Naming Services,” in the *EAServer System Administration Guide*.

Client APIs for naming

For Java and C++ clients, EAServer provides industry-standard client-side APIs for naming services.

For Java clients, EAServer provides implementations of:

- Java Naming and Directory (JNDI) API—see Chapter 3, “Developing EJB Clients,” in the *EAServer Enterprise JavaBeans User’s Guide*
- CORBA standard CosNaming API—see Chapter 8, “Developing CORBA/C++ Components,” in the *EAServer CORBA Components Guide*

For C++ clients, EAServer provides an implementation of the CORBA standard CosNaming API. See Chapter 9, “Developing CORBA/C++ Clients,” in the *EAServer CORBA Components Guide*.

PowerBuilder clients use the naming service implicitly. Name resolution is performed automatically when you create EAServer component instances using the CreateInstance and Lookup functions of the Connection object. You can browse the naming service using the CosNaming API, but such complexity is not necessary. See Application Techniques at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.pb_10.5.apptech/html/apptech/title.htm.

Data source connections

Data source connections allow EAServer components to share pools of preallocated connections to a remote database server, avoiding the overhead imposed when each instance of a component creates a separate connection. Components that support transactions must use a connection from an EAServer data source to interact with remote databases.

For EJB and Web components, EAServer implements the standard mechanism to alias JNDI names to JDBC data sources. For CORBA and PowerBuilder components, EAServer also supports Open DataBase Connectivity (ODBC) and Client-Library™ data sources.

The J2EE connector architecture (JCA) enables you to write portable Java applications that can access multiple transactional enterprise information systems. A connector is a specialized connection factory that provides connections for EJBs, Java servlets, JSPs, and CORBA-Java components. The JCA 1.5 architecture implements contracts between application servers and resource adapters, including work contracts, life cycle enhancements, transaction inflow, and message dispatch. JCA 1.5 inbound resource adapters allow you to receive messages of any type.

See Chapter 4, “Database Access,” in the *EAServer System Administration Guide*.

Transaction management

The EAServer transaction management feature allows you to define a component's transactional semantics as part of the component interface. EAServer supports the Java Transaction Service (JTS). See Chapter 2, "CORBA Component Life Cycles and Transaction Semantics," in the *EAServer CORBA Components Guide*.

Localization

EAServer supports localization in the following ways:

- **Coded character set conversions** EAServer supports multiple coded character sets for clients and components. When a client and component use different coded character sets, EAServer automatically converts character data from one character set to another. For example, if a client uses the roman8 character set and a component uses iso_1, EAServer converts string parameters and return values automatically from roman8 to iso_1 when the client calls the component methods.

  In accordance with the Java standards, Java components and Java clients use 16-bit Unicode. Unicode contains mappings for all characters in all other known coded character sets.

  For C++ components, you can specify code sets using the Management Console. The server Codeset property specifies the default for all C++ components. You can override the default for an individual component by setting the Codeset property at the component level.

  For C++ clients, you can specify a character set when initializing the EAServer client ORB by setting the ORBCodeSet property. See Chapter 9, "Developing CORBA/C++ Clients," in the *EAServer CORBA Components Guide*.

- **Localizing Web applications** EAServer supports the HTTP 1.1 internationalization features defined in the Java Servlet 2.4 specification. Using these features, you can develop servlets that respond in the language specified by the request header, or configure localized versions of Web site's static pages.

  See Chapter 1, "Defining Web Applications," in the *EAServer Web Application Programmer's Guide*.

- **Coded character set interoperability with PowerBuilder** The Sybase document Guidelines for Code Set Interoperability with PowerBuilder and EAServer at http://content.sybase.com/detail?id=1028793 includes information about the following topics:
  - Transmission code set
  - Client code set
  - Component code set
  - PowerBuilder code set
  - The euro character

Result-set support

EAServer methods can return tabular data to the calling client. This feature can be useful for:

- **Use with data-aware controls** Some front-end tools provide objects that can automatically display a result set. For example, if you are using PowerBuilder, you can return results in a DataStore object from component methods and display the results using a DataWindow® control in the client. PowerBuilder DataWindow technology, available in both Web and PowerBuilder clients, allows you to display result sets and synchronize updates with a minimum of coding.
- **Efficiency** For tasks that require returning tabular data, using an EAServer result set is the most efficient alternative. Common uses of result sets include menu and picklist population. For example, in an online clothing catalog, you must list in-stock sizes for each item.

  The EAServer result set allows data to be sent all at once (rather than requiring a get-next-row method and one client-server round trip per method). A large EAServer result set can be sent with less overhead than is required to encapsulate tabular data as an object and send a serialized version of the object to the client.

Each component model provides an interface that allows you to define result sets from scratch or to forward results from a remote database query directly to the client. See the *EAServer CORBA Components Guide*.

For information on using the PowerBuilder DataWindow, see the PowerBuilder *Application Techniques* manual.

Support for asynchronous messaging

The *EAServer Java Message Service User's Guide* describes EAServer messaging support. The EAServer message service supports the Java Message Service (JMS) 1.1 specification, which addresses the demands of distributed systems in a coherent manner. JMS offers an API and a set of semantics that prescribe the interface and general behavior of a messaging service.

The message service allows you to publish or send messages to a queue, where they are stored until they can be delivered to the queue's recipient, which is either a client or a component. The message service provides a pull-style mechanism for client notification and a push-style mechanism for component notification. Clients can check their queue for new messages, or spawn a thread to wait for a message's arrival. Components can also check for new messages and they can register to be notified when messages arrive in their queue.

The message service provides transient and persistent message storage for message consumers, and allows message producers to send messages to specific message queues, or to publish messages, which can be sorted by topic, and are available to all message queues.

Asynchronous processing support

Most server processing is driven by client interaction. However, some tasks are best performed asynchronously; for example, maintaining copies of cached data retrieved from a slower source, updating search indexes on a regular schedule, or performing lengthy calculations. EAServer provides two vehicles to support asynchronous processing:

- Use service components for processing that must run continuously for the life of the server process, or must be performed once each time the server is started. Service components run in a thread that is started when the server starts. See Chapter 4, "Creating Service Components," in *EAServer Automated Configuration Tools*.
- The EAServer thread manager allows you to run threads at any time. Threads started by the thread manager execute independently of the client or component that starts them, and can be configured to run once, or periodically at regular intervals. See Chapter 5, "Using the Thread Manager," in *EAServer Automated Configuration Tools*.

Jetty

EAServer uses Jetty as its Web container. Jetty is an open-source Web container that supports HTTP 1.1, the Java Servlet 2.4 specification, and the JSP 2.0 specification.

JNI compiler

The Java Native Interface (JNI) compiler is a command line tool that can generate:

- C++ proxies for accessing Java classes and interfaces, which are more productive and less error prone than direct JNI calls.
- C++ classes for implementing Java native methods, which are more efficient than IDL-based component calls from Java to C++ or vice versa.

You may want to use the JNI compiler:

- For developing C++ clients that talk to EJB components or JMS using the EJB/JMS client runtime features (for example, data compression), which are not available in the CORBA C++ client runtime.
- To call legacy C/C++ code from your components, and you would rather write EJBs with a few native methods than write CORBA C++ components.

See Chapter 13, “JNI Compiler,” in the *EAServer System Administration Guide*.

Systems management support

EAServer includes enhanced support for remote systems management. The implementation is based on the Java Management Extensions (JMX) agent management framework. Systems management provides the following enhancements:

- Allows you to create management beans (MBeans) that can be run in a JMX framework.
- Supports both SNMP and JMX management tools.

- Allows you to monitor servers, clusters, and key server subsystems such as the message service, the component dispatcher, and network listeners.

See Chapter 14, “Systems Management,” in the *EAServer System Administration Guide*.

IPV6 support

EAServer supports the Internet Protocol Version 6 (IPV6) on platforms that provide the underlying network support such as Windows 2003, Windows XP, Solaris 2.9, and Solaris 10. Windows 2000 does not support IPV6. IPV6 support requires JDK version 1.4 or later.

See “IPV6 support” in Chapter 3, “Creating and Configuring Servers,” in the *EAServer System Administration Guide*.

CHAPTER 2

Developing an EAServer Application

This chapter explains some of the basic concepts and terminology associated with developing component-based EAServer applications in a three-tier environment.

Introductory concepts

An EAServer application consists of one or more packages and a client application or applet. Packages consist of components, and components are made up of one or more methods.

- EAServer can host, manage, and execute components such as JavaBeans or CORBA-compliant components. In EAServer, a **component** is simply an **application object** that consists of one or more methods. EAServer components typically execute business logic, access data sources, and return results to the client. Clients (applets) create an instance of a component and execute methods associated with that component. Components run only within EAServer.
- A **package** is a collection of components that work together to provide a service or some aspect of your application's business logic. A package defines a **boundary of trust** within which components can easily communicate. Each package acts as a unit of distribution, grouping together application resources for ease of deployment and management.

  EAServer supports the following types of components:

  - EJB

    - Java
    - CORBA C++
    - PowerBuilder nonvisual object (NVO)
- A **stub** is a Java class or a C++ stub generated by EAServer, which acts as a **proxy object** for an EAServer component. A stub is compiled and linked with your Java applets or client application. A stub communicates with EAServer to instantiate and invoke a method on a component in the middle tier. Stubs make a remote EAServer component appear local to the client.
- A **skeleton** acts as the interface between the EAServer runtime environment and the user code that implements the method. Skeletons are compiled and linked with each of the components, and at runtime they enable EAServer to locate and invoke an appropriate method.
- EAServer transparently maintains a **session** between a client application and EAServer. Unlike a typical HTTP scenario, where a new connection is created for each request and response, sessions allow a browser to maintain a connection with the server across a multiple request-response cycle.
- A **Web application** is a collection of static HTML pages, Java servlets, and JavaServer Pages. You can develop Web applications to provide a browser-based user interface as an alternative to standalone clients or Java applet clients.

You can develop and distribute an EAServer application across a network. EAServer implements a **three-tier** or **multitier** distributed computing architecture. In this model, three distinct elements work together to give users access to data:

- Client-side applet or application
- Middle-tier components
- The back-end database

Java applets are downloaded to clients, which instantiate components on the server. Client applications are installed on client machines, from which they also instantiate components on the server.

An applet, standalone application, or Web application manages presentation and interaction with an end user. Middle-tier components, which run in EAServer, handle much of the application processing. Finally, the database stores, manages, and processes data.

If the client is an applet, users find and launch applications from traditional HTML pages. Instead of simply loading a static page, EAServer downloads an executable applet to the individual’s browser. If the client is an already-installed application, the user launches the application from his or her machine. Clients communicate directly with an application component running in the middle tier. Server components access data from one or more databases, apply business logic, and return results to the client applet for display.

When a proxy object is created on the client applet, it instantiates a corresponding component registered with EAServer. On the server side, a component is instantiated in response to a request from the proxy object running in the client environment. A method on a component is executed when it is invoked by a proxy object on the client applet.

Web applications can call EAServer components using the same proxies as used by standalone Java clients and applet clients.

Developing an EAServer application

There are three basic steps involved in creating and deploying an EAServer application that employs a Java applet as a client. For information on other types of EAServer clients, see:

- *EAServer EJB User’s Guide*
- *EAServer CORBA Components Guide*

❖ **Creating and deploying an EAServer application**

1. Use an editor to define packages, components, and methods. EAServer generates server-side skeleton files for components, which provide the interface information of each component method.
2. Write the user interface logic for the client model that you have chosen.

   Develop the server-side components that form the business logic of your servlet. EAServer supports many of the integrated development environment (IDE) tools available today.
3. Deploy the application. You can register components on any EAServer installation. Because EAServer is also a Web server, you can write an HTML page for your applet and install it on EAServer.

EAServer runtime environment

A typical EAServer application has an applet or HTML page associated with it. Once you build and deploy such an application, it runs in the following fashion:

1. EAServer receives an HTTP request and downloads the requested HTML page or applet. Included with the applet are the Java stubs, which through a proxy, instantiate components and invoke the methods on those components.
2. The client establishes a session with EAServer. The session, unlike an HTTP connection, allows the client and EAServer to maintain a connection throughout the transaction.
3. The client creates a component instance through a client-side proxy. The proxy used depends on the type of component being instantiated. EAServer validates the user against the component’s access list. If the user is validated, the dispatcher checks the location and status of the component and creates an instance.
4. The client invokes the component’s business logic by executing its methods.
5. The component can interact with remote databases. If it does:
    - The component obtains a connection to the database using the EAServer connection caching feature.
    - EAServer checks the component’s transaction property. If the component is marked as transactional, EAServer ensures that remote database commands execute as part of a larger transaction.
6. EAServer returns the results from the database to the client.
7. The client indicates that it has completed the operation. EAServer destroys the component instance or returns it to a pool for future client instantiations. The client disconnects from EAServer.

CHAPTER 3 EAServer Components

This chapter discusses the component models that EAServer supports.

Overview

EAServer components contain the methods that execute business logic and access data sources. The EAServer server-side component support and client-side stub or proxy support are independent. Any EAServer client can execute any type of component. A component of any model can execute components of another model using intercomponent calls without the use of additional gateway software.

Server-side component support

EAServer provides support for several major component models, including:

- **Enterprise JavaBeans** EAServer supports Java components that follow the Java Software EJB specification, versions 1.1, 2.0, and 2.1. An Enterprise JavaBean is a nonvisual, transactional component that is implemented in Java. For more information, see Chapter 1, “Enterprise JavaBeans Overview,” in the *EAServer EJB User’s Guide*.

  A message-driven bean (MDB) is a type of EJB specifically designed as a Java Message Service consumer. Chapter 2, “Setting up the Message Service,” in the *EAServer Java Message Service User’s Guide* describes how to create MDBs.

- **CORBA-Java** CORBA-Java components follow the CORBA component model and use standard CORBA interfaces for transaction management. Almost any Java class with nonvisual behavior can be adapted to run as an EAServer component. The *EAServer CORBA Components Guide* describes the EAServer Java component support in detail.
- **PowerBuilder NVO components** Using PowerBuilder, you can create nonvisual objects (NVOs) that run natively in EAServer as EAServer components. You can also create NVO proxies for EAServer components, then use the proxies in PowerBuilder client applications. See Application Techniques at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.pb_10.5.apptech/html/apptech/title.htm.
- **CORBA-C++ components** EAServer C++ components are C++ classes that contain methods with prototypes similar to the EAServer component interface; the exact interface mapping complies with the CORBA specification for IDL/C++ language bindings.

  Chapter 8, “Developing CORBA/C++ Components,” in the *EAServer CORBA Components Guide* describes C++ component support in detail.

C++ classes can be run as C++ components If you adapted C++ classes to run as EAServer C components in version 1.1, you can run these classes directly as C++ components in version 2.0 and later.

Client stub and proxy support

Applications invoke an EAServer component using a stub or proxy object. The stub or proxy acts as a local surrogate for the remote component; it provides the same method signatures as the component and hides the details of remote server communication. Stubs and proxies are available for:

- **Java (CORBA and EJB)** Any component can be invoked via a Java stub class. EAServer generates source code for Java stubs. At runtime, your client program instantiates the stub. When you call methods on the stub class, the stub transparently invokes the component method on EAServer. You can also create Java servlets or JavaServer pages that run in EAServer and call components. Using this model, you can create “zero-install” applications. With EAServer built-in HTTP support, these applications have no client-machine installation requirements other than the presence of a Web browser. You can add additional interactive functionality to browser-based clients using Java applets that run in the browser.

  EAServer supports these Java client models:

    - **EJB** Your program uses the EJB (javax.ejb) classes and EAServer EJB stubs to call EAServer component methods. This client model follows the EJB 2.1 specification and is backwardly compatible with the EJB 2.0, 1.1, and 1.0 specifications. Chapter 3, "Developing EJB Clients," in the *EAServer EJB User's Guide* describes how to implement EJB clients.
    - **CORBA-Java** Your program uses the EAServer CORBA-compliant Java ORB or any other CORBA-compliant Java ORB to instantiate stubs. Stub method signatures are mapped from the component's interface definition, based on the CORBA specification of IDL-Java language bindings. Chapter 13, "Developing CORBA/Java Clients," in the *EAServer CORBA Components Guide* describes how to implement CORBA-Java clients.
- **PowerBuilder** PowerBuilder allows you to generate NVOs that act as proxies for EAServer components. Using a proxy, you can call component methods as if they were implemented as local NVO methods. See *Application Techniques* in the PowerBuilder documentation.
- **C++ (CORBA)** Your program uses the EAServer CORBA-compliant C++ ORB or any other CORBA-compliant C++ ORB to instantiate stubs. Stub method signatures are mapped from the component's interface definition, based on the CORBA specification of IDL-C++ language bindings. Chapter 9, "Developing CORBA/C++ Clients," in the *EAServer CORBA Components Guide* describes how to implement C++ clients.

CORBA

CORBA is a distributed component architecture defined by the Object Management Group (OMG). EAServer supports the CORBA IIOP.

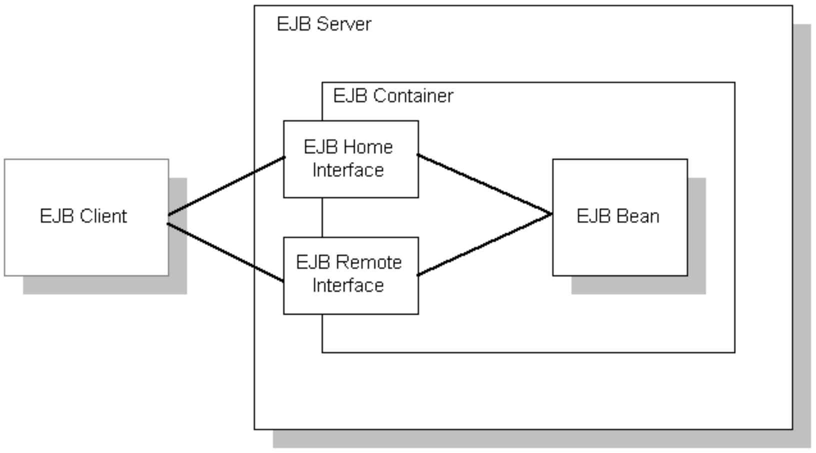
For information on the CORBA architecture, see the specifications available at the OMG Web site at http://www.omg.org.

Enterprise JavaBeans components

The Enterprise JavaBeans (EJB) technology defines a model for the development and deployment of reusable Java server components, called **EJB components**.

An EJB component is a nonvisual server component with methods that typically provide business logic in distributed applications. A remote client, called an EJB client, can invoke these methods, which typically results in database updates. Since EAServer uses CORBA as the basis for the EJB component support, EJB components running in EAServer can be called by any other type of EAServer client or component, and even CORBA clients using ORBs from other vendors that are compatible with CORBA 2.3.

The EJB architecture looks like this:



EJB server The EJB server contains the EJB container, which provides the services required by the EJB component. EAServer is an EJB server.

EJB client An EJB client usually provides the user-interface logic on a client machine. The EJB client makes calls to remote EJB components on a server and needs to know how to find the EJB server and how to interact with the EJB components. An EJB component can act as an EJB client by calling methods in another EJB component.

An EJB client does not communicate directly with an EJB component. The container provides proxy objects that implement the components home and remote interfaces. The component's **remote interface** defines the business methods that can be called by the client. The client calls the **home interface** methods to create and destroy proxies for the remote interface.

EJB container The EJB specification defines a container as the environment in which one or more EJB components execute. The container provides the infrastructure required to run distributed components, allowing client and component developers to focus on programming business logic, and not system-level code. In EAServer, the container encapsulates:

- The client runtime and generated stub classes, which allow clients to execute components on a remote server as if they were local objects.
- The naming service, which allows clients to instantiate components by name, and components to obtain resources such as database connections by name.
- The EAServer component dispatcher, which executes the component's implementation class and provides services such as transaction management, database connection pooling, and instance lifecycle management.

EJB component implementation The Java class that runs in the server implements the bean's business logic. The class must implement the remote interface methods and additional methods for lifecycle management.

EJB component types

You can implement three types of EJB components, each for a different purpose:

- Stateful session beans
- Stateless session beans
- Entity beans

Stateful session beans

A stateful session bean manages complex processes or tasks that require the accumulation of data, such as adding items to a Web catalog's shopping cart. Stateful session beans have the following characteristics:

- They manage tasks that require more than one method call to complete, but are relatively short-lived. For example, a session bean might manage the process of making an airline reservation.
- They typically store session state information in class instance data, and do not survive server crashes unless they are run in a cluster that has persistent storage enabled for the component.
- There is an affinity between each instance and one client from the time the client creates the instance until it is destroyed by the client or by the server in response to an expired instance timeout limit.

For example, if you create a session bean on a Web server that tracks a user's path through the site, the session bean is destroyed when the user leaves the site or idles beyond a specified time

Stateless session beans

A stateless session bean manages tasks that do not require the keeping of client session data between method calls. Stateless session beans have the following characteristics:

- Method invocations do not depend on data stored by previous method invocations.
- There is no affinity between a component instance and a particular client. Each call to a client's proxy can invoke a different instance.
- From the client's perspective, different instances of the same component are identical.

Unlike stateful session beans, stateless session beans can be pooled by the server, improving overall application performance.

Entity beans

An entity bean models a business concept that is a real-world object. For example, an entity bean might represent a scheduled airplane flight, a seat on the airplane, or a passenger's frequent-flyer account. Entity beans have the following characteristics:

- Each instance represents a row in a persistent database relation, such as a table, view, or the results of a complex query.
- The bean has a primary key that corresponds to the database relation's key, and is represented by a Java datatype or class.

EJB transaction attribute values

Each EJB component has a transaction attribute that determines how instances of the component participate in transactions. In EAServer, you set the transaction attribute in the components Ant configuration file.

When you design an EJB component, you must decide how the bean will manage transaction demarcation: either programmatically in the business methods, or by the container, based on the value of the transaction attribute in the deployment descriptor.

A session bean can use either bean-managed transaction demarcation or container-managed transaction demarcation; you cannot create a session bean where some methods use container-managed demarcation and others use bean-managed demarcation. An entity bean must use container-managed transaction demarcation.

EJB container services

The EJB container provides services to EJB components. The services include transaction and persistence support.

Transaction support An EJB container must support transactions. EJB specifications provide an approach to transaction management called declarative transaction management. In declarative transaction management, you specify the type of transaction support required by your EJB component. When the bean is deployed, the container provides the necessary transaction support.

Persistence support An EJB container can provide support for persistence of EJB components. An EJB component is persistent if it is capable of saving and retrieving its state. A persistent EJB component saves its state to some type of persistent storage (usually a file or a database). With persistence, an EJB component does not have to be re-created with each use.

An EJB component can manage its own persistence (by means of the logic you provide in the bean) or delegate persistence services to the EJB container. Container-managed persistence means that the data appears as member data and the container performs all data retrieval and storage operations for the EJB component. See Chapter 2, “Deploying and Configuring EJB Components,” in the *EAServer EJB User’s Guide*.

Java components

EAServer can load and execute a Java class file as a component. The class can be a standalone class or part of a JavaBeans component that does not display any graphics or text, that is, a nonvisual JavaBeans component.

Defining the component's interfaces

The definition of a Java component specifies the interfaces that the component implements as well as its other properties.

All component interfaces for EAServer components are defined in CORBA IDL modules that are stored in the EAServer IDL repository. Chapter 3, "Using CORBA IDL," in the *EAServer CORBA Components Guide* describes how to define IDL interfaces.

Java component developers typically use one of the following to define the interface or interfaces that their component implements:

- **Use existing interfaces from the EAServer IDL repository** In some cases, client and server component developers may have agreed upon one or more interfaces that a component must implement. In this case, it is up to the component developer to implement any specified interfaces. EAServer stores HTML documentation for all interfaces in the IDL repository in the *html/ir* subdirectory of the EAServer installation.
- **Define one or more new IDL interfaces** If you are defining the interface yourself, you can use a text editor to create a new interface for the component. See Chapter 3, "Using CORBA IDL," in the *EAServer CORBA Components Guide*.

CORBA-C++ components

EAServer provides a CORBA-compatible C++ client-side interface. This allows you to create CORBA EAServer C++ applications. C++ components and clients are also interoperable with clients and components using other technologies. See the *EAServer CORBA Components Guide*.

The dynamic invocation interface is not supported.

Defining components

Use the Management Console to define basic information about a C++ component, and generate files that are required to write the component's class implementation and to compile the class into a dynamic link library (on Windows) or shared library (on UNIX).

Write your component as a C++ class; the generated files include a class implementation template in which you can write your method logic. In addition, EAServer supplies an application programming interface that contains classes and methods that you can use to perform EAServer-specific tasks. You can use the EAServer API to write code to handle errors, cache connections to third-tier database servers, return result sets, manage transactions, share data between instances of the same component, retrieve a client's SSL certificate information, and make intercomponent calls.

PowerBuilder components

Using PowerBuilder, you can create NVOs that run natively in EAServer as EAServer components. You can also create PowerBuilder client applications that access EAServer components. Inside EAServer, PowerBuilder components run in the PowerBuilder Virtual Machine (PBVM), which allows the EAServer component dispatcher to call the methods in your NVO component.

The PowerBuilder IDE includes wizards to create and deploy components, and generate proxies for use in PowerBuilder based client applications. For details, see See Application Techniques at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.pb_10.5.apptech/html/apptech/title.htm.

The PowerBuilder IDE runs on Windows platforms, but you can deploy PowerBuilder components to EAServer on any platform for which a compatible PBVM is available, including most UNIX platforms. For more information, see the EAServer release bulletin for your platform.

CHAPTER 4

Web Applications

A Web application allows you to deploy interrelated Web content, JavaServer Pages (JSPs), and Java servlets as a cohesive unit, and configure the Web server properties required by the servlets and JSPs.

This chapter presents an overview of Web applications. For detailed information on developing Web applications, see the *Web Application Programmer's Guide*.

Overview

A Web application is a unit of deployment for interrelated Web content, JSPs, and Java servlets. The Web application contains static files, servlet and JSP implementation classes, and a deployment descriptor that describes how the files, servlets, and JSPs are configured on the host server. The deployment descriptor also allows you to configure application-specific HTTP properties, such as MIME types and per-file security constraints. To tie it all together, a Web application provides an abstract naming convention for the JNDI names of database connections and EJBs.

A Web application represents a subset of the files available on a Web server. Each Web application has a **context path** that forms a prefix for URLs that access the JSPs, servlets, and static pages; for example, http://myHost/Finance.

Each Web application also has a **deployment directory** in the server's file system where the Web application's files are deployed. In EAServer, the deployment directory for the Web application *Finance* is this subdirectory in your EAServer installation:

```
/deploy/webapps/Finance
```

Contents of a Web application

Web applications contain several types of components.

Servlet files

Servlets are Java classes that create HTML pages with dynamic content and respond to requests from client applications that are implemented as HTML forms. Servlets also allow you to execute business logic from a Web browser or any other client application that connects using the Hypertext Transfer Protocol (HTTP). See Chapter 2, "Creating Java Servlets," in the *EAServer Web Application Programmer's Guide*.

All servlets are part of a Web application.Web clients invoke a servlet by prepending the Web application's context path to an alias that is mapped to the servlet. For example, the following URL invokes a servlet mapped to the alias "Account" in the application with the context path "Finance":

```
http://myhost/Finance/Account?type=add
```

JSP files and tag libraries

JSPs allow you to embed snippets of Java code into HTML pages to create dynamic content. JSP tag libraries allow you to extend the standard HTML markup tags with custom tags backed by Java classes. See Chapter 4, "Creating JavaServer Pages," in the *EAServer Web Application Programmer's Guide* for more information on creating JSPs. If you have the PowerBuilder IDE, you can deploy JSP-based Web applications from PowerBuilder to EAServer. For more information, see Working with Web and JSP Targets at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.pb_10.5.wbtarget/html/wbtarget/title.htm.

Static files

You can include files that provide static content for the site in the Web site, including HTML, images, sounds, and so forth. You can also include Java applet files. You can configure the application's deployment descriptor to specify security constraints for static files and any unique MIME types required by your content.

You must deploy static files to the following subdirectory in your EAServer installation directory:

```
deploy/webapps/web-app
```

Where *web-app* is the name of the Web application. You can include subdirectories, which are reflected in your application's URL namespace.

If you import a Web archive (WAR) file, the importer expands the application's static files to this location.

Java classes

A Web application's Java classes include the implementation class for each servlet and JSP, and any server-side utility classes used by the servlets and JSPs.

EAServer uses a custom class loader to run a Web application's servlets and classes referenced by servlet and JSP code. This feature allows hot refresh of servlets and JSPs. The custom class loader also allows each Web application to run with its own effective Java class path.

EAServer also supports class sharing among components and servlets. You can configure custom class lists for components, Web applications, J2EE applications, or a server process. The custom list allows you to support hot refresh of the implementation, and limit the number of copies of shared classes that are loaded into server memory. For example, if a Web application calls an EJB component, you can configure the component and the Web application to share instances of the component stub classes and common utility classes. See Chapter 10, "Configuring Java Class Loaders," in the *EAServer System Administration Guide*.

Deployment descriptor

The application's deployment descriptor catalogs the servlets, JSPs, and files contained in the application, as well as the properties of each. The descriptor must be formatted in XML, using the DTD specified in the *Java Servlet Specification Version 2.4*. You can create a descriptor using the Management Console or another J2EE-compliant development tool.

Index