

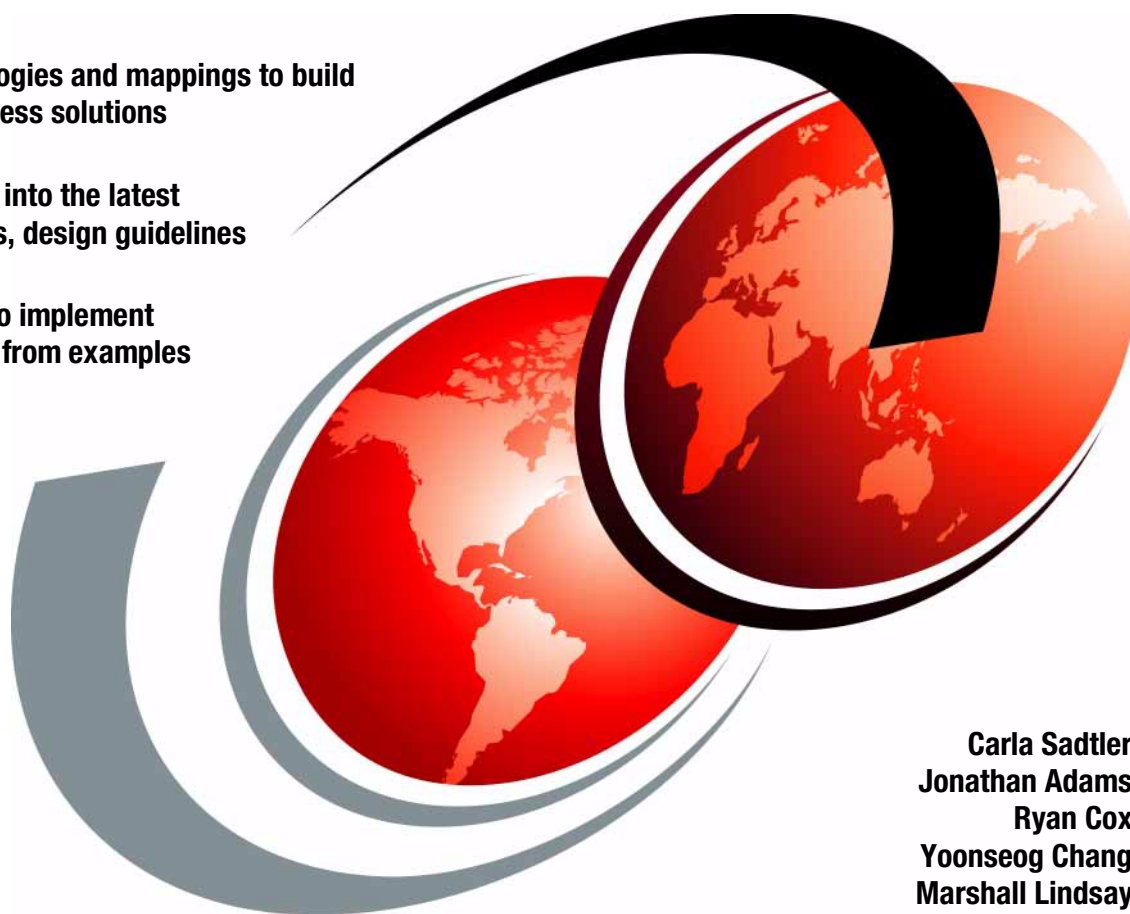
# User-to-Business Patterns Using WebSphere Enterprise Edition

## Patterns for e-business Series

Select topologies and mappings to build  
U2B e-business solutions

Gain insight into the latest  
technologies, design guidelines

Learn how to implement  
the solution from examples



Carla Sadtler  
Jonathan Adams  
Ryan Cox  
Yoonseog Chang  
Marshall Lindsay

[ibm.com/redbooks](http://ibm.com/redbooks)

# Redbooks





International Technical Support Organization

**User-to-Business Patterns  
Using WebSphere Enterprise Edition  
Patterns for e-business Series**

September 2000

**Take Note!**

Before using this information and the product it supports, be sure to read the general information in Appendix B, "Special notices" on page 329.

**First Edition (September 2000)**

This edition applies to IBM WebSphere Application Server 3.0, Enterprise Edition for Windows NT, program number 5639-I09.

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. HZ8 Building 678  
P.O. Box 12195  
Research Triangle Park, NC 27709-2195

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

**© Copyright International Business Machines Corporation 2000. All rights reserved.**

Note to U.S. Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

## Contents

|  |    |
|--|----|
| <b>Preface</b> .....   | ix |
| The team that wrote this redbook .....                                 | ix |
| Comments welcome .....   | xi |
| <br><b>Chapter 1. Introduction to business patterns</b> .....          | 1  |
| 1.1 The Application Framework for e-business .....                     | 1  |
| 1.2 Patterns for e-business .....                                      | 2  |
| 1.2.1 Components of the Patterns for e-business .....                  | 3  |
| 1.2.2 Defined Patterns for e-business .....                            | 3  |
| 1.2.3 How to use these patterns .....                                  | 5  |
| 1.2.4 Patterns for e-business Web site .....                           | 6  |
| 1.3 The User-to-Business pattern .....                                 | 6  |
| 1.4 Component Broker .....   | 7  |
| <br><b>Part 1. User-to-Business patterns: topologies 5 and 6</b> ..... | 13 |
| <br><b>Chapter 2. Choosing the application topology</b> .....          | 15 |
| 2.1 Application topologies .....                                       | 15 |
| 2.1.1 Web-up .....   | 15 |
| 2.1.2 Enterprise-out .....   | 16 |
| 2.2 Application topology 5 .....                                       | 18 |
| 2.2.1 Application topology 5: business driver .....                    | 18 |
| 2.2.2 Application topology 5: key features .....                       | 18 |
| 2.2.3 Application topology 5: considerations .....                     | 20 |
| 2.3 Application topology 6 .....                                       | 20 |
| 2.3.1 Application topology 6: business driver .....                    | 20 |
| 2.3.2 Application topology 6: key features .....                       | 20 |
| 2.3.3 Application topology 6: considerations .....                     | 22 |
| <br><b>Chapter 3. Choosing the runtime topology</b> .....              | 23 |
| 3.1 An introduction to the node types .....                            | 23 |
| 3.2 Runtime topology A .....   | 26 |
| 3.2.1 Basic topology .....   | 26 |
| 3.3 Runtime topology B .....   | 28 |
| 3.3.1 Basic topology .....   | 28 |
| 3.4 Variation to runtime topology A and B .....                        | 29 |
| 3.4.1 Variation 1 (emerging) .....                                     | 29 |
| <br><b>Chapter 4. Product mapping</b> .....                            | 33 |
| 4.1 Product mappings for the basic topology (emerging) .....           | 33 |
| 4.2 Runtime topology variation 1 (emerging) .....                      | 35 |

|   |           |
|---|-----------|
| 4.3 Security . . . . .  | 37        |
| 4.3.1 Component Broker security . . . . .                         | 37        |
| 4.3.2 WebSphere Advanced security . . . . .                       | 39        |
| 4.4 Implementing a redirector . . . . .                           | 40        |
| 4.5 Workload management (WLM) . . . . .                           | 43        |
| 4.5.1 Component Broker . . . . .                                  | 43        |
| 4.5.2 WebSphere Advanced Edition . . . . .                        | 45        |
| <b>Part 2. User-to-Business patterns: guidelines . . . . .</b>    | <b>47</b> |
| <b>Chapter 5. Technology options . . . . .</b>                    | <b>49</b> |
| 5.1 Web client . . . . .  | 50        |
| 5.1.1 Web browser . . . . .                                       | 51        |
| 5.1.2 HTML . . . . .  | 52        |
| 5.1.3 Dynamic HTML (DHTML) . . . . .                              | 53        |
| 5.1.4 XML (client-side) . . . . .                                 | 53        |
| 5.1.5 JavaScript . . . . .  | 54        |
| 5.1.6 Java applets . . . . .                                      | 54        |
| 5.2 Web application server . . . . .                              | 56        |
| 5.2.1 Java servlets . . . . .                                     | 57        |
| 5.2.2 JavaServer Pages (JSP) . . . . .                            | 58        |
| 5.2.3 JavaBeans . . . . .   | 58        |
| 5.2.4 XML . . . . .   | 59        |
| 5.2.5 JDBC . . . . .  | 59        |
| 5.3 Integration Server . . . . .                                  | 60        |
| 5.3.1 Enterprise JavaBeans . . . . .                              | 66        |
| 5.3.2 Connectors . . . . .  | 68        |
| 5.4 Additional enterprise Java APIs . . . . .                     | 70        |
| 5.5 References and where to find more information . . . . .       | 71        |
| <b>Chapter 6. Application design guidelines . . . . .</b>         | <b>73</b> |
| 6.1 Application elements . . . . .                                | 76        |
| 6.2 Understanding supporting technologies . . . . .               | 79        |
| 6.2.1 Java servlets . . . . .                                     | 80        |
| 6.2.2 JavaServer Pages (JSP) files . . . . .                      | 83        |
| 6.2.3 JavaBeans . . . . .   | 86        |
| 6.2.4 Enterprise JavaBeans . . . . .                              | 87        |
| 6.2.5 What's next? . . . . .                                      | 87        |
| 6.3 Application Structure . . . . .                               | 88        |
| 6.3.1 Model-View-Controller (MVC) design pattern . . . . .        | 89        |
| 6.4 Application component contracts . . . . .                     | 93        |
| 6.4.1 Result beans and View beans design pattern . . . . .        | 94        |
| 6.4.2 Advantages and disadvantages of Result beans and View beans | 96        |

|                   |  |            |
|-------------------|--|------------|
| 6.5               | Application output formatting . . . . .                                  | 98         |
| 6.5.1             | Formatter beans . . . . .  | 98         |
| 6.5.2             | Advantages and disadvantages of Formatter beans . . . . .                | 98         |
| 6.6               | Application business logic granularity . . . . .                         | 99         |
| 6.6.1             | Command beans . . . . .  | 100        |
| 6.6.2             | Advantages and disadvantages of Command beans . . . . .                  | 101        |
| 6.6.3             | An alternative approach . . . . .  | 102        |
| 6.7               | The Consumer Banking application . . . . .                               | 103        |
| 6.7.1             | UML and the Consumer Banking Application . . . . .                       | 104        |
| 6.7.2             | MVC and the Consumer Banking application . . . . .                       | 114        |
| 6.7.3             | Consumer Banking implementation . . . . .                                | 120        |
| 6.8               | Implementing topology 5: the Mortgage Payment System . . . . .           | 134        |
| 6.8.1             | UML and the Mortgage Payment System . . . . .                            | 135        |
| 6.8.2             | MVC and the Mortgage Payment System . . . . .                            | 136        |
| 6.8.3             | Implementing the Mortgage Payment System . . . . .                       | 139        |
| 6.9               | Implementing topology 6: the Consumer Banking Plus application . . . . . | 141        |
| 6.9.1             | MVC and the Consumer Banking Plus application . . . . .                  | 143        |
| 6.9.2             | Implementing the Consumer Banking Plus application . . . . .             | 146        |
| 6.10              | Other design considerations . . . . .                                    | 150        |
| <hr/>             |  |            |
| <b>Part 3.</b>    | <b>Working example . . . . .</b>   | <b>151</b> |
| <b>Chapter 7.</b> | <b>Component Broker introduction . . . . .</b>                           | <b>153</b> |
| 7.1               | What is CBConnector? . . . . .   | 153        |
| 7.1.1             | Development . . . . .  | 154        |
| 7.1.2             | System management . . . . .  | 154        |
| 7.2               | CBConnector's focus areas . . . . .                                      | 155        |
| 7.3               | Component Broker – a member of the Transaction Series family . . . . .   | 155        |
| <b>Chapter 8.</b> | <b>U2B topology 5 and 6 implementation . . . . .</b>                     | <b>157</b> |
| 8.1               | Family Sample . . . . .  | 157        |
| 8.1.1             | Modified version . . . . .   | 157        |
| 8.2               | Basic topology . . . . .   | 158        |
| 8.3               | Second topology . . . . .  | 161        |
| 8.4               | User IDs used in the installation . . . . .                              | 163        |
| <b>Chapter 9.</b> | <b>Setting up Component Broker . . . . .</b>                             | <b>165</b> |
| 9.1               | Installing and configuring DCE . . . . .                                 | 165        |
| 9.1.1             | Installing the DCE server or client . . . . .                            | 165        |
| 9.1.2             | DCE server configuration . . . . .                                       | 166        |
| 9.1.3             | DCE client configuration . . . . .                                       | 170        |
| 9.2               | Component Broker installation and configuration . . . . .                | 172        |
| 9.2.1             | Configuring CB SM V3.0.2 . . . . .                                       | 173        |

|  |            |
|--|------------|
| 9.3 Installing MQ Application Adaptor in CB . . . . .                    | 178        |
| <b>Chapter 10. Sample application implementation . . . . .</b>           | <b>179</b> |
| 10.1 Install the DB2 banking database . . . . .                          | 179        |
| 10.2 Installing Family Samples on WebSphere Advanced Edition server      | 180        |
| 10.2.1 Initializing the application's properties file . . . . .          | 180        |
| 10.2.2 Defining the application to the WebSphere Application Server      | 181        |
| 10.3 Installing Family Samples on WebSphere Enterprise Edition (CB) .    | 185        |
| 10.3.1 Step 1 - generating the code . . . . .                            | 186        |
| 10.3.2 Step 2 - importing enterprise beans into Object Builder . . . . . | 188        |
| 10.3.3 Step 3 - compiling the generated code . . . . .                   | 198        |
| 10.3.4 Step 4 - loading and activating the applications . . . . .        | 199        |
| 10.3.5 Step 5 - preparing for DB2 access . . . . .                       | 200        |
| 10.3.6 Step 6 - update the JNDI naming service . . . . .                 | 203        |
| 10.3.7 Step 7 - running the application server . . . . .                 | 204        |
| 10.4 Creating and Installing the Mortgage Payment System application .   | 205        |
| 10.4.1 Step 1 - generating the code . . . . .                            | 205        |
| 10.4.2 Step 2 - importing the enterprise beans into Object Builder . .   | 208        |
| 10.4.3 Step 3 - compiling the generated code . . . . .                   | 214        |
| 10.4.4 Step 4 - Loading and activating the applications . . . . .        | 214        |
| 10.4.5 Create the MQSeries queue manager and queues . . . . .            | 216        |
| 10.4.6 Creating an Access bean for the MQ-EJB component . . . . .        | 217        |
| 10.4.7 MPSPostingService "legacy" application . . . . .                  | 218        |
| 10.4.8 Testing the MQ-EJB component . . . . .                            | 219        |
| 10.5 Running Consumer Banking and Consumer Banking Plus . . . . .        | 219        |
| 10.6 Topology 5 implementation . . . . .                                 | 222        |
| 10.6.1 Installing the application in WebSphere Advanced . . . . .        | 223        |
| 10.6.2 Running the topology 5 application . . . . .                      | 224        |
| 10.7 Topology 6 Implementation . . . . .                                 | 224        |
| 10.7.1 Update the ShowAccounts_CMD Command bean . . . . .                | 225        |
| 10.7.2 Update the ShowAccountsResults JSP . . . . .                      | 226        |
| 10.7.3 Update the ShowTransferAccountsResults JSP . . . . .              | 226        |
| 10.7.4 Update the BankTasks Session bean . . . . .                       | 226        |
| <b>Chapter 11. Redirecting using OSE Remote . . . . .</b>                | <b>227</b> |
| 11.1 Overview of the configuration . . . . .                             | 227        |
| 11.2 Configuring the Web application server . . . . .                    | 228        |
| 11.2.1 Adding host alias entries . . . . .                               | 228        |
| 11.2.2 Configuring the servlet engine transport type . . . . .           | 229        |
| 11.3 Configure the plug-in for OSE Remote . . . . .                      | 230        |
| 11.3.1 Configure the plug-in manually . . . . .                          | 231        |
| 11.3.2 Configure the plug-in using a script . . . . .                    | 234        |



|   |     |
|---|-----|
| <b>Chapter 12. Application security</b> . . . . .                 | 237 |
| 12.1 Securing the WebSphere Advanced Web application . . . . .    | 237 |
| 12.2 Enabling application security in WebSphere . . . . .         | 237 |
| 12.3 Enabling WebSphere global security . . . . .                 | 238 |
| 12.3.1 Protecting the application . . . . .                       | 243 |
| 12.4 Hints and tips . . . . .                                     | 248 |
| <b>Chapter 13. Network security</b> . . . . .                     | 249 |
| 13.1 Determining the firewall ports to open . . . . .             | 249 |
| 13.2 Designating the network interfaces . . . . .                 | 249 |
| 13.2.1 Domain firewall . . . . .                                  | 250 |
| 13.2.2 Protocol firewall . . . . .                                | 250 |
| 13.3 Setting up the general security policy . . . . .             | 251 |
| 13.4 Creating the network objects . . . . .                       | 252 |
| 13.4.1 Domain firewall . . . . .                                  | 252 |
| 13.4.2 Protocol firewall . . . . .                                | 252 |
| 13.5 Configuring the Domain Name Servers . . . . .                | 253 |
| 13.6 Creating the firewall rules and services . . . . .           | 253 |
| 13.6.1 Base topology . . . . .                                    | 254 |
| 13.6.2 Variation 1 . . . . .                                      | 261 |
| 13.6.3 Definitions for OSE Remote to WebSphere Advanced . . . . . | 263 |
| <b>Appendix A. Sample source</b> . . . . .                        | 271 |
| A.1 MPSOutbound DO include() . . . . .                            | 271 |
| A.2 MPSService package (topology 5 implementation) . . . . .      | 274 |
| A.2.1 PaymentServlet.java . . . . .                               | 274 |
| A.2.2 PaymentServlet.servlet . . . . .                            | 278 |
| A.2.3 Util.Java . . . . .   | 279 |
| A.2.4 CMD.java . . . . .  | 280 |
| A.2.5 Payment_CMD.java . . . . .                                  | 281 |
| A.2.6 MPSTransactionException.java . . . . .                      | 286 |
| A.3 Mortgage Payment System JSPs . . . . .                        | 287 |
| A.3.1 MPSPayment.jsp . . . . .                                    | 287 |
| A.3.2 MPSPaymentSubmitted.jsp . . . . .                           | 288 |
| A.3.3 MPSPaymentError.jsp . . . . .                               | 289 |
| A.4 Topology 6 implementation . . . . .                           | 290 |
| A.4.1 Commands.EJB.ShowAccounts_CMD.java . . . . .                | 290 |
| A.4.2 Commands.ShowAccounts_CMD.java . . . . .                    | 294 |
| A.4.3 ShowAccountsResults.jsp . . . . .                           | 295 |
| A.4.4 ShowTransferAccountsResults.jsp . . . . .                   | 298 |
| A.4.5 com.ibm.ibmwebs.sample.BankTasksBean . . . . .              | 304 |
| A.5 MPSOutboundTest.java . . . . .                                | 314 |
| A.6 Loading the MQ application in CB . . . . .                    | 320 |

|   |            |
|---|------------|
| A.7 MPSPostingService application . . . . .       | 323        |
| <b>Appendix B. Special notices . . . . .</b>      | <b>329</b> |
| <b>Appendix C. Related publications . . . . .</b> | <b>333</b> |
| C.1 IBM Redbooks . . . . .                        | 333        |
| C.2 IBM Redbooks collections . . . . .            | 333        |
| C.3 Other resources . . . . .                     | 334        |
| C.4 Referenced Web sites . . . . .                | 335        |
| <b>How to get IBM Redbooks . . . . .</b>          | <b>337</b> |
| IBM Redbooks fax order form . . . . .             | 338        |
| <b>Index . . . . .</b>                            | <b>339</b> |
| <b>IBM Redbooks review . . . . .</b>              | <b>343</b> |

---

## Preface

Patterns for e-business are a group of proven, reusable assets that can help speed the process of developing applications. The pattern discussed in this book, the User-to-Business pattern, is the general case of users interacting with enterprise transactions and data. In particular it is relevant to those enterprises that deal with goods and services that cannot be listed and sold from a catalog.

This redbook discusses two application topologies of the User-to-Business patterns. Application topology 5 links multiple presentation tiers to any back-end client, but the back-end is not hidden to the user. Topology 6 extends topology 5 to describe the situation where multiple presentation tiers are linked to multiple back-end clients. It makes third-tier applications seamless by integrating the business logic at the intermediate tier.

The topologies are illustrated using WebSphere Application Server Advanced Edition V3.021 and Component Broker 3.02. The example used is a modified version of the IBM Family Sample.

Part 1 of this redbook takes you through the process of choosing an application topology and a runtime topology. It then gives you possible product mappings for implementation of the chosen runtime topology.

Part 2 provides a set of guidelines for building your e-business application. It includes information on technology options, application design and application development.

Part 3 takes you through a working example, showing the implementation of an e-business application using application topology 5 and application topology 6.

---

### The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

The overall manager for Patterns:

**Jonathan Adams** is an IT consultant with IBM's Software Group and the leader of the Patterns for e-business initiative. He works closely with all areas of IBM and industry consultants. His commitment to the idea of a systematic approach

to end-to-end e-business architecture is based on his many years of work in the field with major IBM customers in the United Kingdom.

**Carla Sadtler** is a Senior Software Engineer at the International Technical Support Organization, Raleigh Center. She writes extensively in many areas including WebSphere, SecureWay Communications Servers, network integration, and Web-to-host integration products. Before joining the ITSO 14 years ago, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.

**Ryan Cox** is an Object Technology/Application Development specialist working in IBM Advanced Technical Sales Support. His areas of expertise include design and development of distributed applications using standard Web technologies, Java, Enterprise JavaBeans, and CORBA. He has supported and led Proof-of-Concepts for WebSphere Advanced and Enterprise Edition for the past three years. He holds a Bachelor of Science degree in Computer Science from the University of North Texas.

**Marshall Lindsay** is an Advisory Software Engineer with the Customer Solutions Center in San Diego CA. He has many years of planning, architecture and development experience with distributed applications and has spent two years working specifically in the WebSphere Advanced and Enterprise environments.

**Yoonseog Chang** is a senior IT specialist in IBM Korea. His areas of experience include application design and application development using Smalltalk and Java, and he was an IBM San Francisco technical consultant. He was also engaged in several customer projects using IBM e-business frameworks. He holds a degree in Computer Science from Korea University.

Thanks to the following people for their invaluable contributions to this project:

Margaret Ticknor  
International Technical Support Organization, Raleigh Center

Kevin Cattell  
IBM Raleigh

Mark Comer  
IBM Raleigh

Dave Johnson  
IBM Dallas

Eric Lin  
IBM Santa Teresa

Andras Szakal  
IBM Bethesda

Tom Mitchell  
IBM Cranford

---

## Comments welcome

### **Your comments are important to us!**

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 343 to the fax number shown on the form.
- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)



---

## Chapter 1. Introduction to business patterns

We are all familiar with the pace of development of the computer industry during its relatively brief history. The rapid advances in computer hardware have been driven in no small part by the use of standards and well-specified components for assembly. The desire to apply these same approaches to software construction gave rise to object-oriented software, design patterns, and component-based development.

The idea of design patterns has gained acceptance by software designers and developers because it enables an efficiency in both the communication and implementation of software design, based upon a common vocabulary and reference. Information technology architects, encouraged by the success of design patterns, and facing challenges in systematic and repeatable description of systems, have also explored the idea of architectural patterns.

The Enterprise Solution Structure (ESS) work (see “Enterprise Solutions Structure” in *IBM Systems Journal*, Volume 38, No. 1, 1999 at <http://www.research.ibm.com/journal/sj38-1.html>) looked at patterns for complete end-to-end system architectures. ESS is now part of the IBM Global Services methodology.

The following publications are interesting reading for more information on design patterns and their background:

- *Design Patterns - Elements of Reusable Object-Oriented Software*, by E. Gamma, R. Helm, R. Johnson, J. Vlissides)
- *A Pattern Language*, by C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel).
- *Pattern-Oriented Software Architecture - A System of Patterns* by Buschmann, et al.
- *Pattern Hatching - Design Patterns Applied* by J. Vlissides

---

### 1.1 The Application Framework for e-business

The advent of e-business, with the requirement for interoperability that it brings, has been a major catalyst for the more rapid adoption of standards by the industry.

IBM's Application Framework for e-business establishes:

- A recommended approach for building systems, embodied in the Patterns for e-business.
- Innovative technology delivered in a rich product portfolio.

- Cross-platform standards, including Java and XML.

The Framework, with the standards it proscribes for e-business systems and their components, can be applied to:

- Custom application code
- Application packages
- Software products

The Patterns for e-business are an integral part of the IBM Application Framework for e-business. The patterns make it easy to apply the technologies, standards, and products of the Application Framework to provide an e-business solution.

---

## **1.2 Patterns for e-business**

The Patterns for e-business aim is to communicate in a highly accessible fashion the business pattern, systems architecture (application and runtime topologies), product mappings, and guidelines required for different classes of applications. For some patterns there is also an associated Pattern Development Kit, which provides sample application code to illustrate effective use of those patterns.

The goal is to provide the smallest number of Patterns for e-business that will allow IT architects in 80% of cases to quickly develop 80% of their required infrastructure by the reuse of proven:

- Architecture patterns
- Design patterns
- Runtime patterns
- Application development and systems management patterns
- Design, development, and deployment guidelines
- Code



### 1.2.1 Components of the Patterns for e-business

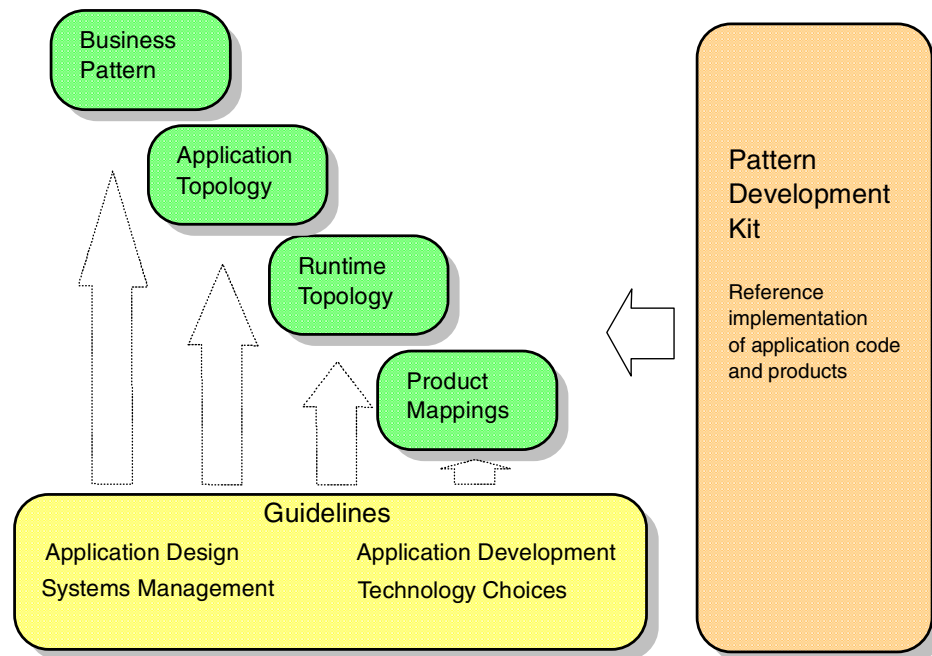


Figure 1. Patterns for e-business

Business patterns describe the interaction between the participants in an e-business solution. The following make up the basic structure of a pattern:

- Application topologies illustrate the various ways to configure the interaction between users, applications, and data. Choosing an application topology will lead to an underpinning runtime topology.
- Runtime topologies use nodes to group functional requirements. The nodes are interconnected to solve a business problem.
- Product mappings show possible combinations of products used to instantiate the runtime topology.
- Guidelines outline and define the processes used to build the e-business application.

### 1.2.2 Defined Patterns for e-business

There are currently six defined Patterns for e-business:

- **User-to-Business** is the general case of users (internal or external) interacting with enterprise transactions and data. In particular it is relevant

to those enterprises that deal with goods and services that cannot be listed and sold from a catalog. It can also be thought of as covering all user to business interactions not covered by the User-to-Online Buying pattern.

- **User-to-Online Buying** is used to describe the special case (a subset of the User-to-Business pattern) where packaged goods, say, are sold through a catalog using a shopping cart, a wallet, etc. This includes both consumers purchasing goods and online buyers purchasing goods from a single supplier. It can also include links to back-end systems to allow for inventory updates and credit checking.
- **Business-to-Business** is used to describe two styles of inter-business to business (Intra-business to business is covered under Application Integration below):
  - The first style (B2Bi) covers programmatic links between arms-length businesses (where potentially a trading partner agreement may be appropriate). A good example of this would be supply chain applications.
  - The second style covers the eMarketPlace where the model supports B2M2B. The M represents the eMarketPlace, which supports multiple buyers and suppliers. The buying function may be performed online or programmatically.
- **User-to-User** is used to describe users collaborating with one another by e-mail, shared documents, etc.
- **User-to-Data** is used to describe users needing to take large volumes of data, text, images, video, etc. and use tools to extract useful information from it.
- **Application Integration** is used to link applications together within a business (such as ERP with existing applications). This can be used within a business pattern or between business patterns.

IBM views e-business as an integration of many application domains into systems that connect a business with its customers, partners, and suppliers. These systems are not confined to Web interfaces, although increasingly many of the user interfaces to the combined system will use Web technology.

The common set of node descriptions in the Patterns for e-business enable communication between architects and designers from very different application domains and will suggest areas for shared nodes and infrastructure.

This is similar to the process of using design patterns to solve a programming design problem, where classes in the composed pattern play multiple roles, derived from the source patterns. It is different, however, in that design pattern composition is based on class diagrams and white box by nature, whereas composing architectural patterns is more component-based.

The Patterns for e-business may be applied to e-business solution areas. Here is a guide to where you may find them most applicable:

*Table 1. Patterns for e-business and ebusiness solutions*

| e-business solution area                    | Business pattern                |
|---|---------------------------------|
| Customer relationship management            | User-to-Business Pattern        |
| e-commerce                                  | User-to-Online Buying Pattern   |
| Supply chain management, e-Marketplace      | Business-to-Business Pattern    |
| Collaboration                               | User-to-User Pattern            |
| Business Intelligence; Knowledge Management | User-to-Data Pattern            |
| Business application integration            | Application Integration Pattern |

### 1.2.3 How to use these patterns

The Patterns for e-business are particularly focused upon addressing common business application problems and providing answers to frequent architecture, design, and implementation questions.

We recommend that you can use the Patterns for e-business in a number of ways according to your needs:

- As a starting point for an end-to-end system architecture.
- As a detailed example and prescriptive approach, following the product mappings and guidance provided.
- As a way to design more complex, multi-channel systems, when several patterns are used together.

As with the design patterns and ESS work, we anticipate that architects and designers will want to combine these patterns to compose solutions to more complex system architectures.

We recommend that you use the Patterns for e-business together with an appropriate development methodology that considers the full set of requirements that are to be understood and implemented, whether these

requirements concern the function of the solution or its operational characteristics such as availability, scalability, or performance.

#### 1.2.4 Patterns for e-business Web site

The Patterns for e-business are published on IBM developerWorks, a portal for developers, and can be located at:

<http://www.ibm.com/software/developer/web/patterns>.

This interactive patterns site acts as a guide to aid you in the selection of the pattern and topologies most relevant to your needs. While you can navigate via shortcuts to the information you most need, the site is structured to enable you to “drill down” into the material as you:

1. Select a business pattern.
2. Select an application topology.
3. Review runtime topologies.
4. Review product mappings.
5. Review guidelines.

At the time of writing, the Web site has material for the User-to-Business and User-to-Online buying patterns, with material for the other business patterns in the process of development.

You can also register at this site for pattern-related updates, which will include information about the Pattern Development Kit for User-to-Business.

---

### 1.3 The User-to-Business pattern

As mentioned earlier, the User-to-Business pattern covers the general case of users interacting with enterprise transactions and data. In particular it is relevant to those enterprises that deal with goods and services that cannot be listed and sold from a catalog. It can also be thought of as covering all user-to-business interactions not covered by the User-to-Online Buying pattern.

The following are some industry examples where the User-to-Business pattern would provide the appropriate application and runtime topologies to fit each particular need.

#### ***Insurance Industry***

- Locate a nearby office
- Locate brokers or agents

- Financial planner and insurance needs analysis tool
- Portfolio summary
- Policy summary and details
- Claims submission and tracking
- Online billing

#### ***Discount Brokerage***

- Portfolio summary
- Detailed holdings
- Buy and sell stocks
- Transaction history
- Quotes and news

#### ***Convenience Banking***

- View account balances
- View recent transactions
- Pay bills and transfer funds
- Stop payments
- Manage bank card

#### ***Telecommunications and Wireless Industry***

- Customers reviewing the account statement
- Customers paying bills online using a credit card
- Customers making changes to their personal profile
- Customers adding, changing, or removing optional services, for example, call waiting or caller ID
- Customers submitting service requests

#### ***Government***

- Taxpayers submitting tax returns
- Drivers renewing automobile licenses
- Downloading forms and applications
- Submitting forms and applications online

#### ***Manufacturing***

- Consumers reviewing required parts and services
- Consumers locating the nearest service center
- Service personnel registering for an upcoming equipment training class
- Consumers submitting and tracking orders

---

## **1.4 Component Broker**

The WebSphere family consists of WebSphere Application Server and other related WebSphere family software that is tightly integrated with the

WebSphere Application Server. The related software provides balancing and caching functions across a pool of Web servers in order to enhance scale and performance. The related software also includes:

- Transcoding support for different devices
- Enablement of rules-oriented applications
- Application personalization
- Linkages to groupware
- Integration with electronic commerce and support for business-to-business interactions
- Content management
- Web site analysis
- Portal support
- A growing set of industry-specific components

There are three editions of WebSphere Application Server available: Standard Edition, Advanced Edition, and Enterprise Edition. The Standard Edition provides a comprehensive platform for designing Java-based Web applications. The Advanced Edition builds on the Standard Edition and adds server capabilities for applications built to the Enterprise JavaBeans Specification from Sun Microsystems. Advanced Edition also adds cloning and workload management capabilities to provide failure bypass and to achieve higher levels of scale. WebSphere Application Server Enterprise Edition builds on the WebSphere Application Server Advanced Edition, adding TXSeries and Component Broker.

There are currently eight logical application topologies associated with the User-to-Business pattern. In Chapter 2, "Choosing the application topology" on page 15, you will be given the information needed to choose the application topology that most suits your needs. This redbook will be dealing specifically with application topologies 5 and 6 using Component Broker, which is a part of the IBM WebSphere Application Server Enterprise Edition.

Component Broker (CB) is IBM's premier distributed component server that provides support for both the Enterprise JavaBeans architecture and Common Object Request Broker Architecture (CORBA) objects, facilitating deep object-oriented integration with legacy systems and a wide variety of clients. It features high levels of systems manageability and scale and is enabled by first-class tools.

A simplified customer profile for Component Broker would have any (not necessarily all) of the following three requirements:

1. Needs a CORBA implementation.

2. Needs an EJB implementation that coordinates changes to heterogeneous systems.
3. Needs the deepest possible integration between objects and tier-3 environments.

This three-item list can be viewed as a simple filter. Customers who are candidates for Component Broker will be deeply committed to objects. These objects could be either CORBA components or Enterprise JavaBeans. For the former, Component Broker is IBM's single solution. For the latter, either Advanced Edition or Component Broker could be used, but if strong transactional coordination is needed, then the focus returns to Component Broker. Component Broker also has a rich infrastructure for hosting objects in a way that supports very deep integration with legacy systems. Component Broker is all about leveraging legacy systems in an object-oriented way. More details are included briefly under each item (below) in order to describe the "next layer of the onion".

1. CORBA implementation.

CORBA provides an alternative to Enterprise JavaBeans (EJBs) for defining distributed object systems.

CORBA is truly open and it reflects more than a decade of input from hundreds of Object Management Group (OMG) members. The EJB specification (and the specification for Java in general) is ultimately under the control of Sun, with input being provided by interested parties in the "Java community". Because CORBA is a true standard rather than a pseudo-standard, CORBA is being embraced and in some cases demanded by industries such as government and telecommunications.

CORBA is language neutral. Business objects may be implemented in any language that a vendor supports. For Component Broker, this includes either Java or C++. With EJBs, only Java can be used as the implementation language. Some customers have strong C++ skills or C++ assets that must be leveraged.

CORBA is strong in terms of interoperability. One implication of this is that clients of the CORBA objects may easily be written in C++ or Java, or they may be ActiveX clients. Clients of EJBs may easily be written in Java, but introducing other client types can be difficult if not impossible (depending on the specifics).

CORBA provides lots of "object services". CORBA currently has an edge on the EJB specification in this area. Object services include the ability to run queries in a way that is essentially independent of the back-end system.

CORBA components are less portable than EJBs. However, CORBA components built for Component Broker are highly portable across the supported platforms (Windows NT, AIX, Solaris, OS/390).

CORBA components are more difficult to build than EJBs. However, Component Broker provides tools that make CORBA component construction an order of magnitude simpler than the alternatives provided by other vendors. EJBs of course are even simpler to build (because of tools like VisualAge for Java).

2. EJB implementation that coordinates changes to heterogenous systems.

For customers who are committed to EJBs (the focus of this redbook), there may be compelling business or functional requirements for keeping back-end systems completely synchronized and consistent. This boils down to high integrity. Everything changes together, or nothing changes at all. This is the classic scenario illustrated by moving money out of the checking account and into the savings account. Component Broker does an excellent job in this area for both EJBs and CORBA objects. The V3.5 capabilities for WebSphere Advanced Edition (WSAE) and Component Broker are as follows:

|      | DB2 (4) | Oracle | Sybase | Informix | CICS | IMS | Encina  | MQSeries |
|------|---------|--------|--------|----------|------|-----|---------|----------|
| WSAE | Yes     | No (1) | Yes    | No       | No   | No  | Yes     | No       |
| WSEE | Yes     | Yes    | No     | Yes      | Yes  | Yes | Yes (2) | Yes(3)   |

**Notes:**

(1) Although WSAE provides a "JTA driver" for Oracle, it provides no recovery. For this reason, "true" two-phase commit support cannot be claimed. However, Oracle (like DB2 and Sybase) is supported for container-managed persistence and as a configuration database.

(2) It should be possible to flow transactions bidirectionally using a CORBA protocol between CB and Encina, and also unidirectionally from CB to Encina using DCE RPC (this is actually accomplished by means of an "EJB wannabe" bridge), but none of these paths have been formally tested.

(3) MQSeries can be coordinated on either the Windows NT or Solaris platforms.

(4) DB2, as listed above, also includes DB2 on S/390. Also, CB supports DB2 5.2 and 6.1, whereas WSAE only supports 6.1.



3. The deepest possible integration between objects and tier-3 environments.

Customers who are really sold on object technology (whether it's based on EJBs or CORBA) get excited about a deep enabling infrastructure that strongly adapts traditional systems and environments to an object-oriented view. So, for example, mapping the security credential associated with an object to the corresponding security signon for a tier-3 system is one means of providing "deep integration". Other examples of a rich "object-friendly" infrastructure include transactional coordination and queryability, connection pooling, caching, configurable concurrency control, and high systems manageability. All of these are considered strong points with respect to Component Broker (for hosting either EJBs or CORBA objects).



---

## **Part 1. User-to-Business patterns: topologies 5 and 6**



## Chapter 2. Choosing the application topology

After identifying the business pattern, in this case, the User-to-Business pattern, the next step in planning an e-business application is to choose the logical application topology that applies to your situation. We will give you a brief overview of the eight defined application topologies for the User-to-Business pattern. The rest of the book will concentrate specifically on application topologies 5 and 6.

### 2.1 Application topologies

The following are typical User-to-Business application topologies. These application topologies do not show middleware, but focus on the shape of the application, the application logic, and associated data.

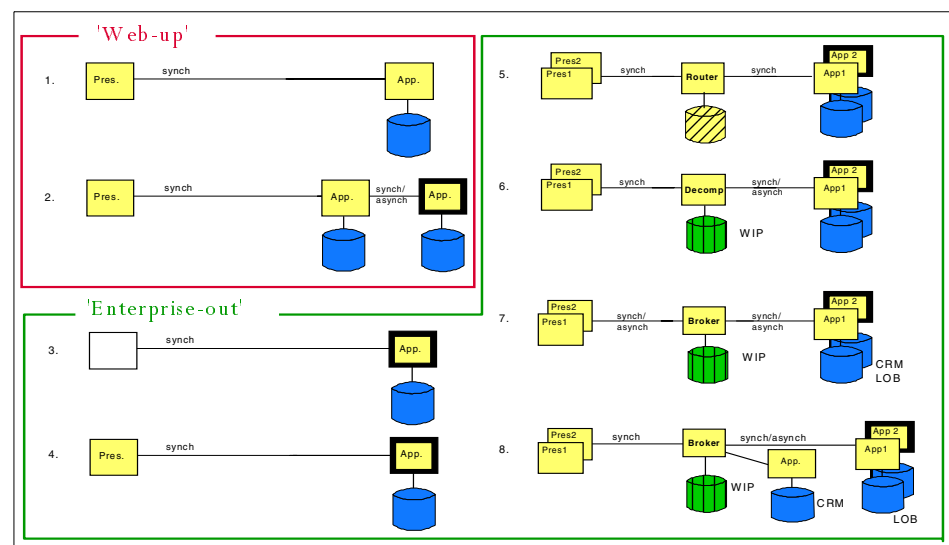


Figure 2. User-to-Business pattern application topologies

#### 2.1.1 Web-up

Web-up is used to represent both an attitude of mind (Web-centric) and the consequent implementation of new Web browser-centric applications focussing only on the Web channel. The data may be acquired by replication from existing sources or typed in by new users from the Web. The two Web-up application topologies (1 and 2) are covered in detail in *Patterns for*

**Topology 1** links a presentation node to business logic residing on the second tier. The second tier can access a local database maintaining the application data.

Topology 1 represents the target topology for most Web-up application server vendors. It aims to fix the scalability problems of client/server and at the same time provide reuse of the business logic and data by all styles of Web browsers. Many vendors promote ease of development by a mixture of scripting and components with little attention to layering the application with separate presentation and business logic layers. This should be avoided. It should also be noted that where the business logic and data are held outside the glasshouse there will be a significant system management cost to manage this asset. The business driver for this is currently to extend the current Web-enabled publishing capability with an e-commerce capability with no back-end integration, a classic Web-up strategy.

**Topology 2** is similar to topology 1, but the business logic on the second tier can access existing applications and data on the third tier.

Topology 2 represents an extension to the Web-up strategy in topology 1. It allows for one or more point-to-point connections to back-end heritage applications or databases so that Web applications can be integrated with existing back-end applications (for example, an e-commerce application integrated with a back-end inventory management applications).

One of the key issues to address is how this topology will be deployed to avoid systems management complexity arising from corporate data on more than one tier.

### **2.1.2 Enterprise-out**

Enterprise-out is used to represent an enterprise-centric attitude of mind and the consequent extension of existing applications out to the new Web channel. Hence support for multiple channels is required.

**Topology 3** represents a very thin client with a 3270/5250/ASCII emulator or Host On-Demand accessing an existing application. The business driver for this is to provide intranet access to existing green-screen applications without having to rewrite/re-engineer these applications. It is a very simplistic enterprise-out scenario.

**Topology 4** represents a thin client (for example, a CICS ECI bean or IBM's Host Publisher) accessing an existing application. The business driver for this is to provide a customized presentation of existing centralized applications without having to rewrite/re-engineer these applications. Care should be taken in the physical implementation so as not to cause an unsupportable number of connections from the Web. Like topology 1 it is a very simplistic enterprise-out scenario.

**Topology 5** links multiple presentation tiers to any back-end client, but the back-end is not hidden from the user, for example a call center or Web browser (multiple presentation) linked to the back-end through an application router.

Topology 5 represents a typical topology used to Web-enable existing robust, highly scalable transactions. The extra scalability requirements generated by enabling thousands of Web users involve security, protocol conversion, session concentration and routing. The business driver for this design is fast, highly scalable, highly available Web-enablement of existing business transactions, that is, a classic Web server enterprise-out strategy. The key feature is the one-to-one-to-one relationship between the tiers at runtime. This design is also valuable in the case of company mergers and takeovers.

**Topology 6** is similar to topology 5, but the mechanics of the back-end applications are hidden. A business request from one of the presentation tiers can be decomposed on the middle tier into multiple back-end transactions, which are then recompiled on the middle tier to return a single business response to the presentation tier.

Topology 6 represents a step beyond topology 5 for the enterprise that wants to make third-tier applications seamless by integrating the business logic at the intermediate tier. This design is increasingly valuable because it can support multiple styles of thin client with a common intermediate tier. The key feature is the one-to-one-to-many relationship between the tiers at runtime. The business driver is to provide customer-oriented support systems rather than product-oriented systems.

**Topology 7** represents mass customization. For example when someone uses a browser to check an insurance claim, the application does cross selling based on personal data. For example, an insurance customer contacts an insurance company to check the status of an insurance claim. While answering his query the system can also retrieve any information about his family, house, car, birthdays, etc. from the corporate data repositories. This data can be held as work-in-progress on the intermediate tier and used to prompt the insurance customer with cross-selling opportunities. The key

feature is the provision of a push mechanism from the second to first tier. The business driver for this design is customizing goods and services to a market of one (mass customization).

**Topology 8** is similar to topology 7 but the customer data is on a different machine/platform from the application server.

In topology 7 the customer relationship management (CRM) data and the line of business (LOB) data are both held on the same third tier. In topology 8 the CRM data is held on the third tier that owns the customer relationship, while the LOB data is accessed from the third tier of various third-party suppliers. The business driver for this design is the need to support virtual enterprises, intermediaries or portals on the Web.

---

## **2.2 Application topology 5**

Topology 5 represents a typical topology used to Web-enable existing robust highly scalable transactions. The extra scalability requirements generated by enabling thousands of Web users involve security, protocol conversion, session concentraton and routing. Topology 5 links multiple presentation tiers to any back-end client, but the back-end is not hidden to the user. For example a call center or Web browser (multiple presentation) linked to the back-end through an application router.

### **2.2.1 Application topology 5: business driver**

The business driver for this design is fast, highly scalable, highly available Web enablement of existing business transactions. There are multiple presentation channels and multiple applications.

Each application stands on its own. There is no need to combine information or features from multiple applications into a single response to the user.

For example, a business could provide access to an order entry application from customer server representatives in a call center. At the same time, it could provide access to inventory data to suppliers using an intranet connection.

### **2.2.2 Application topology 5: key features**

The key feature of topology 5 is the one-to-one-to-one relationship between the tiers at runtime. The application is divided into three logical tiers.



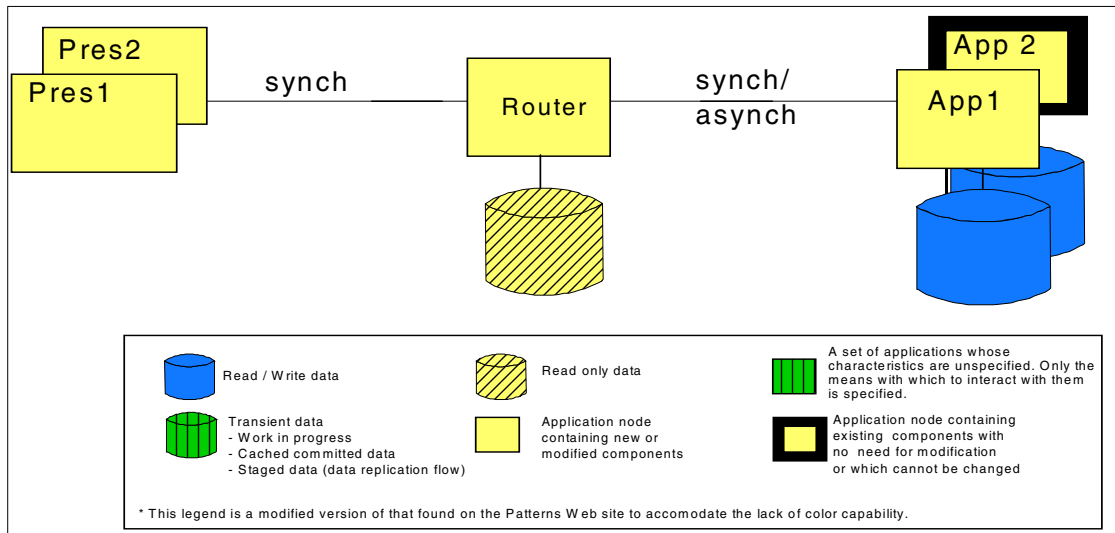


Figure 3. Application topology 5

- The presentation node in the first tier is responsible for the interface into the Web application. It is responsible for all the presentation logic of the application.
- The router node in the second tier links a presentation node to a particular application node. The router provides a common interface to multiple back-end applications, facilitating the connection, but containing no business logic to combine the application data.
- The application node in the third tier is responsible for all the business logic and data access of the application. It may be a new application developed specifically for the Web application or it may be an existing legacy application that may or may not require modification. The business logic and data reside in the third tier.

The communication between the presentation logic and business logic layer is synchronous, meaning that any request coming from the user interface invokes business logic on the application node via the router. After the business logic is executed, control is passed back to the presentation node that uses the results to update the user interface.

The connection between the router node and the application nodes can be a fast asynchronous or synchronous connection. The communication type depends on the communication characteristics and capabilities of the back-end system.

Since the presentation logic and business logic are separated, it is easy to adapt the presentation node to new kinds of clients. The easiest approach is to use thin browser-based clients. But it is also possible to extend the presentation logic to new client platforms, for example, client Java applications or Web appliances like Web-enabled cellular phones or personal digital assistants (PDAs), without the need to change the business logic in the application node.

### **2.2.3 Application topology 5: considerations**

Traditional transaction monitors have been typically used to support the application router on the middle tier. These may still be appropriate unless you have a requirement to move rapidly to application topology 6 and beyond. For this level of functionality more advanced middleware is required, for example, WebSphere Enterprise Edition.

---

## **2.3 Application topology 6**

Topology 6 is similar to topology 5 in that there are multiple presentation channels and multiple applications. Topology 6 extends topology 5 by allowing you to make the third-tier applications seamless by integrating the business logic in the intermediate tier. This design is increasingly valuable because it can support multiple styles of client with a common intermediate tier.

### **2.3.1 Application topology 6: business driver**

As with topology 5, the business driver for this design is fast, highly scalable, highly available Web-enablement of existing business transactions. There are multiple presentation channels and multiple applications. Topology 6, however, provides customer-oriented support systems rather than product-oriented systems.

There is a need to hide the back-end application interfaces from the users to facilitate ease of use and to provide a single point of entry into the applications. Information or features from multiple applications need to be combined into a single response to the user.

For example, two banks merge and wish to provide a consistent customer-oriented view of their many back-end product applications.

### **2.3.2 Application topology 6: key features**

The key feature is the one-to-one-to-many relationship between the tiers at runtime. The application is divided into three logical tiers.

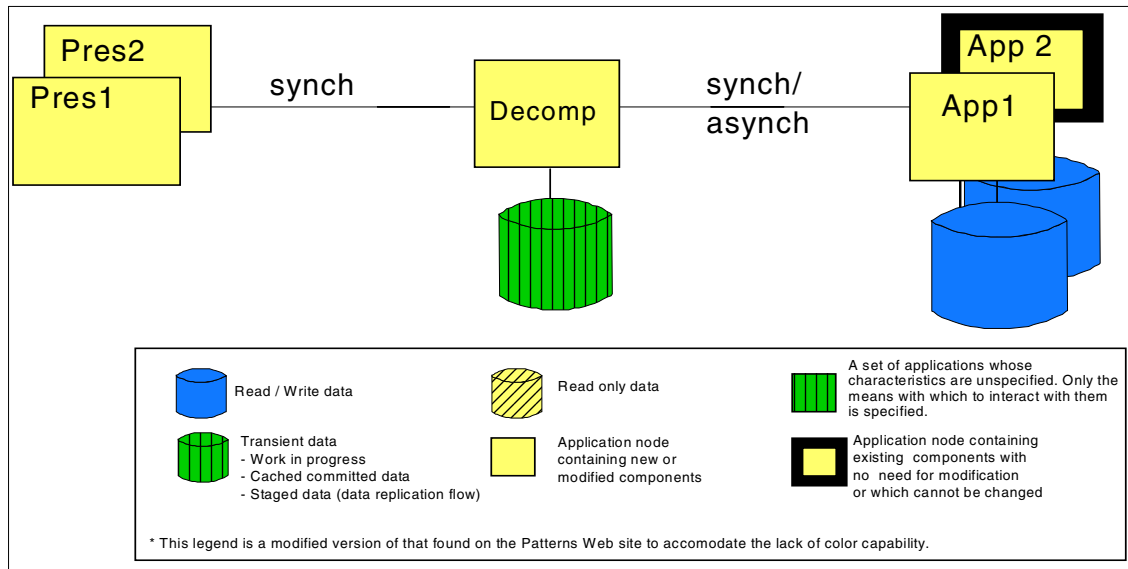


Figure 4. Application topology 6

- The presentation node in the first tier is responsible for the user interface into the Web application. It is responsible for all the Web-based presentation logic of the application.
- A presentation node is linked to the decomposition node in the second tier, which controls the application flow. The decomposition node will break down the user request into individual application requests, forward the requests to the applications and then gather the results to send back to the presentation node. The business logic of the application is divided between the applications and the decomposition node.
- The application nodes in the third tier are responsible for a portion of the business logic and for data access. There may be new applications developed specifically for the Web application or there may be existing legacy applications that may or may not require modification.

The communication between the presentation logic and decomposition node is synchronous, meaning that any request coming from the user interface invokes business logic on the decomposition node. After the business logic is executed and the results have been gathered, control is passed back to the presentation node that uses the results to update the user interface.

The connection between the decomposition node and existing or modified nodes can be asynchronous or synchronous. The communication type

depends on the communication characteristics and capabilities of the back-end system.

Since the presentation logic and business logic are separated, it is easy to adapt the presentation node to new kinds of clients. The easiest approach is to use thin browser-based clients. But it is also possible to extend the presentation logic to new client platforms, for example, client Java applications or Web appliances like Web-enabled cellular phones or personal digital assistants (PDAs), without the need to change the business logic in the application node.

### **2.3.3 Application topology 6: considerations**

There are a number of possible approaches to doing business request decomposition into multiple back-end transactions and recombining the responses into a single business response. These include:

1. RYO (roll-your-own) programming that issues multiple asynchronous requests to back-end systems and combines the responses. This is non-trivial!
2. Using a message broker plus a rules engine, possibly with a two-phase commit (2PC) or compensation mechanism.
3. Using a component broker with 2PC protocols.

This book will focus on the third approach.

---

## Chapter 3. Choosing the runtime topology

Selecting a runtime topology is a function of matching the application requirements with the correct version of the runtime topology, meaning that one size does *not* fit all.

Application topology 5 represents a starting point for delivering a sophisticated e-business application that unites the data integrity and performance of legacy applications. In the legacy back-end system there are various kinds of applications such as CICS, MQSeries, TXSeries, and relational databases.

Once the application topology has been chosen, it is time to choose the runtime topology. A runtime topology uses nodes to group functional and operational components. The nodes are interconnected to solve a business problem. Each application topology leads to one or more underlying runtime topologies.

---

### 3.1 An introduction to the node types

A runtime topology will consist of several nodes representing specific functions. Most topologies will consist of a core set of common nodes, with the addition of one or more nodes unique to that topology. To understand the runtime topologies you will need to review the following node definitions.

#### ***Integration server***

An integration server hosts application logic that can access and use information from existing databases, transaction functions from transaction monitor systems, and application capabilities from application packages. The integration server can access back-end applications individually or combine this information and function in new ways.

At a minimum the integration server acts as an integration point for multiple presentation tiers (for example, call centers, branch offices, Web browsers) so that they can share the infrastructure and applications on tiers 2 and 3.

#### ***Web application server***

A Web application server node is an application server that includes an HTTP server (also known as a Web server) and is typically designed for access by HTTP clients and to host both presentation and business logic.

The Web application server node is a functional extension of the informational (publishing-based) Web server. It provides the technology platform and

contains the components to support access to both public and user-specific information by users employing Web browser technology. For the latter, the node provides robust services to allow users to communicate with shared applications and databases. In this way it acts as an interface to business functions, such as banking, lending, and HR systems.

This node would be provided by the company on company premises, or hosted in the enterprise network inside a demilitarized zone for security reasons. In most cases, access to this server would be in secure mode, using services such as SSL or IPSec.

In the simplest design, this node can provide the management of hypermedia documents and diverse application functions. For more complex applications or those demanding stronger security, it is recommended that the application be deployed on a separate Web application server node inside the internal network.

Data that may be contained on the node includes:

- HTML text pages, images, multimedia content to be downloaded to the client browser
- JavaServer Pages (JSP) files
- Application program libraries, for example, Java applets for dynamic downloading to client workstations.

#### ***Application server node***

This node provides the infrastructure for application logic and may be part of a Web application server. It is capable of running both presentation and business logic but generally does not serve HTTP requests. When used with a Web server redirector the application server node will run both presentation and business logic. In other situations, it may be used for business logic only.

#### ***Public Key Infrastructure (PKI)***

PKI is a collection of standards-based technologies and commercial services to support the secure interaction of two unrelated entities (for example, a public user and a corporation) over the Internet. In the context of the topologies defined in this redbook, PKI supports the authentication of the server to the browser client, using the SSL protocol.

#### ***Domain Name Server (DNS) node***

The DNS node assists in determining the physical network address associated with the symbolic address (URL) of the requested information. The DNS is that of the Internet Service Provider, although DNS is implemented on the accessed site, too.

**User node**

This node is most frequently a personal computing device (PC, etc.) supporting a commercial browser, for example, Netscape Navigator or Internet Explorer. The level of the browser is expected to support SSL and some level of DHTML. Increasingly, designers should also consider that this node may be a pervasive computing device, such as a Personal Digital Appliance (PDA).

**Directory and security services node**

This node supplies information on the location, capabilities and various attributes (including user ID/password pairs and certificates) of resources and users, known to this Web application system. The node may supply information for various security services (authentication and authorization) and may also perform the actual security processing, for example, to verify certificates. The authentication in most current designs validates the access to the Web application server part of the Web server, but it can also authenticate for access to the database server.

**Protocol firewall and domain firewall nodes**

Firewalls provide services that can be used to control access from a less trusted network to a more trusted network. Traditional implementations of firewall services include:

- Screening routers (the protocol firewall in this design)
- Application gateways (the domain firewall)

The two firewall nodes provide increasing levels of protection at the expense of increasing computing resource requirements. The protocol firewall is typically implemented as an IP router, while the domain firewall is a dedicated server node.

**Web server redirector node**

In order to separate the Web server from the application server, a so-called *Web server redirector node* (or just redirector for short) is introduced. The Web server redirector is used in conjunction with a Web server. The Web server serves HTTP pages and the redirector forwards servlet and JSP requests to the application servers. The advantage of using a redirector is that you can move the application server behind the domain firewall into the secure network, where it is more protected than within the demilitarized zone (DMZ). Static pages can be served from the DMZ by this node.

The redirector can be implemented, for example, by either a reverse proxy server or by a Web server plug-in such as the servlet redirector function of IBM WebSphere Application Server Advanced Edition.

### **Existing applications and data node**

Existing applications are run and maintained on nodes that are installed in the internal network. These applications provide for business logic that uses data maintained in the internal network. The number and topology of these existing application and data nodes is dependent on the particular configuration used by these legacy systems.

## **3.2 Runtime topology A**

Runtime topology A implements application topology 5. This topology consists of a basic topology and one variation. In runtime topology A the integration server will act as a router, serving as an integration point for multiple presentation tiers accessing individual back-end applications.

### **3.2.1 Basic topology**

This basic runtime topology features a single Web application server (presentation node) residing in the DMZ, with the integration server (router node) and third-tier applications and data residing in the internal network.

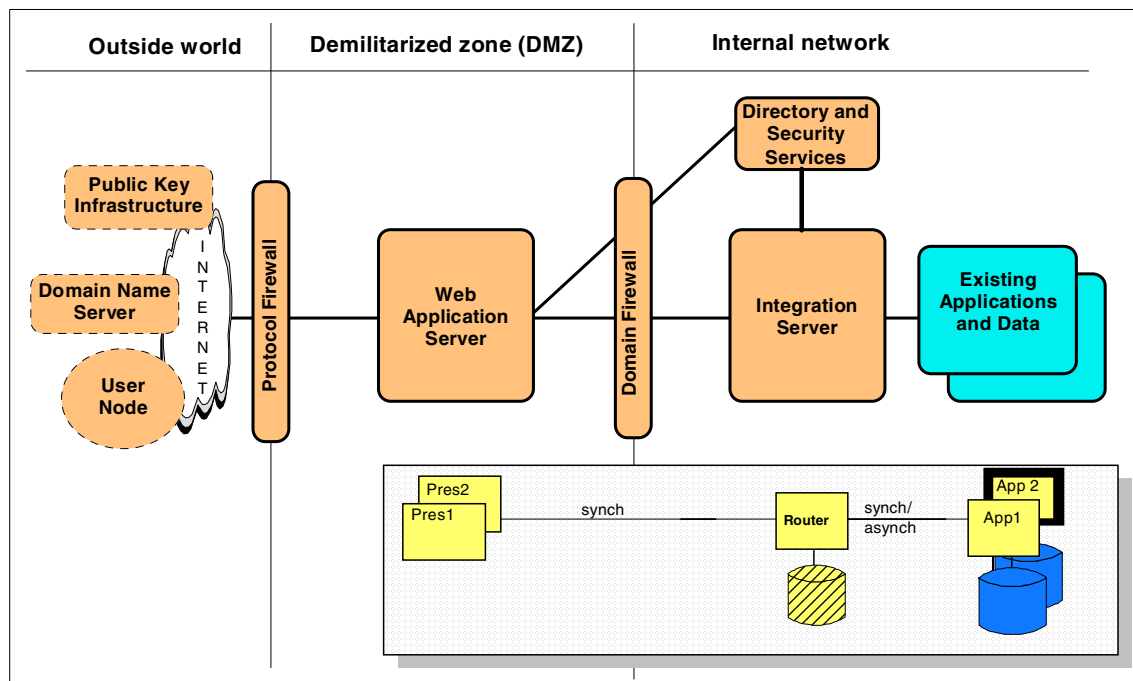


Figure 5. Runtime topology A - basic



The Web application server will contain the presentation logic and the controller logic. It will have the HTML pages, JSPs, and controlling servlets required for each of the back-end applications to be accessed. The Web application server will pass the requests to the integration server, which will determine the destination back-end application and route it on.

The primary business logic resides in the existing applications, with just enough logic in the integration server to determine the appropriate destination for a request.

The data and applications to be accessed are behind the domain firewall in the internal network. The integration server is also placed in the internal network for protection. Access to the Web application server resources is protected by the Web application server's security features, while access to the integration server's resources is protected by the integration server's security features. User information that is needed for authentication and authorization by both servers is stored in the directory and security services node behind the domain firewall in the internal network.

### **3.2.1.1 Benefits and limitations**

This runtime topology offers the following benefits:

- This runtime topology can be closely modeled on a single developer workstation.
- All sensitive persistent data is stored behind the DMZ.
- Business logic can completely reside in the secure network.

Although this topology has limited availability and failover capability, individual products offer features that could be folded into this topology, allowing duplicate servers that can take over in the event of a server failure. Horizontal scalability is also not shown, but there again, individual products may offer workload management features that could be folded into this topology to allow duplicate servers for distributed workload. Vertical scalability can be achieved by adding memory or processors, and/or creating multiple servers on the Web application server and integration server.

The number of clients that access the Web server simultaneously is limited by the capacity of the Web server. The actual numbers depend on the software and hardware platform used. Load balancing among the Web servers is a possibility.

Since the Web server is not separated from the application server, there is no additional security available and the business logic in the Web application server is protected only by the protocol firewall.

### 3.3 Runtime topology B

Runtime topology B implements application topology 6. This topology consists of a basic topology and one variation. In runtime topology B the integration server will act as a decomposition node, capable of taking requests from clients, servicing the requests from multiple back-end applications, and combining the received data into a unified response to the clients.

#### 3.3.1 Basic topology

This basic runtime topology features a single Web application server (presentation node) residing in the DMZ, with the integration server (decomposition node) and third-tier applications and data residing in the internal network.

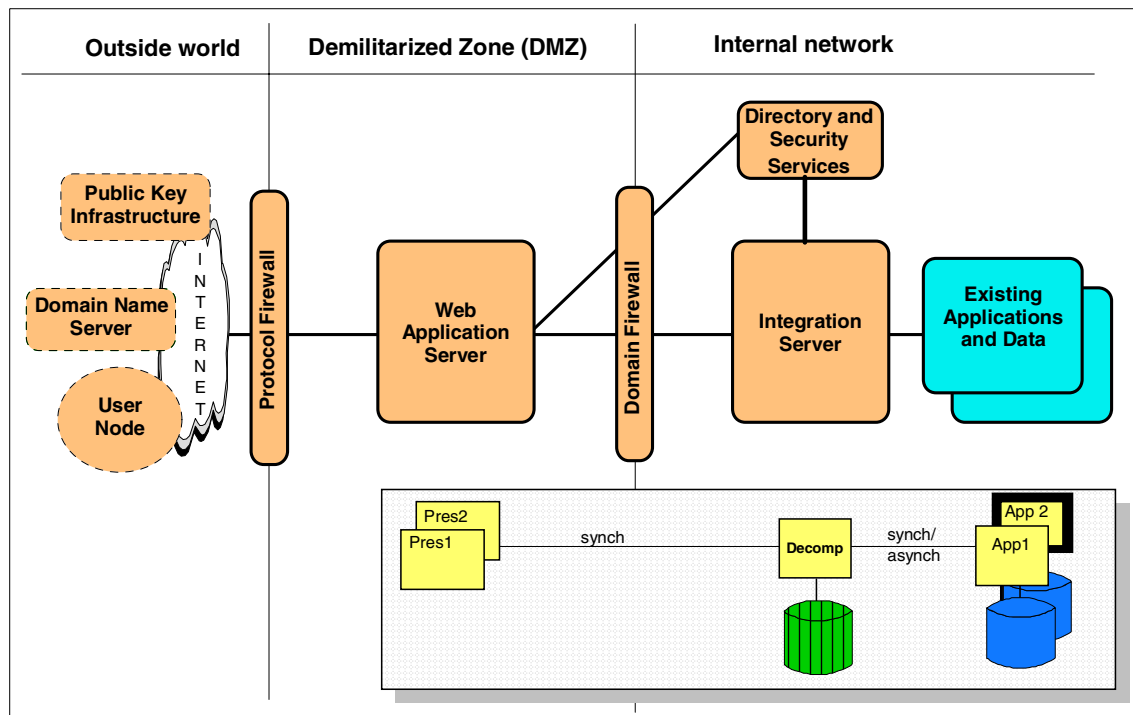


Figure 6. Basic runtime topology B

The presentation control flow resides in the Web application server. The application logic implementing the routing and decomposition functions resides on the integration server, available for reuse by the presentation tiers.

The third-tier data and applications to be accessed are behind the domain firewall in the internal network. The integration server is also placed in the internal network for protection. Access to the Web application server resources is protected by the Web application server's security features, while access to the integration server's resources is protected by the integration server's security features. User information, needed for authentication and authorization by both servers, is stored in the directory and security services node behind the domain firewall in the internal network.

The benefits and limitations of this topology are the same as those of topology A.

---

### **3.4 Variation to runtime topology A and B**

This variation applies to both runtime topologies A and B, building on the appropriate basic topology. The modifications introduced do not change the function of the integration server, which is what differentiates topology A from B. This variation will retain the characteristics of the appropriate basic topology with the exceptions noted.

#### **3.4.1 Variation 1 (emerging)**

Variation 1 builds on the basic topology by moving more of the function into the secure environment.

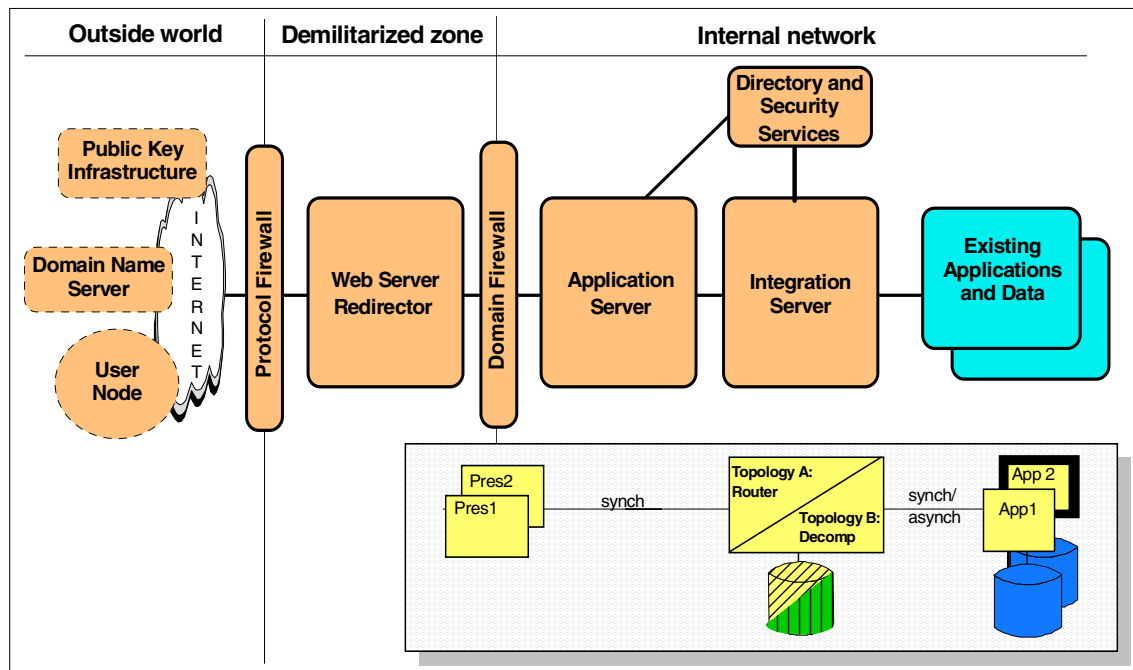


Figure 7. Runtime topology B variation 1

If you remember from our definitions, a Web application server node is a combination of an HTTP server and an application server on one machine. To provide an even more secure environment than the basic topology or variation 1 provides, it is possible to separate the Web server function from the application server, creating two new nodes. This allows you to leave the HTTP server in the DMZ, but move the application server into the internal network, where it is in a secure environment.

This separation of the Web server from the application server is done by using a *Web server redirector* node (or redirector for short). The Web server redirector node combines the HTTP server, which serves static HTTP pages, with a mechanism for forwarding dynamic servlet and JSP requests to a dedicated server.

The presentation logic will reside on the Web server and the application server, while the business logic should reside on the integration server.

#### **3.4.1.1 Benefits and limitations**

Since the Web server is separated from the application server, additional security is available. All business logic and the bulk of the presentation logic is protected by both the protocol and the domain firewall.

As with topologies A and B, there is limited availability, failover capability, and scalability. Individual products may offer features that could be folded into this topology, allowing duplicate servers and workload management features.

Since the requests to the application server need to be forwarded, you could see a performance degradation, depending on the redirector solution chosen.



---

## Chapter 4. Product mapping

Once the runtime topology is chosen, you will be ready to determine what products and platforms will fit your needs. This chapter outlines our product recommendations. These product mappings have been tested using the IBM Family Samples, available at:

<http://www.ibm.com/software/webservers/samples/>

The product mappings for runtime topologies A and B are identical. The difference between the two is in the application implementation.

All product mappings have been labeled “emerging”, meaning that the mappings shown here illustrate the use of newer technology that has been tested in the lab, but not necessarily in a commercial environment.

---

### 4.1 Product mappings for the basic topology (emerging)

These mappings show the products and platforms used in the implementation of the basic runtime topologies A and B. The basic topology features a Web application server between two firewalls and an integration server in the secure network.

Network security is provided by the two firewalls. The firewall between the Internet and the DMZ is called the protocol firewall. It is configured to be open on port 80 only thus, allowing only traffic using the HTTP protocol to flow from the clients in the Internet to the Web application server in the DMZ.

The domain firewall is configured to be open on one port to allow the Web application server to access the LDAP server, implemented with the IBM SecureWay Directory. Three ports are open to allow access to the integration server. This could increase depending on the configuration of Component Broker. The domain firewall restricts traffic based not only by protocol, but by host name.

IBM WebSphere Application Server Advanced Edition is used as the application server. Security for the Web applications (servlets, JSPs, and HTML) is provided by WebSphere Advanced’s security features. Users must authenticate using the LDAP server or the local operating system security.

The integration server is implemented using IBM’s Component Broker, a part of WebSphere Enterprise Edition. Component Broker accesses the back-end data. Both the data and the integration server are protected behind the domain firewall with DCE providing basic security and directory services.

Security can be enhanced by implementing the IBM SecureWay Policy Director.

Figure 8 shows the mapping for an environment where Windows NT is used as the platform of choice for the Web application server and integration server.

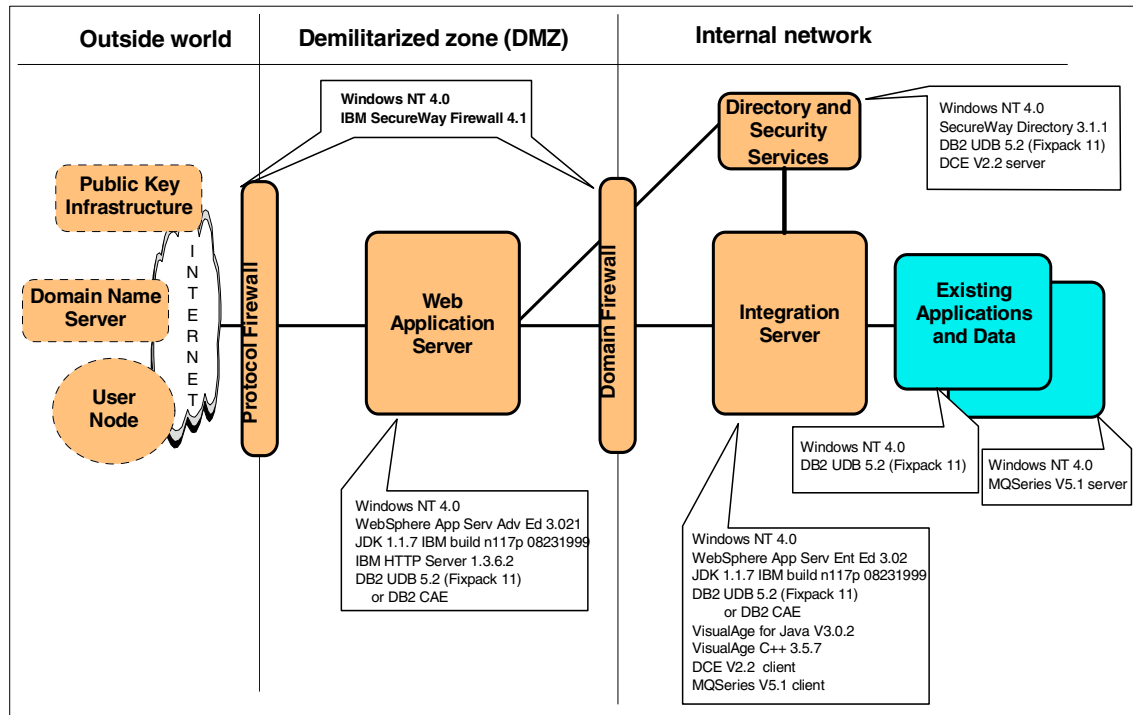


Figure 8. Basic runtime topology: Windows NT

Figure 9 shows the mapping for an environment where AIX is used as the platform of choice for the Web application server and integration server. The products on the other nodes were implemented on Windows NT but are also available on AIX.



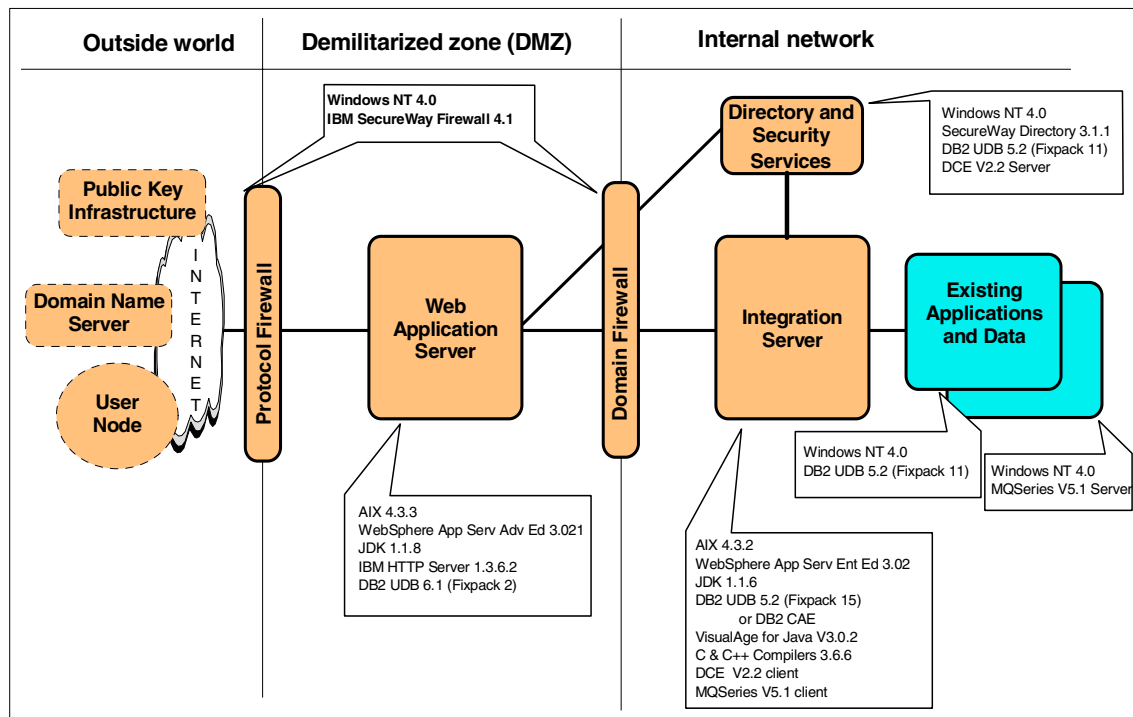


Figure 9. Basic runtime topology: AIX

## 4.2 Runtime topology variation 1 (emerging)

Variation 1 features putting both the application server and the integration server in the secure network. Requests are routed from the Web server to the application server using a Web server redirector.

In this product mapping the Web server redirector is implemented by the OSE Remote feature of WebSphere Advanced. WebSphere offers several options for forwarding requests to the application server. See 4.4, “Implementing a redirector” on page 40 for more information on these options.

The protocol firewall is identical to that in the basic topology, though the requirements for the domain firewall have changed. A minimum of two ports need to be open on the domain firewall to allow traffic to flow from the redirector to the application server. This number would increase by one for each application server. Additional ports may be needed depending on the the method used to configure the redirector.

As with the basic topology, IBM WebSphere Application Server Advanced Edition is used as the application server, which provides Web application security. The integration server is implemented using Component Broker. Component Broker accesses the back-end data. DCE provides basic security and directory services. Security can be enhanced by implementing the IBM SecureWay Policy Director.

Figure 10 shows the mapping for an environment where Windows NT is used as the platform of choice for the Web application server and integration server.

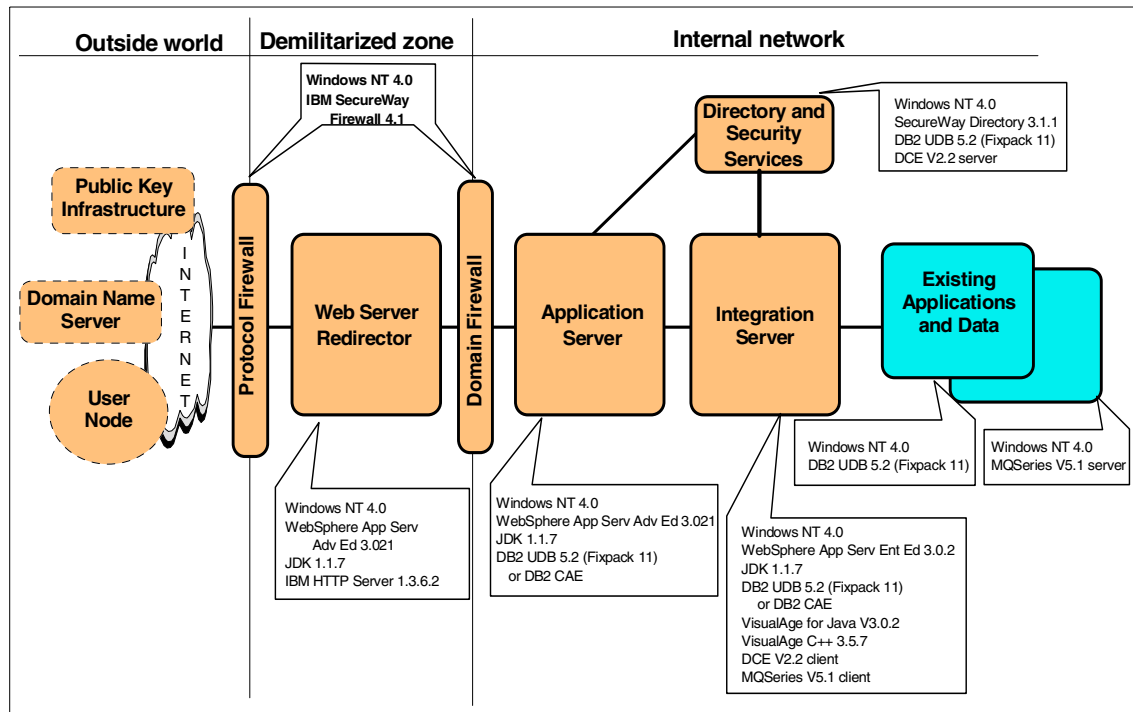


Figure 10. Runtime topology variation 1: Windows NT

Figure 11 shows the mapping for an environment where AIX is used as the platform of choice for the Web application server and integration server. The products on the other nodes were implemented on Windows NT but are also available on AIX.

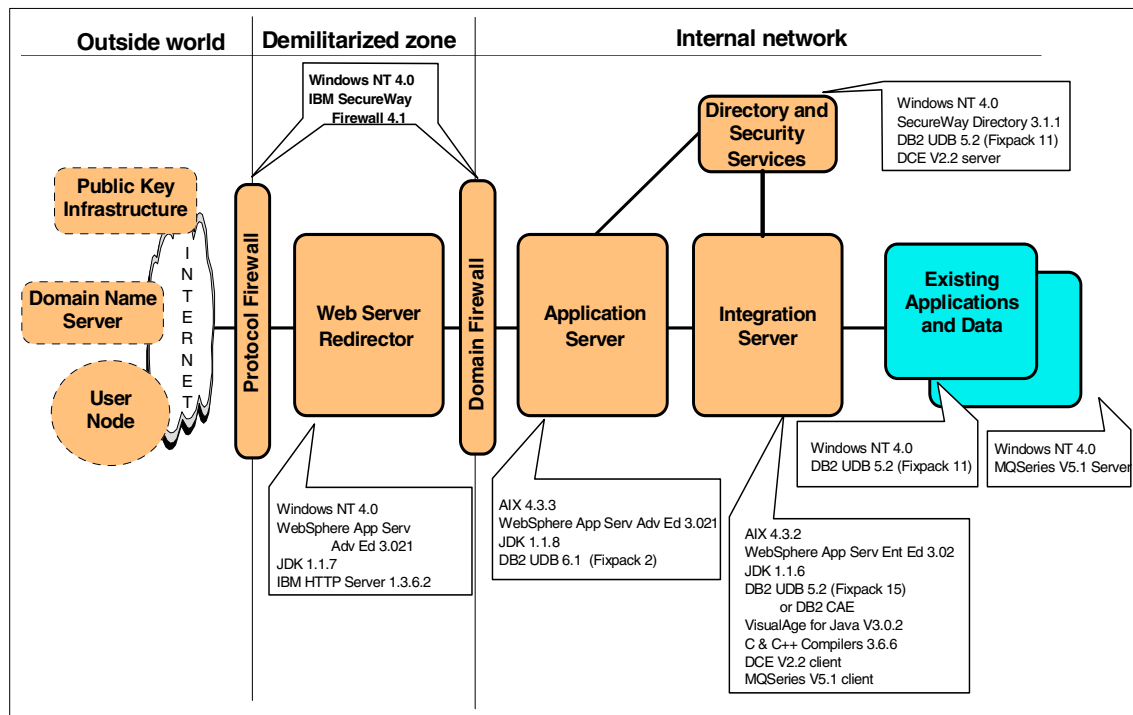


Figure 11. Runtime topology variation 1: AIX

## 4.3 Security

There are several ways to secure your application and resources using WebSphere Application Server Advanced Edition and Component Broker.

### 4.3.1 Component Broker security

This will be a brief overview of Component Broker security. More information on Component Broker security and examples of how to implement it can be found in:

- *Application Server Solution Guide: Enterprise Edition*, SG24-5320
- *WebSphere Application Server Enterprise Edition Component Broker 3.0 First Steps*, SG24-2033

To make the network more secure, Component Broker defines the following security mechanisms:

- **Authentication** allows the servers and the clients to be authenticated to the CORBA system. Component Broker uses the DCE security server as a 3-party authentication scheme. DCE manages a user registry to store all the information of the users, and using the Kerberos/V protocol, it verifies the users with knowledge-based authentication. Component Broker supports the certificate-based authentication using SSL protocol. In this version of CB, only a Java client can use SSL as the authentication mechanism.
- **Authorization** controls the access to the business objects in application servers. When the application server receives a method request from a principal, it checks an ACL to see if the permission to invoke this method has been granted to the principal. By default, CB uses a coarse level of authentication, which means that all the authenticated users can access all the objects in the system. To use a fine level of authorization, we register the objects we would like to protect to a repository and set all the permissions of the principals for each protected object. In order to use a fine level of authorization, we need the IBM SecureWay Policy Director product, which is used by CB. Specifically, the CB uses the following Policy Director components for the fine level of authentication:
  - An authorization server, which protects objects registered to the authorization repository by enforcing the authorization policies defined in the ACL.
  - A management console, which is a GUI to set an ACL to objects.
  - A NetSEAT client, which talks to an authorization server. The NetSEAT client is built into the CB server runtime.
  - An authorization database manager, which maintains a secure authorization policy directory used by authorization servers.
- **Delegation** is the mechanism used to transfer the credentials between objects. Component Broker supports no delegation and simple delegation or impersonation.
- **Credential Mapping** allows you to define a credential mapping between the CB security credential and those of an entity in the data system's security mechanism. Credential mapping supports two different types of mapping. The simple type uses the information in the system management to map the credential. It is not flexible because you can set up only one userid and password for a server or a server group, and it is not secure because all the userids and passwords are stored in clear. SSO-enable allows CB to store the credential mapping in a secure database. This version of Component Broker uses GSO database as secure storage.

Thus, you need to install the IBM Global Sign-On product in order to use this security feature.

### 4.3.2 WebSphere Advanced security

WebSphere Advanced provides an integrated security model to configure security on your Web resources. This can be centrally configured through the WebSphere administrative console. This will be a brief overview of WebSphere Advanced security. More information can be found in:

- *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864
- *IBM WebSphere Standard/Advanced 3.02 Security Overview*, an IBM white paper available at:

<http://www.ibm.com/software/webservers/appserv/whitepapers.html>

#### 4.3.2.1 Authentication

In WebSphere, authentication between a user and the WebSphere application server can be specified in terms of:

- User registry - This is where the user and group information will be stored. The user registry can be an implementation of LDAP, for example, SecureWay Directory, Netscape, etc., or the local operating system. Note that each WebSphere administrative domain can have only the user registry.
- Authentication mechanism - After the user has provided the required data, the authentication mechanism will validate it against an associated user registry. Two types of authentication mechanism are supported:
  - Lightweight Third Party Authentication (LTPA)
  - Native operating system
- Challenge mechanisms - The challenge mechanism specifies how a server will challenge and retrieve authentication data from the user. It can be of the form:
  - None: Security runtime does not challenge user for authentication data.
  - Basic: A user is challenged for ID and password.
  - Certificate: Mutual authentication over SSL.
  - Custom: The ability to specify a custom HTML page to retrieve a user's ID and password.

#### **4.3.2.2 Authorization**

The authorization model in WebSphere 3.0 is based on the classic capability model. In this model, the permissions required to perform a particular operation are associated with a principal.

When a principal requests to perform an operation on a resource, the security runtime considers a set of permissions to be the “required” permissions for performing the operation on the resources. If the requesting principal has been granted at least one of the required permissions, then the security subsystem authorizes the request to be processed.

#### **4.3.2.3 Delegation**

Delegation is the process of forwarding a principal's credentials along with the cascaded downstream requests that occur within the context of work that the principal originated or is having performed on its behalf.

When a client uses an intermediary to invoke a method on a target resource, the intermediary invokes the method assuming a certain identity. The identity could be:

- Client: identity of client requesting the method invocation
- System: identity of server hosting the intermediary resource which will eventually invoke the method
- Specified: a different identity

#### **4.3.2.4 HTTP single sign-on (SSO)**

When you enable HTTP single sign-on, user authentication credentials are preserved across multiple applications in the same domain, for example:

- Cooperating but disparate Web servers
- Cooperating applications such as the IBM OnDemand Server and Windows NT Suites.

With SSO, your application will then avoid repeated requests of user security credentials. However, if your application wants to use the SSO feature, then it must use an LTPA registry (LDAP for example).

---

### **4.4 Implementing a redirector**

There are several methods of implementing a network scenario where the Web server resides in the DMZ, while the application servers reside in the internal network.

- WebSphere's Remote OSE feature

- WebSphere's servlet redirector (base or stand-alone)
- Reverse proxy (only choice for releases of WebSphere prior to V3.0)

Each method has characteristics that should be considered when making a choice of which to use. For more information see:

[http://www.ibm.com/software/webservers/appserv/dmz\\_v3.html](http://www.ibm.com/software/webservers/appserv/dmz_v3.html)

In our implementation of the runtime topologies, we used the WebSphere Remote OSE feature. This is the preferred method of those provided by WebSphere because of its performance advantages over the servlet redirectors.

#### **4.4.0.1 Remote OSE**

The Remote OSE feature, available in WebSphere Advanced Edition V3.021, provides a simple, streamlined method of routing requests from a Web server to a remote application server. Open Servlet Engine (OSE) is a communication protocol originally intended for communication between a Web server and application server located on one physical machine. Remote OSE allows the application server to be physically located elsewhere in the network, presumably behind a firewall in a secure network, while the Web server resides in a DMZ.

Configuring the HTTP Web server for Remote OSE offers the following benefits:

- Significant performance improvement over the servlet redirector
- Support for WebSphere Advanced security
- Supports Network Address Translation (NAT)
- Does not require a Java process on the Web server machine

The following are security considerations:

- Remote OSE does not support SSL between the Web server and application server. To use SSL you will need to use the servlet redirector.
- Remote OSE requires one open port on the firewall to retrieve administrative information from the WebSphere administrative server and an additional open port for each servlet engine.

#### **4.4.0.2 Using WebSphere's servlet redirector**

The servlet redirector is a process that distributes servlet requests to machines remote to the Web server. With WebSphere Application Server 3.021 there are two configurations of the redirector available, base ("thick") and stand-alone ("thin").

**Base (“thick”) redirector**

The base redirector runs as part of the administrative server, and therefore has the overhead of requiring the administrative server’s infrastructure (the database). It uses the RMI/IIOP protocol.

The features of the base redirector include:

- Configuration and management from the WebSphere administrative console on either the redirector’s or application server’s node.
- There is a full WebSphere installation in the DMZ, requiring access to a DB2 administrative database, either in the DMZ, or through the firewall.

The following are security considerations:

- SSL is supported between the Web server and application server.
- Support for WebSphere Advanced security.
- The base redirector requires three open ports on the firewall plus an additional port for each application server.
- There is no support for NAT firewalls.

**Stand-alone (“thin”) redirector**

The stand-alone redirector does not require the administrative server’s infrastructure and therefore does not suffer from that overhead. It does, however, use RMI/IIOP and therefore has more of a performance impact than the lighter-weight Remote OSE option.

The features of the stand-alone redirector include:

- Configuration is done by running scripts on a regular basis that extract configuration information from the remote WebSphere administrative server. This means that the redirector can be out of sync whenever the application server has changed and the changes have not been propagated to the Web servers.
- There is no WebSphere administrative database in the DMZ. That is protected behind the firewall.

The following are security considerations:

- SSL is supported between the Web server and application server.
- The base redirector requires three open ports on the firewall plus an additional port for each application server.
- It does not support WebSphere Advanced security. If you choose to turn on security in WebSphere, you will not be able to run the stand-alone redirector.
- It does not support NAT firewalls.



#### 4.4.0.3 Using reverse proxy

An alternative way to implement a redirector is to use a reverse proxy. A reverse proxy is a method of making a proxy server transparent to the client. A reverse proxy listens on port 80 for requests that have a certain format. It then forwards those requests to an HTTP server that resides on the Web application server. The requests are then fulfilled and passed back to the reverse proxy to the client.

When a proxy server is configured for reverse proxy, it appears to the client to be the originating server. The client is not aware that the request is actually being sent to another server.

Reverse proxies perform relatively fast, can work with NAT firewalls, and support WebSphere security.

This option is the only one that does not have any WebSphere software in the DMZ. The configuration takes place outside the scope of WebSphere and implementation details vary according to the particular reverse proxy product.

---

### 4.5 Workload management (WLM)

Both Component Broker and WebSphere Advanced Edition offer features that allow incoming requests to be distributed among multiple identical servers, on one or more physical machines. This allows you to design a network that provides some degree of fault tolerance and scalability.

#### 4.5.1 Component Broker

A Component Broker network consists of one or more Component Broker hosts. These hosts can be grouped logically for organization and to simplify systems management. Workload management of applications can be achieved through the use of controlled server groups consisting of servers residing on any of the hosts in the network.

A controlled server group is a server group that has been configured with a managing host to provide workload distribution of client work requests across the servers in the group. One host in the group will be the managing host. The managing host will have two special servers:

- **Server Group Control Point (SGCP):** A special server that resides on the managing host of a controlled server group that controls the servers in the group. It exchanges information between clients and application servers, allowing clients to know which servers in the group are active.

- **Server Group Gateway (SGGW):** A special server that resides on the managing host of a controlled server group used by Component Broker to present the server group as a single server to non WLM-enhanced clients. To accommodate a large number of servers that are needed to service an application's workload, consider creating more than one server group (each having its own managing host) so that all requests are not routed through one SGGW server.

Applications are configured onto the server group, making the applications installed on each server in the group the same.

For an application to benefit from automatic workload distribution, it must be installed in a controlled server group and must include designated workload management objects. These "workload managed objects" are stored in a container in the application family (not visible in the System Manager). Applications designate objects for workload distribution by configuring them into workload managing containers (part of the application packaging step when using object builder). These objects must be coded within special guidelines and restrictions that may not be appropriate for all objects.

In terms of workload management, there are two different types of clients, WLM-enhanced clients and non-WLM enabled clients.

WLM-enhanced clients are those that use the Component Broker VisualAge C++ ORB. These clients use the SGCP server for workload distribution. WLM-enhanced clients can be configured to use a binding policy that randomly chooses a server from the server group. Once the server is chosen a binding affinity is established between the client or server process and the selected server. This server will continue to process all requests for workload managed objects for that server group, until that server is no longer available.

Non-WLM enabled clients are those that use an ORB that does not support client-initiated workload distribution, such as the Java ORB and Microsoft Visual C++ ORB supplied with Component Broker. Non-WLM enabled clients use the SGGW server for workload distribution. The SGGW server acts as the single WLM-enabled client of its controlled server group. Because it appears as one client, the result is that all requests are routed to the same application server within the server group unless there is a way of distinguishing the clients. Enabling the Security Service and having each end user authenticate with their own user ID will allow the SGGW server to distinguish between clients and maintain a separate client affinity mapping for each client. In the event that security is not enabled, the SGGW server will use the host name of the client if possible to distinguish between clients.

For detailed information on administering a Component Broker network see *WebSphere Application Server Enterprise Edition Component Broker System Administration Guide*, SC09-4445.

#### **4.5.2 WebSphere Advanced Edition**

The IBM WebSphere Application Server allows you to leverage processor capacity or additional processors by means of application "cloning". Cloning means that a given application is duplicated such that the clients cannot distinguish between the clones. In addition, each WebSphere Application Server (this is WebSphere's term for grouping a servlet engine and related resources) is running in its own JVM. This allows the use of more than one JVM with WebSphere.

Before cloning, you first have to create a model of that resource. Clones are created from a model. After you clone a resource, modifying the model automatically propagates the same changes to all of the clones. You can efficiently administer several copies of a server or other resource by administering its model.

You can also clone servlets, servlet engines, Web applications, EJB containers, and enterprise beans.

The application server balances the workload of the clones running on a server automatically. Thus, you do not have to worry about the machine's utilization. All of that is done automatically by the application server.

There are circumstances where cloning is desirable even if you have only one processor installed. This can be the case if you have an application that is spending most of its time waiting for some resources. During this time, additional requests can be served by the application's clones. Also, if automatic tasks such as garbage collection take too long to complete, cloning may prove a viable alternative on a single-processor machine.

In addition to all the above-mentioned advantages, cloning also increases the availability of a particular Web application as well as the failover capability. If any of the clones fails, the other clones take over the workload.

Overall, it is highly application dependent as to whether or not a performance improvement can be achieved by cloning of applications.

As with all alternatives there are also disadvantages that you have to consider:

1. If you do not require session affinity and want all session-related information to be transparent to the users, there is some performance penalty because all session information needs to be saved and retrieved from a database. Depending on the amount of data, this penalty may prove to be very expensive.
2. If your application assumes that it is running on a dedicated machine (even on a dedicated JVM), cloning will not be an issue for you because it cannot be determined in advance on which machine your application will execute the next time a request is served.

See *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864 for more information on using clones in a WebSphere Advanced environment.

---

## **Part 2. User-to-Business patterns: guidelines**



---

## Chapter 5. Technology options

This chapter looks at the technologies that you should consider for Web applications based upon the open standards and Java-based programming model of the IBM Application Framework for e-business. We do not attempt to cover the many technologies that can be used in developing Web applications, such as Perl or server-side JavaScript. These recommended technologies are outlined in the *IBM Application Framework for e-business Architecture Overview* white paper at:

[http://www.ibm.com/software/ebusiness/arch\\_overview.html](http://www.ibm.com/software/ebusiness/arch_overview.html)

We will look at the technologies as they apply to both the client and the server side of the application. Some technologies such as Java and XML can apply to both. Also, the selection of client-side technologies used in your design will require consideration for the server side such as to whether to store, or dynamically create, elements for the client side.

The sections that follow detail a number of technologies that you will want to consider in your design.

We recommend the following as technologies that are central to the Application Framework and its programming model:

- HTML
- Java servlets and JavaServer Pages
- XML
- Connectors
- Enterprise JavaBeans
- JDBC
- Additional enterprise Java APIs

These technologies are used in the context of the following logical model for an e-business application. This model, which has similarities to the Model-View-Controller approach in GUI development, characterizes the presentation logic as consisting of interaction control (implemented by Java servlets) and page construction (implemented by JavaServer Pages). The business logic may be implemented using beans and/or enterprise beans depending on the transactional characteristics of the application. The business logic may need to access external resources using the appropriate connector technology.

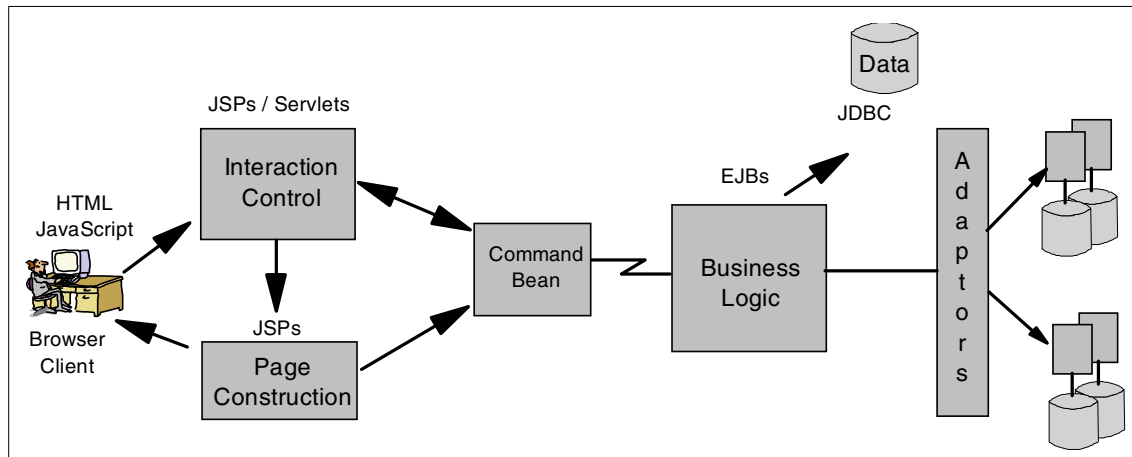


Figure 12. The logical structure of an e-business application using the recommended core technologies

We also include some discussion of the following technologies and the limitations involved in their usage:

- DHTML
- JavaScript
- Java applets

For more information, see the *IBM Application Framework for e-business Architecture Overview: Understanding Technology Choices* white paper, upon which significant portions of this chapter are based:

<http://www.ibm.com/software/ebusiness/buildapps/understand.html>

## 5.1 Web client

The Application Framework recommends the following technology model for a Web client.



## Client Model

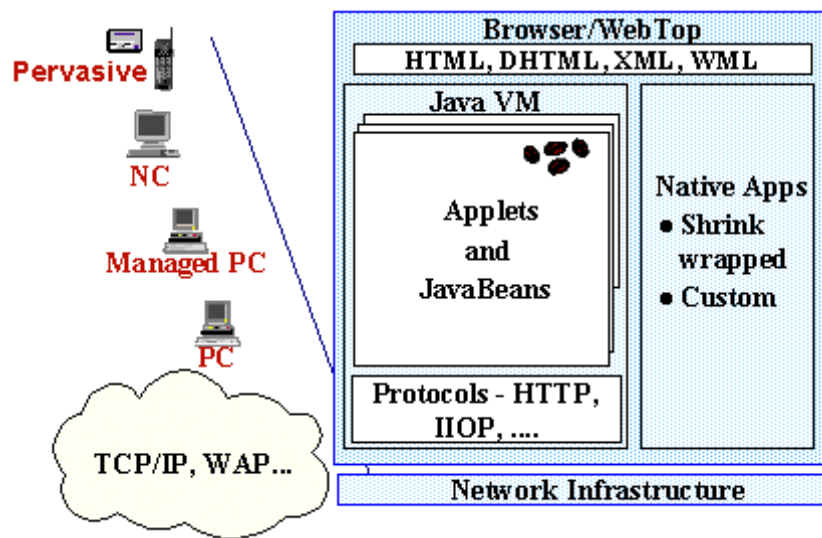


Figure 13. Web client technology model

The clients are “thin clients” with little or no application logic. Applications are managed on the server and downloaded to the requesting clients. The client portions of the applications should be implemented in HTML, dynamic HTML (DHTML), XML, and Java applets.

The following sections outline some of the possible technologies that you should consider, but remember that your choices may be constrained by the policy of your customer or sponsor. For example, for security reasons, only HTML is allowed in the Web client at some government agencies.

Although the Application Framework recommends thin clients, there is also opportunity for developing applications geared toward other client types.

### 5.1.1 Web browser

A Web browser is a fundamental component of the Web client. For PC-based clients, the browser typically incorporates support for HTML, DHTML, JavaScript, and Java. Some browsers are beginning to add support for XML as well. Under user control, there is a whole range of additional technologies

that can be configured as “plug-ins”, such as RealPlayer from RealNetworks or Macromedia Flash.

As an application designer you must consider the level of technology you can assume will be available in the user’s browser, or you can add logic to your application to enable slight modifications based upon the browser level. Regarding plug-ins, you need to consider what portion of your intended user community will have that capability.

For an e-business application that is to be accessed by the broadest set of users with varying browser capabilities, the client is often written in HTML with no other technologies. On an exception basis, limited use of other technologies, such as using JavaScript for simple edit checks, can then be considered based on the value to the user and the policy of the organization for whom the project is being developed.

The emergence of pervasive devices introduces new considerations to your design with regard to the content streams that the device can render and the more limited capabilities of the browser. For example, Wireless Application Protocol (WAP) enabled devices render content sent in Wireless Markup Language (WML).

### **5.1.2 HTML**

HTML is a document markup language with support for hyperlinks, that is rendered by the browser. It includes tags for simple form controls. Many e-business applications are assembled strictly using HTML. This has the advantage that the client-side Web application can be a simple HTML browser, enabling a less capable client to execute an e-business application.

The HTML specification defines user interface (UI) elements for text with various fonts and colors, lists, tables, images, and forms (text fields, buttons, checkbooks, radio buttons). These elements are adequate to display the user interface for most applications. The disadvantage, however, is that these elements have a generic look and feel, and they lack customization. As a result, some e-business application developers augment HTML with other user interface technologies to enhance the visual experience, subject to maintaining access by the intended user base and compliance with company policy on Web client technologies.

Because most Web browsers can display HTML Version 3.2, this is the lowest common denominator for building the client side of an application.

### 5.1.3 Dynamic HTML (DHTML)

DHTML allows a high degree of flexibility in designing and displaying a user interface. In particular, DHTML includes cascading style sheets (CSS) that enable different fonts, margins, and line spacing for various parts of the display to be created. These elements can be accurately positioned using absolute coordinates.

Another advantage of DHTML is that it increases the level of functionality of an HTML page through a document object model and event model. The document object enables scripting languages such as JavaScript to control parts of the HTML page. For example, text and images can be moved about the screen, and hidden or shown, under the command of a script. Also, scripting can be used to change the color or image of a link when the mouse is moved over it, or to validate a text input field of a form without having to send it to the server.

Unfortunately there are several disadvantages with using DHTML. The greatest of these is that two different implementations (Netscape and Microsoft) exist and are found only on the more recent browser versions. A small, basic set of functionality is common to both, but differences appear in most areas. The significant difference is that Microsoft allows the content of the HTML page to be modified by using either JScript or VBScript, while Netscape allows the content to only be manipulated (moved, hidden, shown) using JavaScript.

Because of browser compatibility issues, DHTML is not recommended in environments where mixed levels and brands of browsers are present.

### 5.1.4 XML (client-side)

XML allows you to specify your own markup language with tags specified in a Document Type Definition (DTD). Actual content streams are then produced that use this markup. The content streams can be transformed to other content streams by using XSL (eXtensible Stylesheet Language).

For PC-based browsers, HTML is well established for both document content and formatting. The leading browsers have significant investments in rendering engines based on HTML and a Document Object Model (DOM) based on HTML for manipulation by JavaScript.

XML seems to be evolving to a complementary role for active content within HTML documents for the PC browser environment.

For new devices, such as WAP-enabled phones and voice clients, the data content and formatting is being defined by new XML schema (DTD), WML for WAP phone and VoiceXML for voice interfaces.

For most Web application designs, you should focus your attention on the use of XML on the server side. See 5.2.4, “XML” on page 59 for additional discussion of the server side use of XML.

### 5.1.5 JavaScript

JavaScript is a cross-platform object-oriented scripting language. It has great utility in Web applications because of the browser and document objects that the language supports. Client-side JavaScript provides the capability to interact with HTML forms. You can use JavaScript to validate user input on the client and help improve the performance of your Web application by reducing the number of requests that flow over the network to the server.

ECMA, a European standards body, has published a standard (ECMA-262) that is based on JavaScript (from Netscape) and JScript (from Microsoft) called ECMAScript. The ECMAScript standard defines a core set of objects for scripting in Web browsers. JavaScript and JScript implement a superset of ECMAScript. You can find the ECMAScript Language Specification at:

<http://www.ecma.ch/stand/ECMA-262.htm>.

To address various client-side requirements, Netscape and Microsoft have extended their implementations of JavaScript in Version 1.2 by adding new browser objects. Because Netscape's and Microsoft's extensions are different from each other, any script that uses JavaScript 1.2 extensions must detect the browser being used, and select the correct statements to run.

The use of JavaScript on the server side of a Web application is not recommended, given the alternatives available with Java. Where your design indicates the value of using JavaScript, for example for simple edit checking, use JavaScript 1.1, which contains the core elements of the ECMAScript standard.

*JavaScript: The Definitive Guide, Third Edition*, by David Flanagan, is an excellent book on JavaScript that details the JavaScript objects and methods listing their origin and JavaScript level.

### 5.1.6 Java applets

The most flexible of the user interface (UI) technologies that can be run in a Web browser is offered by the Java applet. Java provides a rich set of UI elements that include an equivalent for each of the HTML UI elements. In

addition, because Java is a programming language, an infinite set of UI elements can be built and used. There are many widget libraries available that offer common UI elements, such as tables, scrolling text, spreadsheets, editors, graphs, charts, etc.

A Java applet is a program written in Java that is downloaded from the Web server and run on the Web browser. The applet to be run is specified in the HTML page using an APPLET tag:

```
<APPLET CODEBASE="/mydir" CODE="myapplet.class" width=400 height=100>
  <PARAM NAME="myParameter" VALUE="myValue">
</APPLET>
```

For this example, a Java applet called myapplet will run. An effective way to send data to an applet is with the use of the PARAM tag. The applet has access to this parameter data and can easily use it as input to the display logic.

Java can also request a new HTML page from the Web application server. This provides an equivalent function to the HTML FORM submit function. The advantage is that an applet can load a new HTML page based upon the obvious (a button being clicked), or the unique (the editing of a cell in a spreadsheet).

A characteristic of Java applets is that they seldom consist of just one class file. On the contrary, a large applet may reference hundreds of class files. Making a request for each of these class files individually can tax any server and also tax the network capacity. However, packaging all of these class files into one file reduces the number of requests from hundreds to just one. This optimization is available in many Web browsers in the form of either a JAR file or a CAB file. Netscape and HotJava support JAR files simply by adding an `ARCHIVE="myjarfile.jar"` variable within the APPLET tag. Internet Explorer uses CAB files specified as an applet parameter within the APPLET tag. In all cases, executing an applet contained within a JAR/CAB file exhibits faster load times than individual class files. While Netscape and Internet Explorer use different APPLET tags to identify the packaged class files, a single HTML page containing both tags can be created to support both browsers. Each browser simply ignores the other's tag.

A disadvantage of using Java applets for UI generation is that the required version of Java must be supported by the Web browser. Thus, when using Java, the UI part of the application will dictate which browsers can be used for the client-side application. Note that the leading browsers support variants of the JDK 1.1 level of Java and they have different security models for signed applets.

A second disadvantage of Java applets is that any classes such as widgets and business logic that are not included as part of the Java support in the browser must be loaded from the Web server as they are needed. If these additional classes are large, the initialization of the applet may take from seconds to minutes, depending upon the speed of the connection to the internet.

Because of the above shortcomings the use of Java applets is not recommended in Internet environments where mixed levels and brands of browsers are present. Small applets may be used in rare cases where HTML UI elements are insufficient to express the semantics of the client-side Web application user interface. If it is absolutely necessary to use an applet in an Internet environment, care should be taken to include UI elements that are core Java classes whenever possible. Applets work better in an intranet environment, where there is some expectation of consistent browser levels.

## 5.2 Web application server

The Application Framework recommends the following technology model for a Web application server.

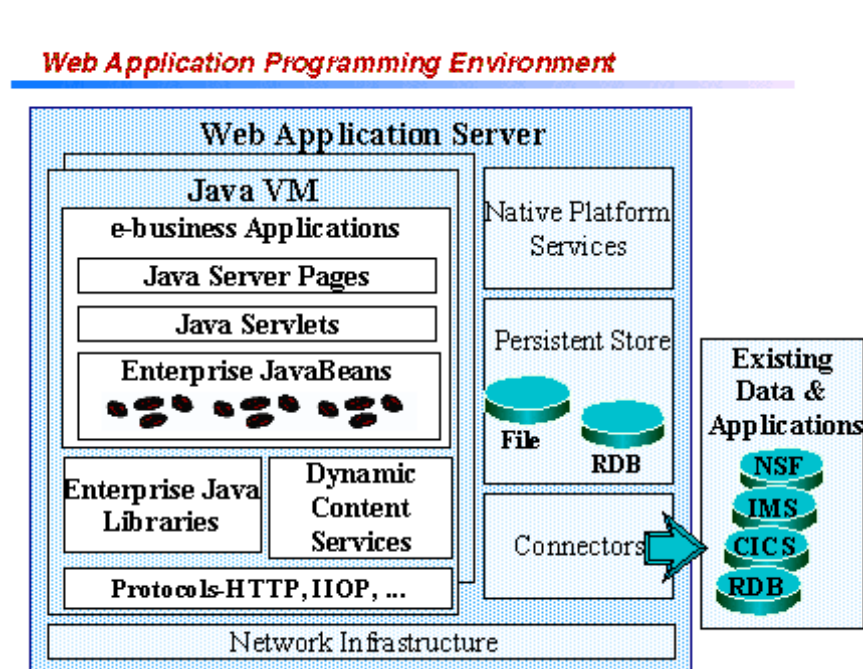


Figure 14. Web application server technology model

We will assume in this section that you will be using a Web application server and server-side Java. While there have been many other models for a Web application server, this is the one that is experiencing widespread industry adoption. For more detail on the Java APIs discussed in this section see *Java Enterprise in a Nutshell* by David Flanagan, Jim Farley, William Crawford and Kris Magnusson.

Before looking at the technologies and APIs available in the Web application programming environment, let's have a word about two fundamental operational components on this node, the HTTP server and the Java Virtual Machine (JVM). For production applications, these essential components should be chosen for their operational characteristics in areas such as robustness, performance, and availability.

Relating the Model-View-Controller design structure so often used in user interfaces to the Web application programming model:

- The View is generally best implemented using JavaServer Pages.
- The Interaction Controller, which is primarily concerned with processing the HTTP request and invoking the correct business or UI logic, often lends itself to implementation as a servlet.
- The Model is represented to the View and Interaction Controller via a set of JavaBean or Enterprise JavaBean components.

### 5.2.1 Java servlets

Servlets provide a replacement for CGI-based techniques in Web programming. Servlets are small Java programs that run on the Web application server. They interact with the servlet engine running on the Web application server through HTTP requests and responses, which are encapsulated as objects in the servlet.

One of the attractions of using servlets is that the API is a very accessible one for a Java programmer to master. The most current level of the servlet API is 2.2. To learn more about the servlet API visit:

<http://www.javasoft.com/products/servlet/>.

Servlets are a core technology in the Web application programming model. They are the recommended choice for implementing the “Interaction Controller” classes that handle HTTP requests received from the Web client.

### 5.2.2 JavaServer Pages (JSP)

JSP files were designed to simplify the process of creating pages by separating Web presentation from Web content. In the page construction logic of a Web application, the response sent to the client is often a combination of template data and dynamically generated data. In this situation, it is much easier to work with JSP files than to do everything with servlets.

The chief advantage JSP files have over Java servlets is that they are closer to the presentation medium. A JavaServer *Page* is an HTML page. JSP files can contain all the HTML tags that Web authors are familiar with. A JSP may contain fragments of Java code that encapsulate the logic that generates the content for the page. These code fragments may call out to beans to access reusable components and back-end data. To learn more about JSP files visit : <http://www.javasoft.com/products/jsp/>.

JSP files are compiled into servlets before being executed on the Web application server. The most current level of the JSP API is 1.1.

JSP files are the recommended choice for implementing the “view” that is sent back to the Web client. For those cases where the code required on the page will be a large percentage of the page, and the HTML minimal, writing a Java servlet will make the Java code much easier to read and therefore maintain.

### 5.2.3 JavaBeans

JavaBeans is an architecture developed by Sun Microsystems, Inc. describing an API and a set of conventions for reusable, Java-based components. Code written to Sun’s JavaBeans architecture is called Java beans or just beans. One of the design criteria for the JavaBean API was support for builder tools that can compose solutions that incorporate beans. Beans may be visual or non-visual.

Beans are recommended for use in conjunction with servlets and JSP files in the following ways:

- As the client interface to the “Model Layer”. An “Interaction Controller” servlet will use this bean interface.
- As the client interface to other resources. In some cases this may be generated for you by a tool.
- As a component that incorporates a number of property-value pairs for use by other components or classes. For example, the JavaServer Pages specification includes a set of tags for accessing JavaBean properties.



#### 5.2.4 XML

XML and XSL style sheets can be used on the server side to encode content streams and parse them for different clients, thus enabling you to develop applications for both a range of PC browsers and for the emerging pervasive devices. The content is in XML and an XML parser is used to transform it to output streams based on XSL style sheets.

This general capability is known as transcoding and is not limited to XML based technology. The appropriate design decision here is how much control over the content transforms you need in your application. You will want to consider when it is appropriate to use this dynamic content generation and when there are advantages to having servlets or JSP files specific to certain device types.

XML is also used as a means to specify the content of messages between servers, whether the two servers are within an enterprise or represent a business-to-business connection. The critical factor here is the agreement between parties on the message schema, which is specified as an XML DTD. An XML parser is used to extract specific content from the message stream. Your design will need to consider whether to use an event-based approach, for which the SAX API is appropriate, or to navigate the tree structure of the document using the DOM API.

For more detail on the use of XML in the server side of your Web applications, see *XML and Java: Developing Web Applications* by Maruyama, Hiroshi, Kent Tamura and Naohiko Uramoto.

#### 5.2.5 JDBC

The business logic in a Web application will access information in a database for a database-centric scenario. JDBC is a Java API for database-independent connectivity. It provides a straightforward way to map SQL types to Java types. With JDBC you can connect to your relational databases, and create and execute dynamic SQL statements in Java.

JDBC drivers are RDBMS specific, provided by the DBMS vendor, but implement the standard set of interfaces defined in the JDBC API. Given common schemas between two databases, an application can be switched between one and the other by changing the JDBC driver name and URL. A common practice is to place the JDBC driver name and URL information in a property or configuration file.

There are four types of JDBC drivers from which you can choose, based on the characteristics of your application:

- Type 1: JDBC-ODBC bridge drivers. This type of driver, packaged with the JDK, requires an ODBC driver and was introduced to enable database access for Java developers in the absence of any other type of driver.
- Type 2: Native API Partly Java drivers. This type of driver uses the client API of the DBMS and requires the binaries for the database client software. This type of driver offers performance advantages but introduces native calls from the JVM.
- Type 3: Net-protocol All Java drivers. A generic network protocol is used with this type of driver. Portability is a major advantage of this type of driver, but it has the limitation that it requires intermediate middle ware to convert the Net-protocol to the DBMS protocol.
- Type 4: Native-protocol All Java drivers. This type of driver is portable and uses the protocol of the DBMS. Type 3 and 4 drivers are well suited for applets that access a database server on an intranet, as they only require Java code to be downloaded.

An important technique used to enhance the scalability of Web applications is connection pooling, which may be provided by the application server. When application logic in a user session needs access to a database resource, rather than establishing and later dropping a new database connection, the code requests a connection from an established pool, returning it to the pool when no longer required.

The most recent level of the JDBC specification is 2.0, but many JDBC drivers you use will still implement 1.0.

---

### 5.3 Integration Server

The Application Framework defines the application integration component as that which allows disparate applications to communicate with each other. In this case, we are going to be using IBM's Component Broker as the integration server and will base this discussion on that product.

More information about developing IBM Component Broker applications can be found in *WebSphere Application Server Enterprise Edition Component Broker 3.0 First Steps*, SG24-2033.

Component Broker is an integrated package that allows you to develop, deploy, run, and manage a new generation of multi-tier, object-oriented programs. The Component Broker runtime environment supports applications written to Sun's Enterprise JavaBeans (EJB) technology and CORBA. We will be focusing our solutions on EJBs.

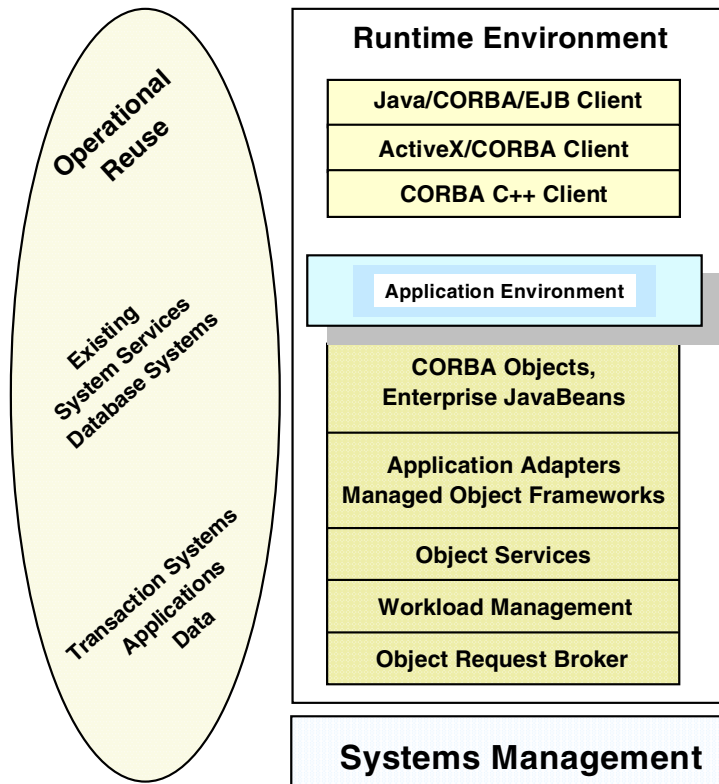


Figure 15. Integration server - IBM Component Broker

Component Broker connects to most of today's popular client environments. It supports clients written as Java applets, applications, and servlets, clients written in VisualAge for C++, and clients developed for Microsoft's ActiveX platform. While you certainly can continue to run "fat-client" topologies, Component Broker is ideally suited for the thinner client variety. Component Broker does not restrict you in any way from using your favorite GUI frameworks or tools.

You code the user interface and any local business logic as usual. In addition, the developers of your Component Broker server code will provide you with the constructs needed to connect to your server. What you get depends on your client platform. In any case, the appropriate source code usage bindings are available for all the above environments. Java code can be downloaded or cached locally on the client, as usual. In the other environments, you will need to install the necessary DLLs to build and run your application. All

clients connect to the application server through a CORBA-compliant Object Request Broker (ORB) and its Internet Inter-ORB (IIOP) protocol.

However, the core of Component Broker lies in its middle-tier application server. The server allows you to separate the traditionally extremely *volatile* client software environment from the code that implements your business logic. It also isolates that business logic from back-end technology specifics, such as databases or transaction monitors. This way, it provides you more opportunity to concentrate on the demands of your business, leaving the idiosyncrasies of the other tiers to the specialists in those areas.

The main components housed within the middle-tier application server are:

- **Application Adapters, Managed Objects, EJBs, and frameworks**

These provide the core infrastructure and scaffolding for your business logic to enable it to run in a mission-critical environment. They take care of such aspects as transactional behavior, connecting to back-end data stores and transaction monitors for persistence or legacy reuse, or interfacing to Object Management Group (OMG) object services.

Note that Component Broker offers a "container" that supports both CORBA objects and Enterprise JavaBeans. In spite of the considerable momentum behind Enterprise JavaBeans, CORBA remains a viable option in many cases. For example, it provides an alternative for customers who wish to use C++ as an implementation language, or who wish to use services such as Object-Oriented Query, Events, or Notifications. It provides a simpler means for communication from pure CORBA clients. Integrating ActiveX clients is straightforward. Though CORBA-based development may be more difficult than the Java platform, there are some industries where CORBA is mandated, largely because it is a truly open standard. It's important to realize that Component Broker tooling also eases the burden of CORBA development relative to other CORBA offerings available in the marketplace today. Finally, it's important to recognize that the frameworks and runtime infrastructure described here are relevant to both CORBA objects and to Enterprise JavaBeans, providing the same qualities of service in each of these cases. We will discuss more about Enterprise JavaBeans in a subsequent section.

From a CORBA perspective, Application Adapters and the Managed Object Framework administer what is called a Managed Object Assembly. Collectively, the elements of that assembly may be thought of as "the CORBA object". These elements include:

- **Business Objects**

These contain your actual business domain code (that is, your business logic).

- **Data Objects**

These isolate the Business Objects from the specifics of back-end databases and transaction monitors. The Data Objects themselves use the services of other, Framework-provided objects to implement the connection to back-end data stores. They can make use of extensive caching services to provide the kind of performance you would expect in today's online environments.

- **Managed Objects**

These *wrap* the above two. The Framework interfaces primarily with Managed Objects. When business logic operations are invoked on a Managed Object, the implementation is delegated to your Business Object. Component Broker-required methods are either implemented by the Managed Object itself or delegated to other parts of the runtime environment. Examples for the latter are setting up the transactional environment and implementing object services, such as the identity methods used by clients to compare object references.

As previously mentioned, Component Broker also provides an environment for Enterprise JavaBeans:

- **Enterprise beans (EJBs)**

Enterprise beans (referred to as EJBs) typically reside in the middle tier and provide the business logic implementation. Enterprise beans are distinguished from beans in that they are designed to be installed on a server, and accessed remotely by a client. The EJB Framework provides a standard for server-side components with transactional characteristics. EJBs are discussed further in 5.3.1, "Enterprise JavaBeans" on page 66.

There are a number of additional objects, such as:

- **Homes**

In *object speak*, homes are both factories and collections. They provide lifecycle functionality for objects of one particular type. They are also used to implement the query facility as well as iterability over the objects they contain.

- **Containers**

Containers shield client programmers from hairy details, such as memory management for your distributed objects. They take care of

activating and passivating objects according to policies you can specify through systems management facilities.

In general, containers provide you with an administrative boundary for implementing policies on your CORBA objects and EJBs. These policies are set up at install and are then used at runtime to set up the *quality of service* required in the case at hand.

- **Object Services**

Just to give you an overview, here is what is included. Note that Component Broker implementation may vary depending on the platform. See the individual product documentation for each platform to determine if the following services apply.

- **Naming Service**

The Naming Service is the mechanism for objects on the ORB to locate other objects by name. The Component Broker Naming Service is compliant with the OMG specification with some extensions to make working with the naming tree easier.

- **Event Service**

An Event Service allows objects to dynamically register or unregister their interest in specific events. The Component Broker Event Service, at the Release 1 level, implements OMG's transient, non-typed events.

- **Notification Service**

The Component Broker Notification Service is based on the Component Broker Event Service implementation with additional capabilities such as filtering and quality of service (QoS).

- **Security Service**

At the time of writing, a security implementation based on the Open Software Foundation/Distributed Computing Environment (OSF/DCE) is available. It supports authentication.

- **LifeCycle Service**

A LifeCycle Service provides operations for creating, copying, moving, and deleting objects in a distributed environment. Component Broker extends the OMG standard by giving you a measure of control over where you want to place your objects within the network of servers. This is accomplished through a concept called a *location*. Locations embody the notion of proximity, which may be interpreted in a geographical, structural, or even temporal sense.

– **Externalization Service**

The Externalization Service provides a mechanism by which objects are able to save and restore their state in a non-object form. The Externalization Service has two main purposes:

- Moving or copying the state of objects to different memory locations, between hosts, or simply to create a new object with the state of another already existing object.
- Moving the state of objects in and out of a persistent store.

– **Query Service**

The Query Service enables you to query for a set of objects that satisfy a set of conditions that you specify. You can access Component Broker server objects using keys. You can then follow links using references to get to related objects. Component Broker also allows you to execute queries using the query engine and query evaluators it provides. One way to do so is to specify your query using OO-SQL.

Component Broker cooperates with back-end RDBMS systems, in particular with DB2, to optimize query performance. In particular, it uses a query push-down technology to evaluate as much as possible on the back end. This way, indices present in the database can be used efficiently, and unnecessary data movement is avoided.

– **Transaction Service**

The Transaction Service enables programmers to implement transactions using standard object-oriented interfaces in a distributed environment. The OMG Object Transaction Service is implemented mostly under the covers by Application Adapter Frameworks that support this requirement. Component Broker can act as a transaction coordinator, managing two-phase commit operations with back-end database systems. It works with the XA protocols, as defined in the X/Open Distributed Transaction Processing specification (XA). LU6.2 conversations are also implemented for coordination of IMS and CICS.

– **Concurrency Service**

The Concurrency Service is a set of interfaces that allow an application to coordinate access by multiple transactions or threads to a shared resource. Component Broker's implementation of the OMG Concurrency Service does the job you would expect from it when you want to serialize access to resources. All of the well-known types of locks are supported.

– **Identity Service**

Component Broker derives an object identity from relative information that positions the object within its container, server, host, and ultimately domain. This information can be used within the Component Broker managed object framework to uniquely identify it from any other object in the distributed system.

- **Other services**

In addition to the object services, Component Broker also provides session services, allowing applications to define unit of work semantics for back-end systems that don't participate in a full two-phase commit protocol, and caching services to provide better performance and concurrency for applications.

- **Object Server runtime**

The Object Server manages the collaboration among most of the other parts of the Component Broker platform. With respect to user code, it coordinates system resources such as execution threads or context information related to transactions or security, for example.

- **Workload management**

This essential scalability feature brings load balancing into the picture. It allows clients to talk to a single, logical server image. In reality, operations that clients invoke may well be dispatched to different physical server processes. Introducing a new dimension in distributed object processing, Server Groups bring increased scalability and reliability to the picture.

- **Language interoperability**

Essentially, the Component Broker object model is based on CORBA 2.0+. Moreover, Component Broker includes a set of runtime libraries that supports inter-language calls between CORBA 2.0+ objects within the same process.

### 5.3.1 Enterprise JavaBeans

“Enterprise JavaBeans” is Sun's trademarked term for their EJB architecture (or “component model”). When writing to the EJB specification you are developing “enterprise beans” (or, if you prefer, “EJB beans”).

The EJB framework specifies clearly the responsibilities of the EJB developer and the EJB container provider. The intent is that the “plumbing” required to implement transactions or database access can be implemented by the EJB container. The EJB developer specifies the required transactional and security characteristics of an EJB in a deployment descriptor (this is sometimes referred to as declarative programming). In a separate step, the



EJB is then deployed to the EJB container provided by the application server vendor of your choice.

There are two types of Enterprise JavaBeans:

- Session
- Entity

A typical session bean has the following characteristics:

- Executes on behalf of a single client.
- Can be transactional.
- Can update data in an underlying database.
- Is relatively short lived.
- Is destroyed when the EJB server is stopped. The client has to establish a new session bean to continue computation.
- Does not represent persistent data that should be stored in a database.
- Provides a scalable runtime environment to execute a large number of session beans concurrently.

Component Broker's session beans may also be used to encapsulate messaging infrastructure (specifically, MQSeries).

A typical entity bean has the following characteristics:

- Represents data in a database.
- Can be transactional.
- Shared access from multiple users.
- Can be long lived (lives as long as the data in the database).
- Survives restarts of the EJB server. A restart is transparent to the client.
- Provides a scalable runtime environment for a large number of concurrently active entity objects.

With Component Broker, entity beans may be backed by procedural systems as well as relational databases.

Typically an entity bean is used for information that has to survive system restarts, while in session beans, the data is transient and does not survive when the client's browser is closed. For example, a shopping cart containing information that may be discarded uses a session bean, and an invoice issued after the purchase of the items is an entity bean.

An important design choice when implementing entity beans is whether to use Bean Managed Persistence (BMP), in which case you must code the JDBC logic, or Container Managed Persistence (CMP), where the database access logic is handled by the EJB container.

The business logic of a Web application often accesses data in a database. EJB entity beans are a convenient way to wrap the relational database layer in an object layer, hiding the complexity of database access. Because a single business task may involve accessing several tables in a database, modeling rows in those tables with entity beans makes it easier for your application logic to manipulate the data.

The latest EJB specification is 1.1. The most significant changes from EJB 1.0 are the use of XML-based deployment descriptors and the need for vendors to implement entity bean support to claim EJB compliance.

To learn more about Enterprise JavaBeans visit:

<http://www.javasoft.com/products/ejb/index.html>

## 5.3.2 Connectors

e-business connectors are gateway products that enable you to access enterprise and legacy applications and data from your Web application. Connector products provide Java interfaces for accessing database, data communications, messaging and distributed file system services.

IBM provides a significant set of e-business connectors with tool support, for CICS, Encina, IMS, MQSeries, SAP, Domino, DB2 and other relational databases. IBM is basing its tool support for most of these connectors on a Common Connector Framework (CCF). For resources on System/390, IBM is delivering native connectors based on CCF. The command bean model of the CCF allows you to code to the specific connector interface(s) of your choice while hiding the connector logic from the rest of the Web application.

WebSphere Enterprise Edition provides special connectors called *Application Adaptors*. Application Adaptors not only provide connectivity to various resource managers, they provide runtime qualities of service such as connection pooling, caching, recoverability, concurrency, security and two-phase distributed transaction support.

Connectors play an important role in topologies 5 and 6.

### 5.3.2.1 Common Connector Framework (CCF)

The task of connecting an application to a back-end data store is relatively standard and follows the same basic pattern whether you are considering the interactions between applications, servlets, EJBs, message queueing systems, relational databases, transactional systems or some other pieces of enterprise infrastructure. The typical approach to integration has been to hand-craft the code required to drive each component of a system.

IBM's Common Connector Framework (CCF) recognizes that most interactions follow a standard pattern and provides a standard Java-based infrastructure for integrating various system components together. The CCF is a framework made up of client and server interfaces. IBM provides a basic implementation of the CCF in VisualAge for Java and a more mature implementation in WebSphere Enterprise Edition in the Procedural Application Adapter.

The CCF simplifies enterprise connectivity development by providing:

- A common client programming model for connectors that greatly reduces the learning curve for an application developer
- A common infrastructure programming model for connectors
- A plug-in interface for higher-level tools, making them independent of a particular connector

VisualAge for Java provides the Enterprise Access Builder (EAB) tool, allowing you to create an EAB command that hides the complexities of enterprise connectivity. The programmer can easily use this EAB command as a bean that provides enterprise access functionality in the same manner for all the connectors.

Connectors currently exist for:

- CICS—both External Call Interface (ECI) and External Presentation Interface (EPI) modes
- IMS
- MQSeries
- Host On-Demand
- Encina DE-Light
- SAP R/3

CCF connectors use three main interfaces:

**ConnectionSpec:** A ConnectionSpec implementation uniquely identifies a connection and holds all connection-relevant attributes of a CCF Connector such as host name, port number, and time-out specifications. It may also encapsulate connector-specific connection data. For example, an MQSeries connector would require a channel identifier.

**InteractionSpec:** An InteractionSpec retains all interaction-relevant attributes of a CCF Connector such as a program name argument to a CCF Connector Communication when a particular interaction has to be carried out.

**Communication:** An implementation of Communication “drives” a particular interaction along via its execute method. Three arguments are passed to the execute method to carry out an interaction via a Communication. An instance of an InteractionSpec must be provided to identify the concrete interaction characteristics. The other arguments are an input and an output record that are used to carry the exchanged data.

In addition, it is necessary to specify input and/or output records to allow data to flow through the framework.

**input/output:** These represent the parameters to, and data returned from, the addressed Enterprise Information System (EIS). They may be a simple byte array but are usually a Java bean. Each bean is either based on the Java Record Library (a number of helper classes in the com.ibm.record, com.ibm.record.ctypes, and com.ibm.record.util packages intended to facilitate working with both fixed-length and variable-length records), or is defined specifically for a particular connector.

Applying the connector is relatively straightforward, even if performed by hand, but the preferred way of using the CCF connectors is through the integrated tools.

The CCF is easily applied within a feature-rich component environment such as that provided by WebSphere Application Server to support servlets or enterprise beans. However, it is not restricted to such environments and can be readily applied within applets or “fat” client applications. In these situations, it is necessary for the application to directly supply an implementation of the CCF runtime support infrastructure since this is normally provided by the supporting component environment.

---

## 5.4 Additional enterprise Java APIs

In developing a server-side application, you may also need to be familiar with the following enterprise Java class libraries:

- Java Naming and Directory Interface (JNDI). This package provides a common API to a directory service. Service provider implementations include those for LDAP directories, RMI and CORBA object registries. Sample uses of JNDI include:
  - Accessing a user profile from an LDAP directory
  - Locating and accessing an EJB Home
- Remote Method Invocation (RMI). RMI and RMI over IIOP are part of the EJB specification as the access method for clients accessing EJB

services. RMI can also be used to implement limited function Java servers.

- Java Message Service (JMS). The JMS API enables a Java programmer to access message-oriented middle ware such as MQSeries from the Java programming model. Such messaging middle-ware is a popular choice for accessing existing enterprise systems and is one of your options if you are implementing a solution based on application topology 2.
- Java Transaction API (JTA). This Java API for working with transaction services, is based on the XA standard. With the availability of EJB servers, you are less likely to use this API directly.

---

## 5.5 References and where to find more information

For more information on topics discussed in this chapter see:

- *WebSphere Studio and VisualAge for Java Servlet and JSP Programming*, SG24-5755-00
- *Servlet/JSP/EJB Design and Implementation Guide*, SG24-5754-00
- *WebSphere Application Server Enterprise Edition Component Broker 3.0 First Steps*, SG24-2033
- *Component Broker Advanced Programming Guide*, SC09-4443
- *Connecting Enterprise to the Web by Example: CCF Connectors and database connection using WebSphere Advanced Edition*, SG24-5514
- *Building e-business Solutions*, SC09-4432
- Flanagan, David, *JavaScript: The Definitive Guide*, Third Edition, O'Reilly & Associates, Inc., 1998
- Maruyama, Hiroshi, Kent Tamura and Naohiko Uramoto, *XML and Java: Developing Web Applications*, Addison-Wesley 1999
- Flanagan, David, Jim Farley, William Crawford and Kris Magnusson, *Java Enterprise in a Nutshell*, O'Reilly & Associates, Inc., 1999
- For information on the IBM Application Framework for e-business:  
<http://www.ibm.com/software/ebusiness/>
- For information about the ECMAScript language specification:  
<http://www.ecma.ch/stand/ECMA-262.htm>
- To learn more about Java technology:  
<http://www.javasoft.com/products>



---

## Chapter 6. Application design guidelines

e-business application design presents some unique challenges compared to traditional application design and development. The majority of these challenges are related to the fact that traditional applications were primarily used by a defined set of internal users, whereas e-business applications are used by a broad set of internal and external users such as employees, customers, and partners. Web applications must be developed to meet the varied needs of these end users. The following list provides key issues to consider when designing e-business applications:

- The look and feel of the site needs to be constantly enhanced to leverage emerging technologies in order to attract and retain site users.
- New features have to be constantly added to the site to meet customer demands.
- Such changes and enhancements will have to be delivered at record speed to avoid losing customers to the competition.
- e-business applications in essence represent the corporate brand online. Developers have to work closely with the marketing department to ensure the digital brand effectively represents the company image. Such intra-group interactions usually present content management challenges.
- It is hard to predict the runtime load of e-business applications. Based on the marketing of the site, the load can increase dramatically over time. If the load increases, the design must allow such applications to be deployed in various high-volume configurations. Runtime configurations are discussed in Chapter 3, “Choosing the runtime topology” on page 23 and Chapter 4, “Product mapping” on page 33. It is important to be able to move Web applications between these runtime configurations without making significant changes to the code.
- Security requirements are significantly higher for e-business applications compared to traditional applications. In order to execute traditional applications from the Web a special set of security-related software may be needed to access private networks.
- The emergence of the personal digital assistant (PDA) market and broad-band Internet market will require the same information to be presented in various user interface formats. PDAs will require a light-weight presentation style to accommodate the low network band width. Broad-band users on the other hand will demand a highly interactive rich graphical user interface.

In order to meet these challenges it is critical to design Web applications to be flexible. This chapter helps you understand some of these design challenges and presents various design pattern solutions that promote loosely coupled design to provide a maximum degree of flexibility in a Web application.

**Problem Domain:** Our problem domain is determined by the lessons we want to teach:

- We want to demonstrate how to use the Model-View-Controller (MVC) pattern to construct Web-based applications.
- We want to demonstrate how application topology 5 will support access to the enterprises' legacy data stores and back-end applications from a single location.
- We want to demonstrate how application topology 6 will support the integration of these data stores and back-end applications within a single application.

So we will imagine a problem domain that lets us address these problems. Our imaginary scenario is one where a consumer banking company has purchased a mortgage company.

As a first step to integrating their operations, the management of the consumer bank has decided to build a topology 5 solution that will provide access to both the bank's database and the mortgage company's back-end applications from a single Web site. The solution to this will be to use WebSphere Enterprise Edition's Component Broker as the integration node, acting as a router to direct data and service requests from the company's new Web site to these legacy elements. Component Broker was chosen for the integrator node because of its performance, robustness, manageability, and support for integration with relational databases as well as nonrelational back-end systems and legacy applications.

The *Consumer Banking* application will make customer account information available through a Web browser. Customers will also be able to transfer money between the checking and saving accounts the bank holds for them. Of course, they will not be able to deposit or withdraw cash.

This application will use JSPs (for the "view"), servlets (as the "controller" piece), and enterprise beans (for the "model"). Customers may have one or more checking and savings accounts with the bank. They may or may not have a mortgage account. Building the presentation parts of this application with JSPs will let us create clean, legible displays for any of these situations. At the back-end of the application, we will be dealing with other people's



money and need the robustness and immunity from software and hardware failures that enterprise beans provide.

We will explore the design of this application in some detail in this section. We should note that, in practice, we have derived this application from the WebSphere Application Server 3.0 Family Samples application, using the portion of the application relevant to our topologies (enterprise beans on Component Broker) and ignoring the elements of that application meant to demonstrate Java Business Objects (JBO) and the WebSphere Advanced Editions TxSeries connectors. The original Family Sample application is available at:

<http://www.ibm.com/software/webservers/samples/>

We recommend downloading these samples and becoming familiar with them. The discussion and sample code described throughout this chapter are based on the Family Sample application.

A mortgage company with which the bank recently merged had implemented its back-end applications using MQSeries. The company's tellers would send mortgage payment messages to an MQSeries "Mortgage Payment" queue. At the end of each business day, the mortgage payment application would empty these messages from the queue and update each customer's account.

We will build a new *Mortgage Payment Service* application that will be Web-based and will place mortgage payment messages on the existing payment queues. This will allow management to do away with all of the fat clients that had run at each teller's window to record mortgage payments, simplifying the management of the mortgage company's software and making it easier to distribute new functionality.

In a second phase, the bank's management plans to provide the customer with a Web-based access to both their bank and mortgage accounts. Customers would be able to view the balances of all their accounts from the Web and would be able to make mortgage payments from either their checking or saving accounts.

To implement this second phase, they will need a topology 6 solution. In this topology, we will again use Component Broker for the integration node, but this time it will be decomposing requests and composing responses for the user. Component Broker can perform the services and integration needed to bring together these two financial applications into a single new application in the middle tier called *Consumer Bank Plus*. While other application server's can certainly generate calls to these back-end elements, only Component Broker can provide two-phase commit coordination across the various

resource managers and legacy technologies required by this financial institution.

## 6.1 Application elements

The design guidelines outlined here focus primarily on User-to-Business Web applications. Before exploring these guidelines it is important to understand the overall structure of these types of Web applications. Chapter 3, “Choosing the runtime topology” on page 23 presents the overall topology of such e-business applications and explains the responsibilities of various nodes in the topology. Chapter 5, “Technology options” on page 49 presents various technology options available for implementing a User-to-Business Web application and recommends the use of server-centric Java-based technologies such as servlets, JSPs, JavaBeans, and EJBs for such implementations. Figure 16 identifies the key elements of such Web applications and explains the responsibilities of these elements.

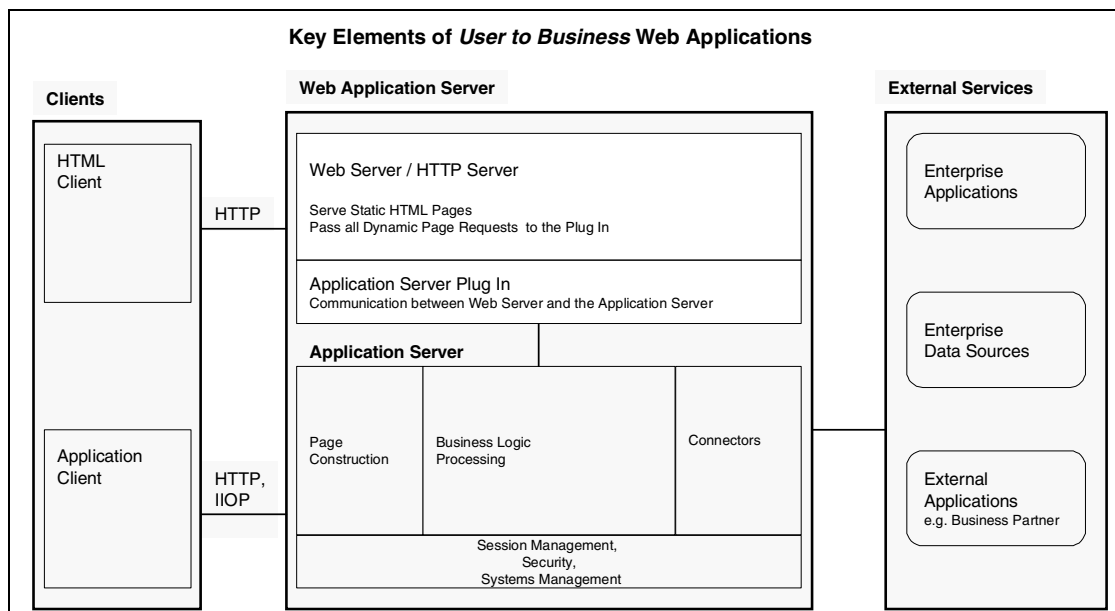


Figure 16. Key elements of User-to-Business Web applications

*Clients* are responsible for accepting and validating the user input, communicating the user input to the Web application server, and presenting the results received from the Web application server to the user. Clients may use HTTP, IIOP, TCP/IP, or other Internet standard protocols to communicate

with the Web application server. These clients can be broadly classified into the following categories:

- HTML clients use HTTP protocol to communicate with the Web application server. These clients display HTML and DHTML Web pages. In addition, they are capable of processing client-side JavaScript for enhancing navigation to perform simple input validation and to handle simple errors. Furthermore, the majority of HTML clients can display small Java applets to enhance the GUI.
- Application clients are primarily large Java applets or Java applications. These clients provide rich graphical user interfaces compared to HTML clients. They may communicate with the Web application server over a number of protocols including HTTP, IIOP, MQ, etc. Application clients communicate with the Web application server primarily to receive data rather than pre-formatted HTML pages. These clients use the data received to format and render the user interface. All of the user interface processing is performed on the client side. In addition, under this model, some parts of the business logic can also be processed on the client side.

WebSphere Application Server Advanced Edition supports both of these client models. However, HTML clients provide the following benefits compared to application clients:

- The majority of the presentation logic and all of the business logic will reside on the server. Hence it is easy to make the necessary changes to support a broad range of client devices including personal digital assistants (PDAs), WebTV, etc.
- The client part of the application is lightweight and downloads quickly.
- It is easier to secure, scale, and maintain presentation and business logic that reside on the server.
- Client applications that use Java applets and Java applications require a particular version of Java to be supported by all clients, thus limiting the universal accessibility of the applications.
- Client applications that use Java applets and Java applications are downloaded to a user's local machine and can be de-compiled to view the business logic. This poses security concerns. Hence critical business logic should not be directly coded in these downloadable applications.

WebSphere Enterprise Edition (Component Broker) supports only application clients, which are most commonly servlets, but can be Java applets or applications. Also, because Component Broker is a CORBA server, it supports CORBA-compliant client applications written in Java, C++, or Visual Basic/ActiveX (COM). In the sample application discussed in this book, we

use WebSphere Advanced Edition for the servlet and JSP environment, and the servlets act as application clients to enterprise beans running in Component Broker.

For these reasons, the majority of Web applications being designed today are zero-maintenance HTML clients. The guidelines outlined in this chapter focus primarily on designing and developing HTML client Web applications. Further details on these client models can be found at:

<http://www.ibm.com/developer/features/framework/framework.html>

*Web application servers* are the focal point of the Web application topology. Their responsibilities include receiving requests from the clients, selecting and executing the appropriate business logic based on these requests, coordinating with *External Services* to retrieve data and execute external applications, and finally formulating the response and dispatching it back to the client. To meet these requirements, Web application servers provide a range of dynamic page construction, business logic processing, data access, external application integration, session management, load balancing, and fail-over services. At a high level, one can identify the following sub-components in the majority of Web application servers including WebSphere Application Server:

- Web Server/HTTP Server - are responsible for receiving and responding to HTTP requests. They are capable of serving static HTML pages without help from other sub-components of the Web application server. However, they pass all dynamic page requests to the application server plug-in. In the case of WebSphere Application Server, the HTTP server is usually configured to pass all requests for servlet and JSP execution to the WebSphere plug-in.
- Application server plug-in provides the connection between the HTTP server and the page construction services of the Web application server.
- Page Construction services - WebSphere Application Server supports Java-based technologies such as Java servlets and JavaServer Pages (JSP) files for dynamic page construction.
- Business logic services provide a robust environment for processing business logic independent of the user interface client types. WebSphere Application Server supports components based on technologies such as JavaBeans and Enterprise JavaBeans (EJBs) for programming business logic.
- Connectors are components that support the communication between the application server and the external services such as databases, legacy applications, and business partner applications. WebSphere Application

Server Advanced Edition provides a number of connectors including JDBC drivers, JNDI class libraries, CICS connectors, MQ connectors, and IMS connectors.

- Further details of these connectors, class libraries, and related tools can be found at:

<http://www.ibm.com/developer/features/framework/framework.html>.

- Session management services are necessary because inherently HTTP is a stateless protocol. In order to address the issues of a stateless protocol, a number of HTTP session management techniques have been developed. WebSphere Application Server supports a number of these techniques and provides a simple HttpSession API to handle the session information. This API is based on the standard Java Servlet API.
- Security services include authentication, authorization, data integrity, privacy (encryption) and non-repudiation services. WebSphere Application Server provides these services by supporting industry-standard protocols such as SSL, LDAP, etc.
- System management services provide a robust runtime environment for the application hosted in the Web application server. These services allow load balancing, fail-over support, remote monitoring, etc. WebSphere Application Server supports these features.
- External Services include enterprise data sources (for example, relational databases such as DB2 and Oracle), existing or new enterprise applications (for example, CICS/IMS transactions, ERP systems, financial systems, etc.), and business partner systems.

Component Broker is an application server included in WebSphere Enterprise Edition which provides services for business logic and coordination with *External Services*, similar to how Web application servers are described above. However, Component Broker is not a “Web” application server, in that it does not service HTTP requests; it is a pure CORBA component and EJB server, which means it services only IIOP and RMI/IIOP requests.

---

## 6.2 Understanding supporting technologies

Before presenting some of the design pattern solutions, it is important to establish a common understanding of the key technologies involved. Chapter 5, “Technology options” on page 49 discusses various technology options available for Web application development and recommends the use of HTML, JavaScripts, Java servlets, JSPs, JavaBeans, and EJBs for implementing user-to-business applications. This section introduces you to

server-side Java technologies, namely servlets, JSPs, JavaBeans, and EJBs. However in order to understand the design guidelines you are expected to be familiar with HTML and client-side JavaScripts. This is because dynamically generated Web pages contain client side JavaScripts that perform page navigation and simple edits. A good introduction to HTML can be found at <http://www.w3.org/MarkUp/> and to JavaScripts found at <http://www.ecma.ch/stand/ECMA-262.htm>.

### 6.2.1 Java servlets

Servlets are protocol and platform-independent server-side Java components. They implement a simple request and response framework for communication between the client and the server. The Java servlet API is a set of Java classes that define a standard interface between Web clients and the Web application server. The API is composed of the following two packages:

- `javax.servlet`
- `javax.servlet.http`

The `javax.servlet` package implements the generic protocol-independent servlets. The `javax.servlet.http` package extends this generic functionality to include specific support for the HTTP protocol. In this section we explore the key classes and methods of the `javax.servlet.http` package. The subsequent sections introduce other classes and methods of this package if such services are exploited by the topic under consideration.

**HttpServlet** is an abstract class that provides methods for handling various HTTP requests. Typically, all servlets that respond to HTTP traffic would extend this class and override some of the methods such as `doGet()` to handle GET requests and `doPost()` to handle POST requests. WebSphere Application Server provides ways to register these servlets with the Web server. Upon receiving an HTTP request the Web server determines if it needs to be handled by a servlet and if so, it passes such requests to WebSphere. WebSphere in turn calls the `service()` method which then calls the HTTP-specific method based on the type of HTTP request. The following are some of the key methods provided by the `HttpServlet` class:

- `init()` - The purpose of this method is to perform necessary servlet initialization. It is guaranteed to be the first method to be called on any servlet instance. The servlet implementer may choose to override this method to perform custom servlet initialization.
- `service(HttpServletRequest req, HttpServletResponse resp)` - The Web application server invokes the `servlet.service()` method upon receiving an

HTTP request targeted towards that servlet. This method in turn invokes the appropriate HTTP-specific method based on the type of request. `HttpServletRequest` is an input parameter and contains the HTTP protocol-specific header information. `HttpServletResponse` is an output parameter containing an HTTP protocol-specific header and returns data to the client. Further details on these classes can be found below.

- `doGet(HttpServletRequest req, HttpServletResponse resp)` - The `service()` method invokes the `doGet()` method if the HTTP request type is GET. The servlet implementer overrides the `doGet()` method if the servlet is intended to handle GET requests. HTTP GET requests are expected to be safe and read-only. They are suitable for queries.
- `doPost(HttpServletRequest req, HttpServletResponse resp)` - The `service()` method invokes the `doPost()` method if the HTTP request type is POST. The servlet implementer would override the `doPost()` method if the servlet is intended to handle POST requests. HTTP POST requests are not expected to be either safe or read-only. They are suitable for requests that result in updates to stored data.
- `destroy()` - This method is called just before unloading a servlet. It is overridden only if there is a need to perform some cleanup operations such as closing connections or files before unloading a servlet.
- `getServletContext()` - This method returns a `ServletContext` object that contains information about the environment in which the servlet is executing. This method is particularly useful in dispatching control from a servlet to a JSP. Developers are never expected to override this method.

**HttpServletRequest** represents a communication channel from the client and is passed as an input parameter into the `HttpServlet.service()` method, which in turn passes it to the appropriate HTTP-specific methods such as `doGet()` and `doPost()`. The servlet can invoke this object's methods to get information about the client environment, the server environment, and any HTTP protocol-specified header information that is received from the client. The following are some of the key methods provided by this interface:

- `getParameter(java.lang.String name)` - returns a string containing the value of the specified parameter.
- `getParameterValues(java.lang.String name)`, on the other hand, returns the parameter value as an array of strings. These methods can be used to retrieve the HTML FORM information set by GET and POST methods.
- `getSession(boolean create)` - returns the current valid `HttpSession` associated with this request. If `create` is true and a current valid `HttpSession` does not exist then a new session is created.

- `setAttribute(java.lang.String key, java.lang.Object o)` - is used to store an attribute in the request context. Attributes are stored as name-value pairs in a hash table and can be retrieved by the `getAttribute()` method. This method may be used to pass data between various servlets and JSPs. It is important to note that the attributes are reset between requests.

**HttpServletResponse** represents a communication channel back to the client and is passed as an output parameter into the `HttpServlet.service()` method. It provides a number of set methods that allow servlets to manipulate HTTP protocol-specific header information and to set the response data to be returned to the client. The following are some of the key methods provided by this interface:

- `getWriter()` - returns a print writer object. The print writer object is an output stream to which servlets write dynamically generated text such as HTML and XML. Upon completion, all the text that is written to the print writer will be sent to the client, for example sent to the browser to be displayed.
- `setContentType(java.lang.string type)` - sets the MIME content type to be sent back to the client, for example "text/html". The content type must be set before obtaining the print writer or writing to the output stream.
- `sendRedirect(java.lang.string type)` - sends a temporary redirect response to the client using the specified redirect location URL. The URL must be absolute (for example, `https://hostname/path/file.html`). Relative URLs are not permitted here.

**HttpSession** represents an association between an HTTP client and an HTTP server. By design, HTTP is a stateless protocol. Over the years, a number of approaches have been developed to maintain application sessions between HTTP requests. `HttpSessions` are used to maintain application state and user identity across several page requests from the same user. The following are some of the key methods provided by this interface:

- `putValue(java.lang.String name, java.lang.Object value)` - can be used to store application-specific state data as a name-value pair. The application server is responsible for storing this information in a hash table and allows servlets to access this data across HTTP requests.
- `getValue(java.lang.String name)` - is used to retrieve the application-specific value from the hash table by referencing the name-value pair by its name.
- `removeValue(java.lang.String name)` - is used to remove the name-value pair from the hash table. In order to manage scarce server resources, it is important to remove state information that is no longer required.



**ServletContext** object contains information about the environment in which the servlet is executing. A servlet can obtain its context by calling the `getServletContext()` method. The following is the key method provided by this interface:

- `getRequestDispatcher(java.lang.String urlpath)` - takes the URL path of resources such as other servlets and JSPs as input and returns a `RequestDispatcher` object that implements a wrapper around a server resource. `RequestDispatcher` is responsible for locating such resources and also forwarding any requests made by the servlet to the appropriate resource. As shown below we use them to forward requests from the servlets to JSPs.

```
RequestDispatcher rd;  
rd = getServletContext().getRequestDispatcher("Sample.JSP");  
rd.forward(req, res);
```

In this example, the servlet retrieves the `ServletContext`. Using this context, it obtains the `RequestDispatcher` to a JSP. Finally it forwards the request to the JSP by calling the `forward()` method on the `RequestDispatcher`. This invokes the `service()` method on the target JSP.

WebSphere Application Server V3 supports the Java servlets API 2.1. Further details of this API can be found at:

<http://java.sun.com/products/servlet/2.1>.

*Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755 provides further details and examples of servlet programming.

### 6.2.2 JavaServer Pages (JSP) files

JSP files provide a simple yet powerful mechanism for inserting dynamic content into Web pages. Dynamic content might be:

- Optional paragraphs generated based on the visitor's location, buying habits or other attributes, or
- Optional tables containing one or more answers to a visitor's request for local stores, or products in their price range.

A JSP file looks like an HTML page with Java statements inserted. In fact, it is a script that allows a JSP processor to create a Java servlet from the embedded Java statements, which will emit the HTML statements of the script. On the first reference to the JSP page, the processor will construct the

servlet and then for that and all subsequent references, invoke the servlet to construct the HTML page.

The JSP processor does its work by generating a generic `HttpServlet` object and creating a custom `service()` method for the object from the script. A statement is generated in the `service()` method to write each HTML statement of the script into the `HttpServletResponse` object. The Java statements are embedded directly into the `service()` method.

The `HttpServlet` has a context which the Java statements must be able to access. The statements must certainly be able to place text into the `HttpServletResponse` and where the text they output will depend on features of the visitor or the visitor's request, these statements must be able to access the `HttpServletRequest` or the session. The author of the JSP file must know the names of these objects in the generic `HttpServlet` generated by the JSP processor.

The HTML text of the JSP page will also need to include text strings that will only be available in the context of the executing servlet. The author of the JSP file can reference these using special tags in the HTML text.

The JSP specification documents the features that are available to the author of a JSP page and that all complying JSP processors must support. For information on all the features of JSPs, refer to:

<http://www.java.sun.com/products/jsp>

IBM has extended this definition for the WebSphere Application Server. For a description of these extensions refer to:

<http://www.ibm.com/software/webservers/appserv/library.html>

In this document we will use only a few of the features described in the JSP specification 1.0. We will use these variable names from the generated servlet context in our Java statements:

*Table 2. JSP implicit objects*

| Object Name | Object Type                                    | Description  |
|-------------|--|--|
| request     | <code>javax.servlet.HttpServletRequest</code>  | Represents the HTTP request received from the client.  |
| response    | <code>javax.servlet.HttpServletResponse</code> | Represents the HTTP response to be sent to the client. |

| Object Name | Object Type                 | Description  |
|-------------|-----------------------------|--|
| session     | avax.servlet.HttpSession    | Represents the session object created for the requesting client (if any).      |
| out         | javax.servlet.jsp.JspWriter | Represents the output stream to be sent to the client as part of the response. |

We will use these tags to include dynamic text in our HTML statements:

For the purposes of this chapter we focus on the following JSP Specification 1.0 tags:

- JSP scriptlet

Syntax: `<% code %>`.

Scriptlets can be used to insert any code fragment in the specified scripting language into a JSP. By default, the scripting language is assumed to be Java.

- JSP expression

Syntax: `<%= expression %>`.

The expression here stands for any valid operation that can be executed at runtime. The output from such an operation is converted into a string and is emitted into the output stream *out*.

- `jsp:useBean`

Syntax: `<jsp: useBean id="beanInstanceName" scope="page|request|session|application" typespec/>`

The server uses `id` and `scope` to look for the bean. If it exists it is accessed. If not, it is created. The `typespec` is used to specify the bean type.

The `jsp:useBean` tag is used to declare a JavaBean you would like to use within a JSP.

- `jsp:getProperty`

Syntax: `<jsp:getProperty name="beanName" property="propertyName" />`

The `<jsp:getProperty>` converts the value of the specified property into a string and inserts this string into the implicit *out* object. `getProperty` on a bean is usually called after declaring the bean instance using the `jsp:useBean` tag.

- `jsp:setProperty`

**Syntax:** `<jsp:setProperty name="beanName" property="propertyName" value="propertyValue"/>`

`<jsp:setProperty>` is used to set the bean property value.

WebSphere Application Server V3 supports JSP Specification 1.0 and JSP Draft Specification 0.91. You have to choose a particular specification level for a JSP page. We recommend using JSP Specification 1.0. Further details on these specifications can be found at:

<http://www.java.sun.com/products/jsp/>

In addition, WebSphere supports WebSphere-specific tags that allow you to connect to databases and perform repeats, etc. Details on the WebSphere extension to the JSP specification can be found in the WebSphere Library at:

<http://www.ibm.com/software/webservers/appserv/library.html>

We will discuss the JSP files built for this application and include fragments of those pages in this document. While both HTML and Java are fairly simple languages, they have different semantics, different conventions, and operate in very different environments. This can make preparing JSP pages by hand difficult.

WebSphere Studio provides easy-to-use tools for creating JSP files. It allows us to “glue” text, images, buttons and dynamic elements together on the screen, and will generate the correct JSP page.

### 6.2.3 JavaBeans

The JavaBeans architecture defines reusable software components that can be manipulated visually by builder tools. Code written to this architecture, called Java beans, or beans, are specialized Java classes that support the following features: properties, methods, events, introspection, and persistence. Basically, properties are attributes that have set and get methods. Methods are simple Java methods that can be called from other components. Events define a framework for one component to notify the other when something noteworthy happens. The JavaBean specification defines a set of conventions for defining properties, methods, and events. Using this convention, builder tools can analyze the bean to allow for visual manipulation. In addition, beans should implement the persistence (serialization) mechanism allowing the customized JavaBean state to be stored and retrieved when necessary. Beans can be either visual or non-visual; however, they should all allow manipulation by visual builder

tools. This is usually done by using Java introspection techniques. Further details on the JavaBean Specification can be found at:

<http://java.sun.com/beans/index.htm>

#### **6.2.4 Enterprise JavaBeans**

The Enterprise JavaBeans (EJB) architecture extends the reusable software component model of JavaBeans to multi-tier, distributed, transactional computing environment. Code written to the Enterprise JavaBeans architecture will be referred to in this redbook as EJBs or enterprise beans. Our applications will use enterprise beans for their business logic components.

The Managed Object Framework (MOFW), introduced in Chapter 5, “Technology options” on page 49, is the underlying framework implemented by Component Broker to provide support for the CORBA and EJB architectures. IBM contributed heavily to the EJB specification, and in fact, much of the EJB architecture was based on the MOFW.

Further details on the EJB Specification can be found at:

<http://java.sun.com/products/ejb/index.html>

#### **6.2.5 What’s next?**

The previous sections have established a common set of terminology and a common understanding of the key technologies involved in a User-to-Business Web application. Based on this understanding, the subsequent sections of this chapter will:

- Introduce a design challenge.
- Discuss the recommended design pattern solution.
- Explain the motivation for using the recommended design pattern solution.
- Document the implications of the chosen approach on the runtime topology.
- Provide an example that applies the recommended design technique using class diagrams, interaction diagrams, and sample code.
- Document the advantages and disadvantages of the proposed solution.

## 6.3 Application Structure

A User-to-Business Web application can be viewed as a set of interactions between the browser and the Web application server. The interaction begins with an initial request by the browser for the welcome page of the application. This is usually done by the user typing in the welcome page URL on the browser. All subsequent interactions are initiated by the user either by clicking a button or a link. This causes a request to be sent to the Web application server. The Web application server processes the request and dynamically generates a result page and sends it back to the client along with a set of buttons and links for the next request.

A closer examination of these interactions reveals a common set of processing requirements that need to be considered on the server side. These interactions can be easily mapped to the classical Model-View-Controller (MVC) design pattern as shown below. As outlined in the book *Design Patterns: Elements of Reusable Object-Oriented Software* the relationship between the MVC triad classes are composed of Observer, Composite, and Strategy design patterns. For further details on these detailed design patterns, please refer to the above-mentioned design patterns book.

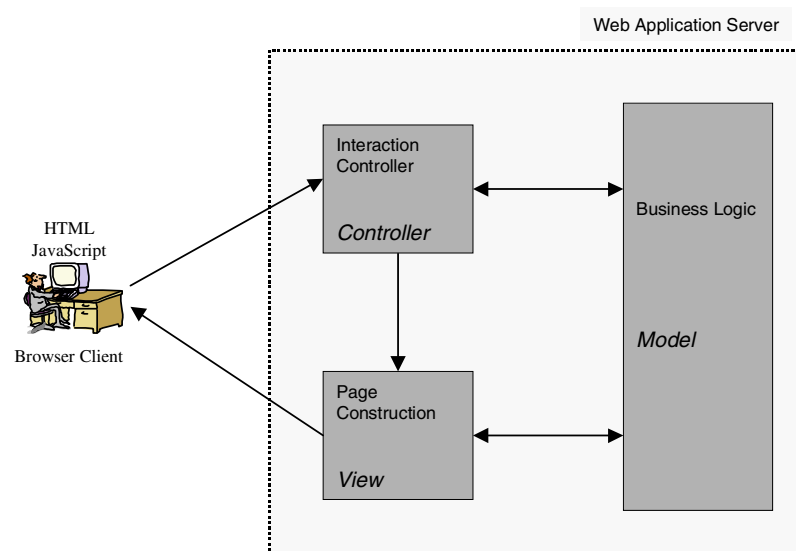


Figure 17. The structure of user-to-business Web applications

*Model* represents the application object that implements the application data and business logic. The *view* is responsible for formatting the application

results and dynamic page construction. The *controller* is responsible for receiving the client request, invoking the appropriate business logic, and based on the results, selecting the appropriate view to be presented to the user. A number of different types of skills and tools are required to implement various parts of a Web application. For example the skills and tools required to design an HTML page are vastly different from the skills and tools required to design and develop the business logic part of the application. In order to effectively leverage these scarce resources and to promote reuse we recommend structuring Web applications to follow the Model-View-Controller design pattern.

### 6.3.1 Model-View-Controller (MVC) design pattern

Over the years, a number of GUI-based client/server applications have been designed using the MVC design pattern. This powerful and well-tested design pattern can be extended to support the user-to-business Web applications as shown by Figure 17 on page 88. Throughout this chapter, *model* is often referred to as business logic, *view* is referred to as page constructor or display page, *controller* is referred to as interaction controller. This section further outlines the responsibilities of each of these components and discusses what technologies could be used to implement the same.

**Interaction controller (*controller*)** - The easiest way to think about the responsibility of the interaction controller is that it is the piece of code that ties protocol independent business logic to a Web application. This means that the interaction controller's primary responsibility is mapping HTTP protocol-specific input into the input required by the protocol-independent business logic (that might be used by several different types of applications), scripting the elements of business logic together and then delegating to a page construction component that will create the response page to be returned to the client. Here's a list of typical functions performed by the interaction controller:

- Validate the request and session parameters used by the interaction.
- Verify that the client has the necessary privileges to access the requested business task.
- Transaction demarcation.
- Invoke business logic components to perform the required tasks. This includes mapping the request and session parameters to the business logic component's input properties, using the output of the components to control logic flow and correctly chain the business logic.
- Call the appropriate page construction component based on the output of one or more of the business logic commands.

Interaction controllers can be implemented using either Java servlets or JSPs. It is important to note that interaction controller code is primarily Java code and Java code is easy to develop and maintain using Java Integrated Development Environments (IDE) such as VisualAge for Java. Since servlets are also Java classes, it is possible to leverage such IDEs to write, compile, and maintain servlets. JSPs on the other hand provide a simple script-like environment. Even though JSPs were primarily developed for dynamic page construction, they can be used to code interaction controller logic. Due to their simplicity, JSPs have a broad appeal to script programmers. However tools available today for JSPs are primarily targeted toward dynamic page construction.

Finally, it is up to the project team to decide whether to use JSPs, servlets or both for coding interaction controller logic. What is much more important is to recognize the need for the separation between interaction controller logic and page construction logic. Such a separation is necessary under the following conditions:

- One display page needs to be reusable because it can be called by multiple interaction controllers. For example an error page may be called by more than one interaction controller. Under such a scenario if we were to combine the error page construction logic and the interaction controller logic, then the error page logic would need to be duplicated in several places throughout the application.

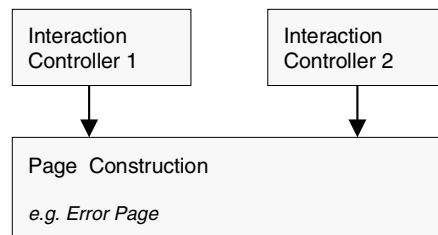


Figure 18. One display page component being called by multiple interaction controllers

- The interaction controller is required to do page selection. There are several reasons for this, such as the need to include different display pages depending on runtime results, national language support, different client browser types, different customer types, etc. For example, the following figure shows an interaction controller calling either the admin or normal page constructor, based on the used type.



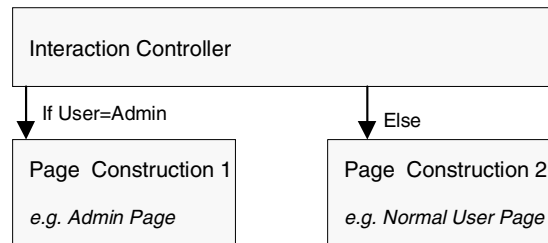


Figure 19. One interaction controller calling multiple page constructors

- If there is a need to use different tools and skills for coding interaction controllers and display pages then it is good to separate the two to simplify the development process. Failing to do so could result in multiple people having to write different parts of the same file thus complicating the version control and code management process.

**Note**

Hence it is recommended that servlets should be used in most cases to implement interaction controllers. However, for simple applications where there is no conditional or transactional logic involved, it is possible to combine the interaction controller and page construction logic into one component. Under such conditions, a JSP would be the best choice.

Another common design issue to consider is the relationship between interactions and interaction controllers. The following is an overview of the options:

- One interaction to one interaction controller: For every unique interaction, there is a unique interaction controller. For example, login is handled by loginServlet, and logoff is handled by logoffServlet.
- One interaction group to one interaction controller: A group of related Web interactions are all handled by the same interaction controller. The interaction controller for the group is passed a parameter to differentiate which interaction within the group is being performed. For example, login and logoff both could be handled by authenticateServlet which can get a parameter called action type that could be either set to login or logoff.
- All interactions to one interaction controller: This approach extends the interaction group to all interactions and builds a monolithic interaction controller.

We recommend the first option in most cases. Occasionally it might be appropriate to choose the second option and implement one interaction controller that supports a group of related interactions.

**Page constructor (view)** - The page constructor is responsible for generating the HTML page that will be returned to the client; it is the view component of the application. Like interaction controllers, WebSphere allows display pages to be implemented as either Java servlets or JavaServer Pages. JSPs allow template pages to be developed directly in HTML, with scripting logic inserted for dynamic elements of the page. Hence, JSP is the best choice for implementing page construction components. In addition, there are number of visual tools such as WebSphere Page Designer that can be used to develop dynamic display pages using JSP technology.

In many cases, the interaction controller will pass the dynamic data as JavaBeans to the display page for formatting. In other cases, the display page will invoke business logic directly to obtain dynamic data. It makes sense to have the interaction controller pass the data when it has already obtained it and when the data is an essential component of the contract between the interaction controller and the display page. In other cases, the data needed for display is not an essential part of the interaction and can be obtained independently by inserting calls to business logic directly in the display page. However, such direct access to business logic from the page construction component increases the complexity of the display page, since the page designer must know the details of the business logic methods. For this reason, care must be taken to minimize such direct access to business logic from the display pages.

Once the page constructor has obtained the dynamic data, either from the interaction controller or via its own logic, it will typically format the data. This can be done in two ways. The simplest mechanism is to format the data using simple scripting inside the page constructor. An alternative is to develop reusable formatting components called Formatter beans that will take a data set and return formatted HTML. 6.5, “Application output formatting” on page 98 further elaborates this concept.

**Business logic (*model*)** - The business logic part of a Web application is the piece of code ultimately responsible for satisfying client requests. As a result, business logic must address a wide range of potential requirements, which include ensuring transactional integrity of application components, maintaining and quickly accessing application data, supporting the coordination of business workflow processes, and integrating new application components with existing applications. To address these requirements, WebSphere supports business logic written in and using the full facilities of

the Java runtime including support for Java servlets, Java beans, enterprise beans, JDBC, CORBA, LDAP, and connectors to MQ, CICS, IMS and other enterprise services.

While a discussion on how business logic should be developed is beyond the scope of this chapter, it is valuable to consider the interface between the Web parts of the interaction (interaction controllers and the display pages) and the business logic. We recommend that the business logic be wrapped with beans or enterprise beans. Such a separation of business logic from the Web-specific interaction controller and display page logic isolates the business logic from the details of Web programming, increasing the reusability of the business logic in both Web and non-Web applications. Further details on how such a wrapping can be done can be found in 6.6, “Application business logic granularity” on page 99.

Convinced of the soundness of the MVC pattern, the Consumer Banking application discussed throughout this book is built to this pattern, and serves as an example MVC application.

---

## 6.4 Application component contracts

When dealing with issues related to passing the data between the various model-view-controller components, there are guidelines for defining the contract between the interaction controller, business logic, and display pages.

In the case where the interaction controller expects a single field back upon executing the business logic, the single result field can be returned as a simple string. If the interaction controller expects more than one field as a result of executing the business logic, then we recommend returning a data bean that wrappers all the result fields. Since this data bean represents the result of executing business logic, we call it a *Result bean*. A *Result bean* effectively defines the contract between a particular piece of business logic and a particular interaction controller.

Now let us turn our attention towards the contract between the interaction controller and the page constructor. For simple interactions the *Result bean* itself can be used to define such a contract. However, in some cases there might be a reason to introduce a *View bean*. A *View bean* is responsible for combining the result data and the display-specific attributes.

Such *Result beans* and *View beans* collect multiple result fields, minimize the number of calls, and simplify the contract between various components. The following figure demonstrates the relationship between MVC components, *Result beans* and *View beans*. It is important to note that in simple Web

interactions, both a Result bean and a View bean could be implemented by the same Java bean.

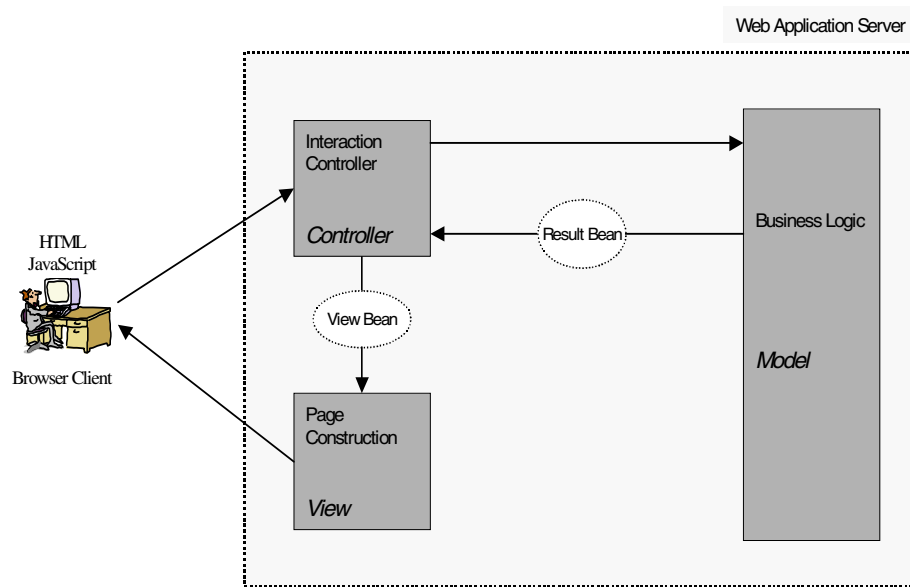


Figure 20. Result bean and View bean design pattern

#### 6.4.1 Result beans and View beans design pattern

**Result bean** - defines the contract between the interaction controller and the business logic. It wrappers all the return values the interaction controller expects to receive upon executing a piece of business logic. Hence it provides an easy communication mechanism between the developers of these two components. Result beans are usually implemented as simple Java beans. Since Java beans by definition are serializable they can be passed by value between EJB-based business logic implementations. Whenever possible, Result bean properties should be implemented as read-only properties. A constructor could be used to initialize these properties during instantiation. This prevents the interaction controllers and page constructors from inadvertently updating the data.

It is important to note that a Result bean can be reused by multiple business logic methods or objects. For example, based on some condition, the controller may decide to call two different business logic methods. If it is appropriate, both methods could use the same Result bean to return the results. The reverse can also be true. Multiple controllers may call the same business logic that returns the same Result bean to all controllers.

**View bean** - defines the contract between the controller and the view. It lists all the attributes the JSP can display. The main benefit of defining such a View bean is to make it easy for the JSP page designer to get all the required data in one place. The display page often contains the data from the following sources:

- Result bean properties (returned by the business logic)
- HTTP request data (including attributes, parameters, cookies, URL string)
- Session state
- Servlet context

The controller is responsible for instantiating the View bean and initializing all the properties of the bean. View beans can be designed to be responsible for view-specific transformations. For example, a View bean can be responsible for converting the monetary values into the user-preferred currency. Such a View bean can have two properties: the monetary value in a base currency and the currency display type. The controller can initialize both of these properties. The View bean can use this information to call a reusable currency conversion library and get the monetary value in the appropriate format.

Usually, View beans are tightly coupled with a JSP since its primary purpose is to provide all the properties the JSP designer would need in one place. However, under special circumstances one can reuse the View beans by inheritance.

For example, let us assume that there are two types of users of a banking system, namely customers and customer service representatives (CSR). A particular screen in the system allows customers to see the transaction history. The same screen allows the CSRs to see transaction history and transaction details, allowing CSRs to answer any customer questions about the transaction such as where it was conducted, which bank representative was involved, etc. Since the CSR screen has all the data that the customer is able to see and has additional information, the `CSRTransactionHistoryViewBean` could be inherited from `CustomerTransactionHistoryViewBean`. This would ensure that the customers

and CSRs see the same basic information, with the CSRs seeing additional details. The following figures depict this scenario:

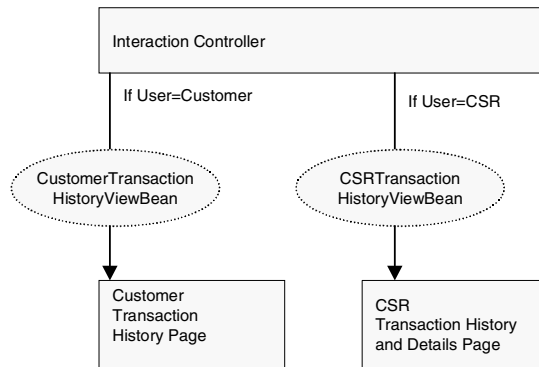


Figure 21. One interaction controller calling multiple display pages

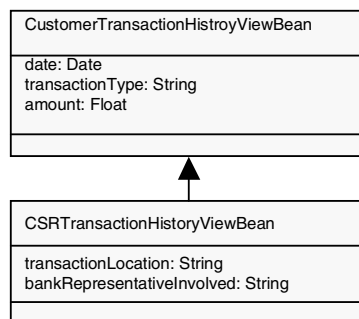


Figure 22. Class diagram - View bean inheritance example

#### 6.4.2 Advantages and disadvantages of Result beans and View beans

In summary, Result beans define the contract between controllers and model and View beans define the contract between controllers and view. Both Result and View beans are implemented using simple Java beans. In some cases it may make sense to use the same bean as both the Result and the View bean.

##### Advantages of Result beans

- Clearly define what the controller expects back from the business logic.
- Once a Result bean is clearly defined, the developers of the controller and the model can develop their components independently. This simplifies

and optimizes the development process and allows for parallel development.

- Since Result beans can be serialized, they can be sent to remote servers or received from remote servers such as EJB-based distributed applications. In addition, they can be stored on a file for persistence purposes.
- The Result bean data structure can be reused by multiple business logic objects and interaction controller objects.

### **Advantages of View beans**

- Clearly define all the fields a view (JSP) can display.
- Once a View bean is clearly defined the developers of the controller and the view can develop their components independently. This simplifies and optimizes the development process and allows for parallel development.
- The view (JSP) designer can get all the dynamic data from one View bean and use `<jsp:useBean>` and `<jsp:getProperty>` tags to insert these values. This allows the JSP designer to concentrate on the look and feel of the page rather than worrying about gathering data from various sources (for example, sessions, cookies, Result beans, etc.) and coding complex view-specific Java code. View beans effectively hide these complexities from the display page designer.
- Complete separation of the view-specific logic from the business logic, for example, currency conversion based on the user preference.
- Using inheritance as outlined above, you can promote View bean reuse and ensure similar information is received by all users, for example, CSRs and customers.
- View beans can be used with tools such as WebSphere Page Designer, which allows a developer to insert JavaBean properties into JSPs.

### **Disadvantages of View beans**

On the flip side, it is important to recognize the following disadvantages of introducing Result beans and View beans:

- View beans are tightly coupled with display pages and interaction controllers. This implies that any changes to the dynamic content of the display page will require a change to the interaction controller. We depend on the interaction controller to gather all the required information in one View bean.

- For small applications the introduction of Result beans and View beans could result in too many individual pieces of code and increase the complexity of application management.
- The number of artifacts to be coded, managed, and maintained may be greatly increased.

---

## 6.5 Application output formatting

The View bean concept described above tries to minimize the need for inserting Java code directly inside a display page. For the most part, by using View beans the display page designer can use `<jsp:useBean>` and `<jsp:setProperty>` tags to insert dynamic content. In doing so one can use tools such a WebSphere Page Designer that allows you to insert JavaBean properties into JSPs.

In order to insert complex tables or drop-down lists, complex conditional loops may be needed. JSP API 1.0 does not define repeat tags that allow for looping through an indexed JavaBean property. In order to overcome this limitation, consider the following options:

- Implement the complex table or drop-down list generation code in Java and wrapper it inside the bean.
- Insert complex conditional loop Java code inside the JSP.
- Use WebSphere specific `<tsx:repeat>` tag.

Among these options we recommend the first one since it hides the complexity from the JSP designer and promotes reuse. This allows non-programmers to design the display page easily.

### 6.5.1 Formatter beans

It is possible for a number of display pages to share common methods that dynamically generate table bodies, drop-down lists, and radio button lists based on the values in an indexed field. In addition, it is possible to build reusable beans that convert values from one unit to the other. We call such reusable beans *Formatter beans*.

### 6.5.2 Advantages and disadvantages of Formatter beans

To summarize, a Formatter bean is a bean that wrappers reusable HTML formatting logic inside a method.



### Advantages of Formatter beans

- Eliminate the need for inserting complex scripting logic inside a JSP.
- Promote reusability of the formatting logic.
- Hide the complexity of scripting logic from view designers.
- Promote the ability to drop common information on multiple pages such as displaying the current weather information and current stock price on all the pages of the Web application.

### Disadvantages of Formatter beans

- Some of the display page functionality that is best expressed in a JSP is now moved into Java code.

---

## 6.6 Application business logic granularity

This section primarily focuses on the design issues related to the model part of the MVC design pattern. From our discussions earlier, we recognize that the business logic part of a Web application is the piece of code ultimately responsible for satisfying client requests. Hence, it must address a wide range of potential requirements including transactional integrity, application data access, workflow support, and integration of new and legacy applications. In order to achieve this, business logic components may use various protocols including JDBC, JNDI, IIOP, RMI, messaging, etc. to communicate with enterprise applications, enterprise data sources, and external applications. The model is not only responsible for implementing the business logic, but also for hiding the details of the data and application access protocols. This can be achieved by wrapping the model with a Java bean.

In implementing such business logic components one can choose between the following levels of granularity:

- One business logic bean per task:

This approach implements one business logic bean per business task, offering the lowest granularity. We call these beans *Command beans*. 6.6.1, “Command beans” on page 100 explains this concept.

- One business logic bean that groups a related set of tasks:

This approach implements multiple methods, each representing a piece of the business logic, and groups related methods under one bean. In doing so, usually all methods related to a *State* in the underlying business process are wrapped together by a bean. Hence these beans are called

*State beans.* Since only related methods are grouped together, this style has a medium level of granularity. 6.6.3, “An alternative approach” on page 102 explains this concept further.

- One business logic bean that groups all the tasks:

This approach implements multiple methods, each representing a business task and groups them all under one bean. This implementation has the highest level of granularity among all the options listed here. Such a monolithic approach complicates system development and any maintenance processes. Hence this style should be avoided.

Among these, we recommend the first two approaches. Both Command and State bean approaches have advantages over the last approach.

### 6.6.1 Command beans

Command beans can be defined as Java beans that provide a standard way to invoke business logic. The following are the key characteristics of Command beans:

- Each Command bean corresponds to a single business logic task, such as a query or an update task.
- All Command beans inherit from a single command interface. In essence they implement the command interface.
- The inherited Command bean defines business domain specific properties such as account numbers.
- Command execution results are stored as properties of a Command bean, therefore, Command beans also act as Result beans.
- Commands have a simple, uniform usage pattern:
  - a. Create an instance of the Command bean.
  - b. Initialize the bean by setting its properties.
  - c. Cause the bean to execute its action by calling one of its methods.
  - d. Access the results of command execution by inspecting its properties.
  - e. Discard the command object.
- Commands can be serialized.

The following figure shows how such a Command bean would interact with the other components of the Web application.

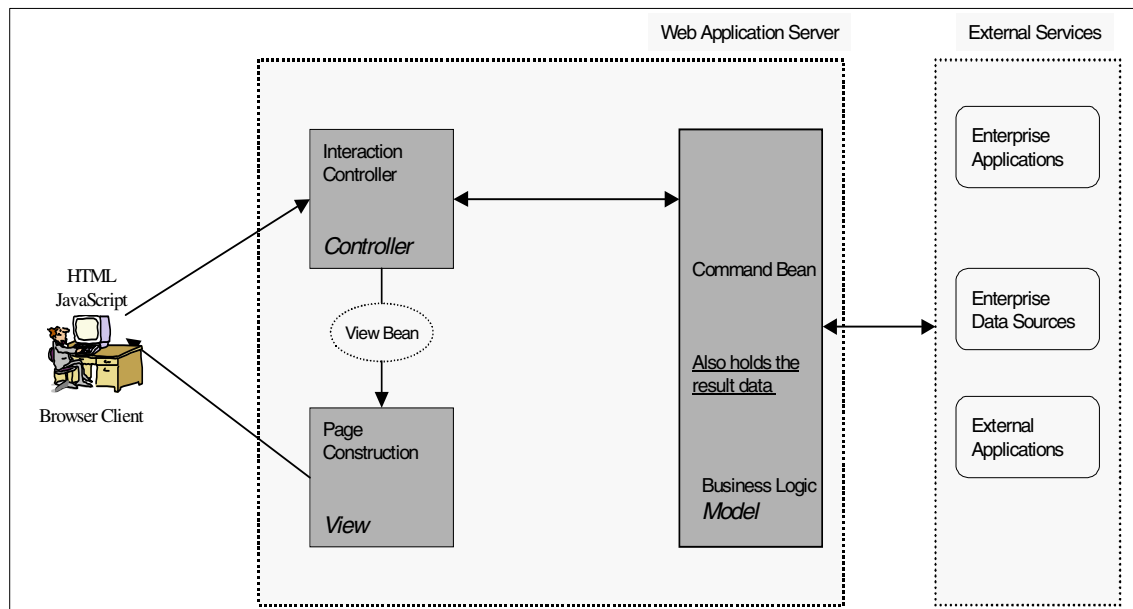


Figure 23. Command beans

#### No need for Result beans

It is important to note that the Command bean contains the command execution results; hence there is no need to introduce another Result bean. In essence, the Command bean encapsulates both the business logic and result data.

### 6.6.2 Advantages and disadvantages of Command beans

In summary, Command beans are stylized beans that provide a common interface for executing business logic. In addition to implementing the business logic, they hide the complexity of data and back-end application access.

#### Advantages of Command beans

- Provide a common interface for executing business logic.
- This common interface makes it easier to implement application development tools. Further releases of WebSphere development tools are expected to provide significant support based on this framework.

- This common interface can be further extended to support cross-tier communication and remote execution using the Command Manager pattern. WebSphere V3.5 is expected to implement this framework and provide the necessary API for cross-tier communication and remote execution of commands.
- This structure also makes it easy to implement automatic data caching. WebSphere V4.0 is expected to further extend the Command Manager pattern to implement caching.
- It hides the data and application access protocol details from the interaction controllers and page construction components. This abstraction layer allows for changing the data access mechanism without altering other components in the MVC design pattern.

Command beans are ideal where the Web application server acts primarily as a hub that receives and forwards requests to back-end applications where the complex business logic is executed. If there is a need to implement complex business logic locally on the Web application server node, then one could consider the approach described below.

### 6.6.3 An alternative approach

This approach provides the medium level of business logic granularity and implements one business logic bean that groups a related set of tasks. State beans are an example of this approach. State beans can be defined as business domain specific objects that group related business tasks associated with a state in the business process. It is critical to note that the State bean methods are responsible for accessing data from external services and should hide any protocol-specific details. Such an implementation will make it easy to change the back-end interface without affecting interaction controllers and display pages. State beans differ from the Command beans primarily in their granularity.

We use a discount brokerage application example to discuss how State beans can be designed. Let's say that this discount brokerage application allows users to enter a trade request, execute the trade, and view the status of the trade. Such an application may have the following states:

- Trade Entry: This allows the user to get a stock quote and research a stock, enter the details of the trade including stock ticker symbol, quantity, buy or sell, limit price, etc., update details of the trade, confirm the trade or cancel the trade.
- Trade Execution: Here there may be a need for the internal department to intervene if the required funds are not available. If things flow smoothly,

the trade gets executed. This state may also allow the customer to see the status of the trade and to cancel the trade if the trade has not been executed yet.

- Trade Completed: Here the customer will be able to see the trade execution price, quantity purchased or sold, etc. This state can also allow the user to query a history of completed trades.

Under the State bean approach one would have three objects, each representing the state in the business process namely TradeEntryBean, TradeExecutionBean, and CompletedTradesBean. Each of these beans would have multiple methods, each representing a business task. For example, TradeExecutionBean may have the following methods: cancelTrade(), getAllOpenTrades() and getTradeStatus(). Each of these methods can take required parameters as input and return a Result bean.

Designing such a business logic model is very specific to a problem domain. Such a design follows the traditional object-oriented analysis and design process. We will not go into detail on object-oriented principals or design processes. Object-oriented designers are already familiar with modeling such business logic objects. Currently the Pattern Development Kit does not provide examples for implementing State beans.

The primary disadvantage of this approach is the lack of a standard interface for invoking business logic. Hence it does not lend itself to automatic caching techniques that can be implemented by extending the Command bean approach. However, the State bean approach is suitable for applications that require complex modeling on the Web application server node.

---

## 6.7 The Consumer Banking application

In this section, we discuss the high-level specification of the Consumer Banking application using Unified Modeling Language (UML), and how this example application fits into the MVC pattern. We will also walk through the implementation of a model-view-controller scenario within the application to further detail how development took place. This will help us understand how the MQSeries-based Mortgage Payment Service will be integrated into the application and how the enterprise beans dictated by this design will be deployed into the Component Broker environment.

Recall that our application will allow the customer to view account balances, transfer funds between accounts, and review account histories. Recall also that we have already decided that we will implement our displays with a set of

JavaServer Pages, and that we will implement our business logic with enterprise beans.

We can use UML to document our application's functional and component architecture, and to manage the construction of test cases. The WebSphere Enterprise Edition development toolkit provides a bridge between Rational Rose and Object Builder (a piece of the toolkit used for developing CORBA components and deploying enterprise beans). Using this bridge, a Rose model representing your application can be passed to Object Builder to create skeleton CORBA components for the construction of application components. IBM also provides a similar bridge for importing Rose models into VisualAge for Java for developing enterprise beans.

### **6.7.1 UML and the Consumer Banking Application**

The first step in preparing the UML definition of our application is to identify the actors, either end users such as customers or another program or computer, as will often be the case for distributed applications maintained between cooperating companies. Each will have certain attributes and require selected capabilities, which we will make explicit in our definition.

We might define tellers, branch managers, or monthly statement writers. Tellers might have unique teller identifiers which, given a customer identifier, would allow them to view a customer's account information from restricted teller consoles. Bank managers would have unique identifiers which, with a customer identifier, would allow them to arbitrarily modify that customer's accounts. A monthly statement writer would be a program which, when executed from the bank's secure server, would review the month's transaction history for every customer account without a customer identifier.

In fact, we will define only one actor, a customer. Each customer will have an identifier unique from every other customer and the identifier will name accounts and transaction histories belonging to that customer. Each customer will be able to review the balance of their accounts, transfer money between accounts, and review the history of transactions on their accounts. A customer's accounts must be made safe from tampering by other customers.

Having named our actor, we will define the actor's use case. A use case describes one of the command and reply exchanges between the actor and the application. We have already noted that our customer will make a secure connection to the bank, will get a list of account balances, will move money from one account to another, and will sever the connection. Our customer's use cases are represented in the following figure.

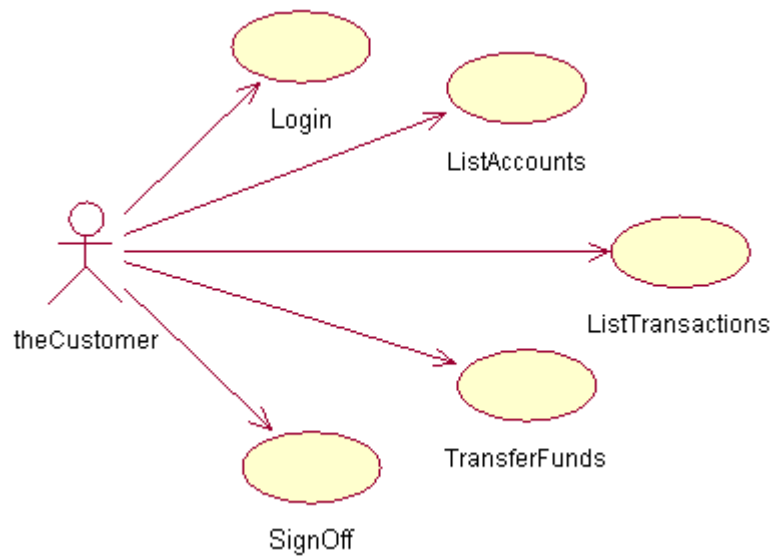


Figure 24. Use case

Our use cases are:

**Login** - the customer will be presented with a login screen and challenged to provide a valid user ID. If an invalid key is presented, an error message will be displayed; otherwise the customer will be invited to select any of the other use cases.

**List Accounts** - the authenticated customer will be presented with a list of its accounts and account balances and will be invited to select any of the ListTransactions, TransferFunds, and SignOff use cases.

**ListTransactions** - the authenticated customer will be presented with a list of transactions committed against its accounts and will be invited to select any of the ListAccount, TransferFunds, and SignOff use cases.

**Transfer Funds** - the authenticated customer will be invited to transfer money from its checking to savings account or savings to checking account, the source account will be tested for sufficient funds and the customer will be invited to select any of the ListAccounts, ListTransactions, TransferFunds and SignOff use cases.

**Signoff** - the customer's session will be ended, the customer will be unauthenticated, and the customer will not be invited to select another use case.

We will continue the design process with a state diagram for our application.

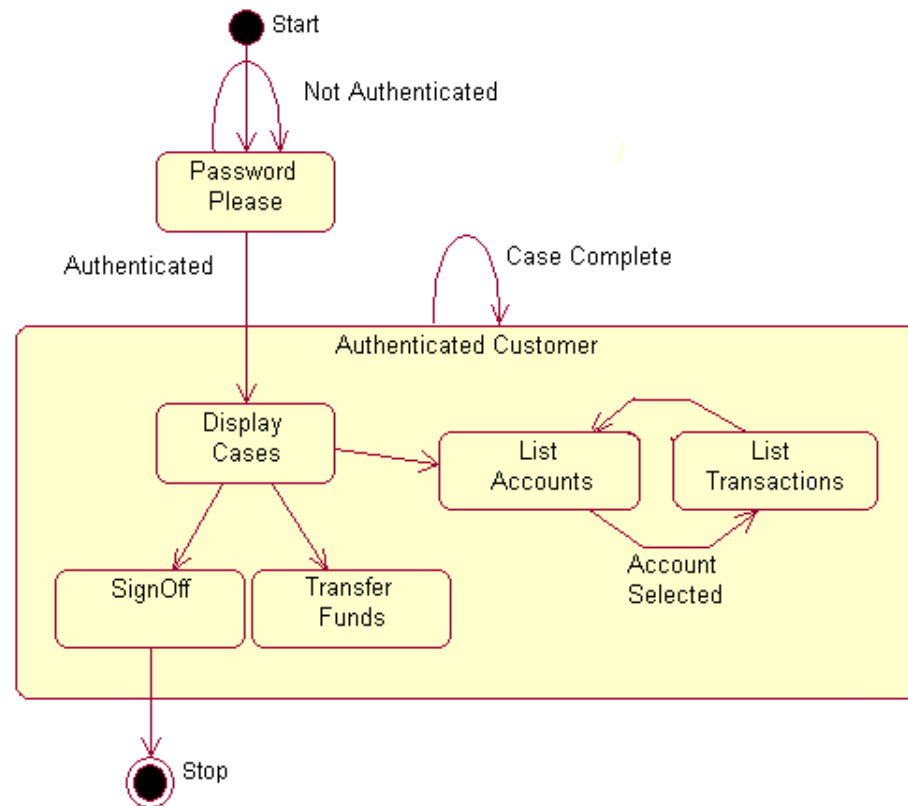


Figure 25. Consumer Banking application state diagram

This state diagram illustrates the dependency of the account browsing and management cases on the login case, which authenticates the user, and that we will keep accepting commands to execute new cases until the user signs off. Other than that, it is not particularly interesting. Let's look at another one.



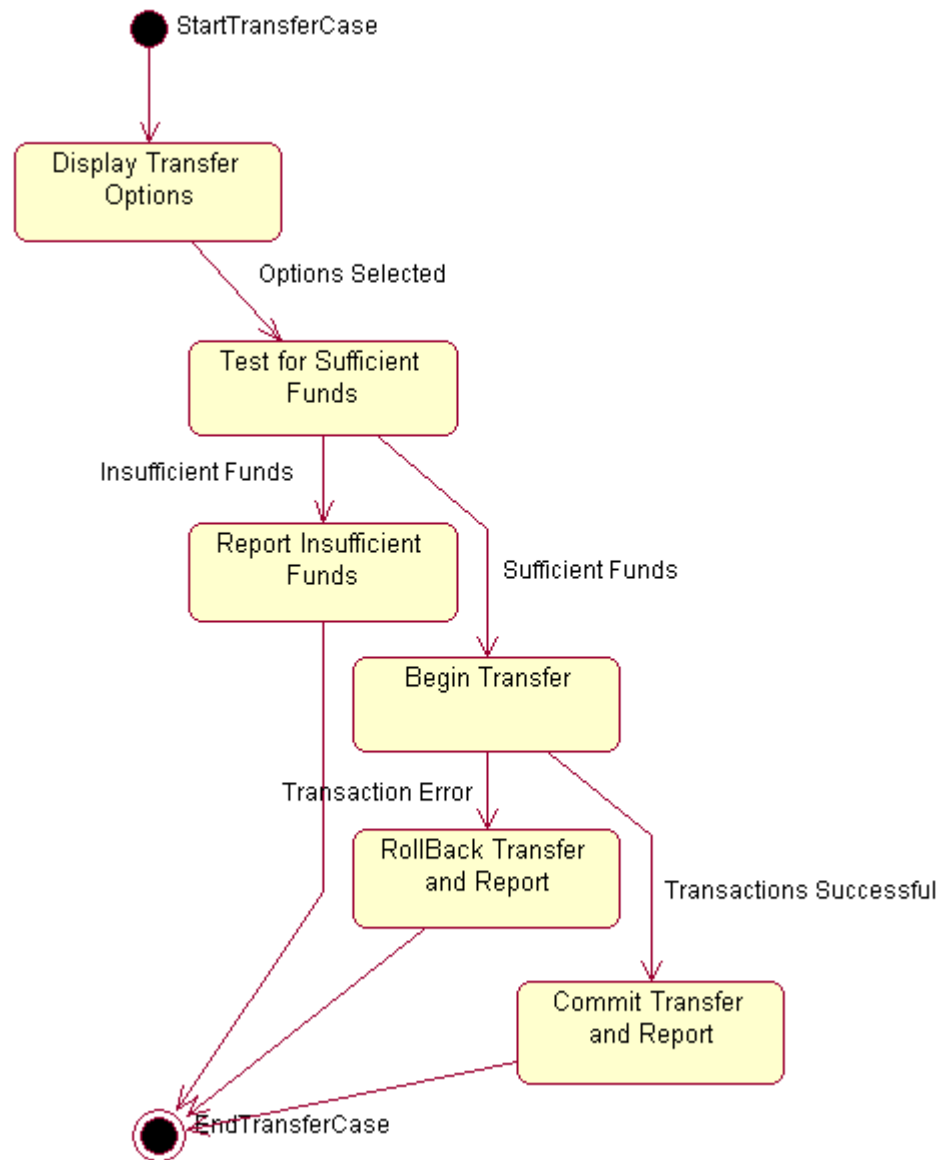


Figure 26. Transfer Funds use case

This diagram illustrates the Transfer Funds use case. It begins with a request for the Transfer Funds option view and ends with the display of either a Results page or an error page.

The first state, “Display Transfer Options” displays the transfer options available to the customer and collects the customer’s choices. These choices are the source account, the destination account, and the amount of the transfer.

When we have the required parameters we will transit to the second state, “Test for Sufficient Funds”. The arc (the arrow), from “Display Transfer Options” labeled “Options Selected” represents this transition.

There are two alternative successor states to this “Test...” state. If we do not have sufficient funds, we will transition to the “Report Insufficient Funds” state. Notice that there is only one arc out of this state and that arrow is not labeled with a condition. When we reach this state, we will transition to the end state immediately after completing the report.

If funds are sufficient at the “Test...” state, we will transition to the “Begin Transfer” state. We will handle the transfer as a transaction, a set of database reads and writes that must all successfully complete together or must all be completely canceled together or “rolled back”. Once the transfer has been started, the transaction handler will be able to tell us if the entire transaction can be committed or must be rolled back. This will lead us to one of the two successor states to “Begin Transfer”. We will either transition by way of the “Transaction Successful” arc to “Commit Transfer and Report” state, or we will transition to the “Rollback Transfer and Report” state.

In either case, these states each have only one arc out and these arcs are not labeled with a condition. As soon as their work is done we will transition to the End Transfer.

The states of these diagrams do not immediately correspond to the components we will need to build for our application. We cannot yet see the model, view, or controller parts in them.

We will use a UML activity diagram to continue refining our design.

An activity diagram documents the owners of elements of the state diagram. While the state diagram documents all the choices an application might make and all the ways it might change its data, the activity diagram documents which of many cooperating machines, programs, or human operators will be responsible for which transitions and transformations, and when information about these changes must be communicated between the cooperating parties.

An activity diagram for our Transfer Funds function is illustrated below.

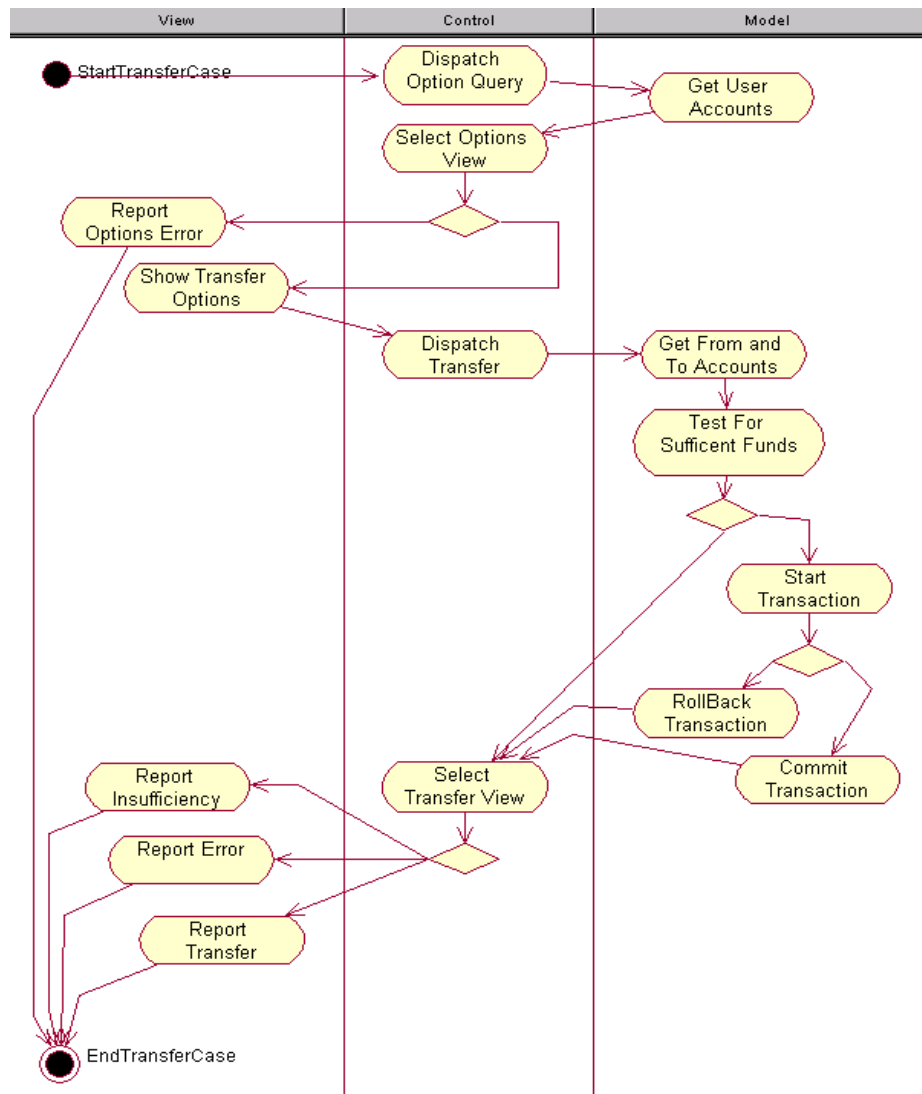


Figure 27. Transfer Funds activity diagram

A casual inspection will make clear how this activity diagram has evolved from our state diagram. The most interesting new feature of the diagram is the pair of straight lines dividing this diagram into three "swim lanes". These lanes are used to sort out the work of cooperating threads, machines, people, or software components. In our case, we use them to sort out the functions that will be performed in the three components of our design pattern.

We argued that the view managed the presentation logic of the application and collected the user input, the mouse clicks and key strokes. We have not detailed the “Report...” or “Show...” screens in our use case, a step that would happen very early in the design of an actual application, but we can imagine they will consist of displaying some amount of dynamic data in an HTML page and the construction of HTTP requests from simple HTML forms. We will look at this in more detail shortly. It is sufficient here to note that these make up a very shallow presentation layer.

When we defined the controller we argued that it would map user inputs to operations in the model and would manage the generation of appropriate presentations for the results. We have placed several “Dispatch...” and “Select...” activities in the control lane. Our “Dispatch...” activities will be responsible for mapping the Transfer Funds HTTP request from a “Select Banking Task” activity that precedes this diagram and the HTTP request constructed by the “Show Transfer Options” activity into appropriate commands for the model. An inspection of the diagram makes the role of our “Select...” activities translating the result of an operation in the model into the appropriate presentations obvious.

And we have argued that the model is responsible for the business logic of the application: of preserving its entity relationships, enforcing the rules governing changes to the business model, and persisting in those changes. Our “Get User Accounts” activity is an example of the model implementing an entity relationship, returning all of the accounts known to belong to a particular user. The “Test for Sufficient Funds” activity enforces a rule that no more money can be transferred from an account than is known to be in it. Finally, our “Start Transaction” and subsequent activities ensure that the changes to the business model are made correctly to the model or not at all.

We are almost prepared to present our application as a set of classes, a fairly low level in the design. At this point we will be able to discuss the statements that implement this model. We will need to take two steps first; we must make two implementation choices and record those decisions in an analysis diagram.

We must decide if we have a single, monolithic code object implementing each of our controller and model layers, or if each layer is implemented by a collection of control or model elements.

Note that our controller contains two pairs of matched “Dispatch...” and “Select...” activities. While there are many ways these activities could have been drawn, this diagram captures the useful fact that when a particular command is dispatched, someone must be around to collect that particular

result. We will leap to the conclusion that these matched pairs should be represented by a single code object. Now that there are two of these pairs, should they both be implemented by a single code object? There are activity diagrams for each of our other user cases. Should each of the pairs we might find in those diagrams be implemented by the same code object?

We will implement each matched pair as a servlet. There are deployment, maintenance, performance and management consequences to this choice and we will address some of these issues later. Here we note that the work of each matched pair is essentially context free. While in the larger scheme we must not allow a customer to transfer funds until they have been authenticated, the work of transferring funds and displaying the results has nothing to do with the work of listing accounts. There is no advantage to the programmer in keeping these lines of code in a single code object and as we are programmers, this is the criteria that makes the choice for us. We will code several small servlets rather than a single large one.

We will make the opposite decision for the model. Here again we see two bound sets of activity in our model: collecting the customer's accounts for the transfer options view, and the activities that actually perform the transfer. Likewise, we can assume there are similar fragments in each of the other use case activity diagrams. Each of these could be implemented as an independent code object but we will implement them as methods of a single Business Tasks class instead. Our decision is once again driven by programming convenience. Recall that we have already decided to implement the model with enterprise beans. We will implement container-managed Entity beans to provide access to our persistent data object, securing them with transactional semantics and building support for their data associations. The business logic will be implemented by the methods of the Session bean. In our application, the Entity beans are the difficult part of the model. Isolating each of the comparatively simple business rules into its own Session bean, all of which would be dependent on the same Entity beans, offers no advantage and obliges us to do much more management.

We can capture these conclusions in a UML analysis diagram. The analysis diagram is a variation on a class diagram. It differs from the class diagram in that it documents the role our various classes will play in the application. Figure 28 shows our applications analysis diagram.

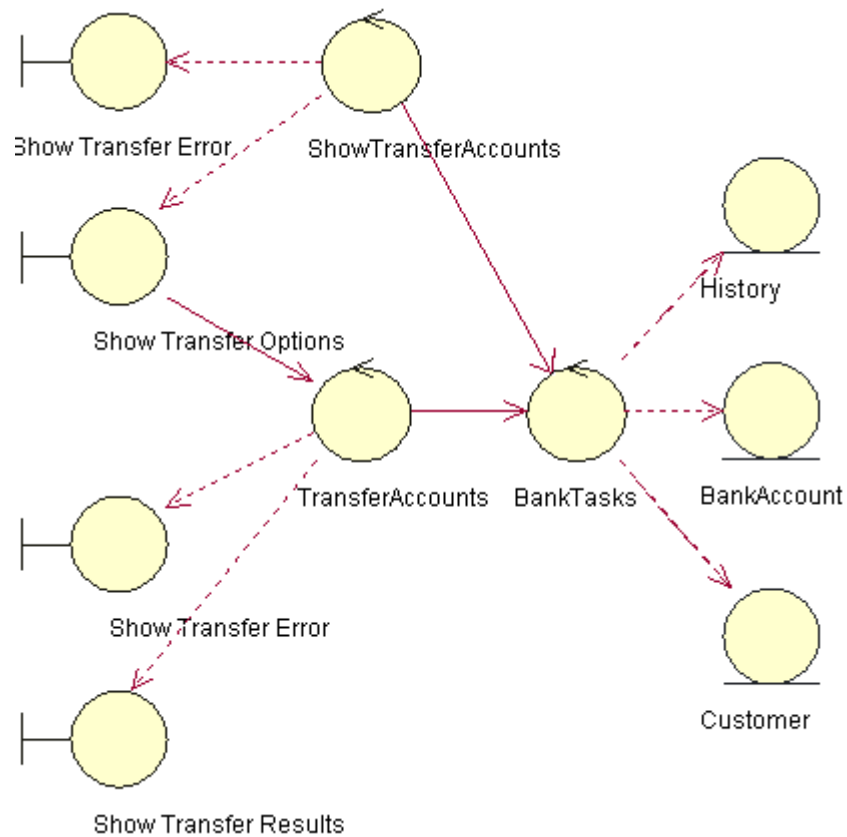




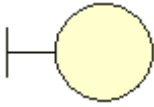


Figure 28. Analysis diagram

The icons we have assigned to each node represent the roles each will play in our application. This assignment very nearly represents our conclusions about the technology we will use to implement each type of object.

|  |  |
|--|--|
|  | <p>A Boundary, representing a point where data or control either enters or exits the application. In our application these will be implemented with JSP files.</p> |
|--|--|

|   |   |
|---|---|
|  | A Control, representing a point at which a computation or other significant state change is generated. In our application, the Make Request, Get Results node will be implemented by a servlet. The Perform Task node, committed as we are to enterprise beans at the back-end, will be implemented by an EJB Session bean. |
|  | An Entity, representing a persistent data object and the code that encapsulates it. In our application, these will be the EJB Container Managed Entity beans.   |

The arrows are also significant.

|   |  |
|---|--|
|    | A Communicates relationship: the object at the tail of the arrow will send a message (or pass a parameter list) to the object at the head. |
|    | An Instance relationship: the object at the tail of the arrow will create the object at the head.  |
|  | A Generalize relationship: the object at the tail of the arrow is a specialization of the object at the head.                              |

The object of a UML design effort is to create a class diagram documenting the classes of the application and their fields, methods, and parent types. This diagram would be decorated with notes detailing the intended function of each method and become the source document for the coding effort. The Rational Rose tool can communicate the internal representation of this diagram to VisualAge for Java, which will use it to construct the class skeletons into which code would be inserted.

We will now present the Consumer Banking applications class diagram and will detail how it executes the MVC pattern.

### 6.7.2 MVC and the Consumer Banking application

The Consumer Banking application implements the MVC design pattern, where the *view* is implemented using JSP files; the *controller* piece is implemented using servlets and Command beans; and the *model* is implemented using enterprise beans. Figure 29 illustrates this implementation in more detail.

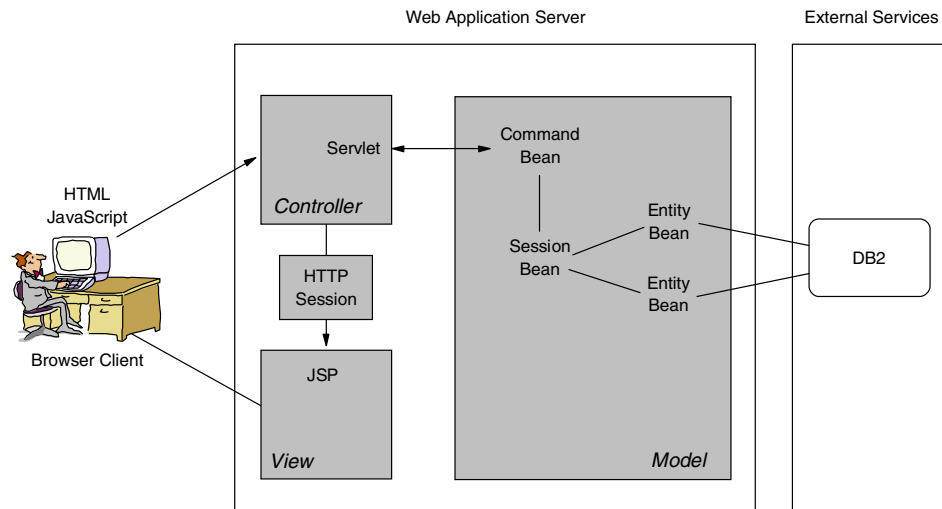


Figure 29. MVC implementation of the Consumer Banking application

#### 6.7.2.1 Consumer Banking view

We have already discussed the role of views in the MVC pattern. Here we will discuss the role of the application architect selecting these views.

Views might first be described with simple storyboards. These storyboards might be hand-drawn pages prepared at a conference table, slideware prepared for an architectural presentation, or HTML mockups built with a development tool. We used IBM's WebSphere Studio tool to build our mockups. Modelling with a development tool has the advantage of producing mockups that can be rolled over into production and generating maps that can be used to document the relationships among our views.

Our application implements a small set of use cases, as described in 6.7.1, "UML and the Consumer Banking Application" on page 104. These are:



- Customer Login
- List Accounts
- List Transactions
- Transfer Funds
- Sign Off

The Customer Login view simply requests a customer ID to log in to the application.

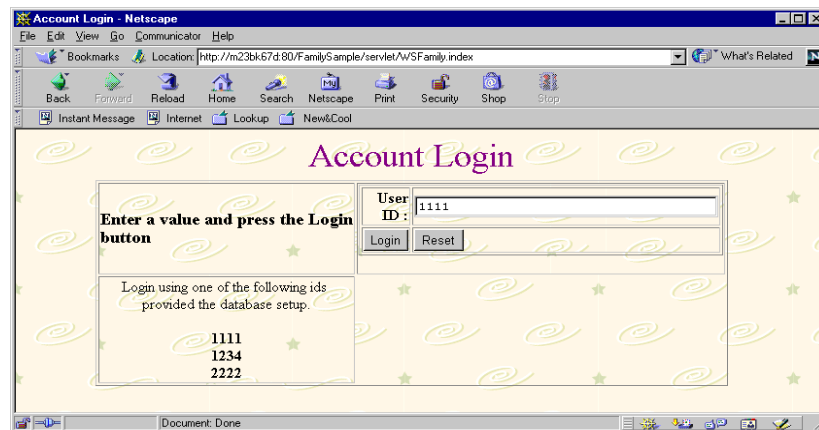


Figure 30. Customer login

Once logged in, the customer can choose an action from the following menu.

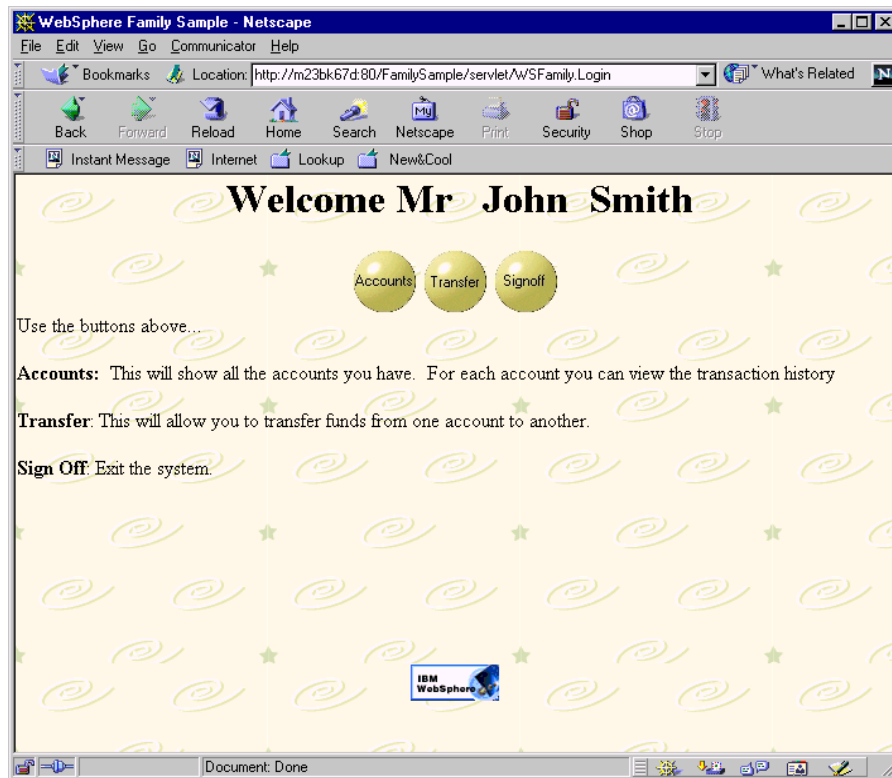


Figure 31. Menu options

Clicking the **Accounts** button will display all accounts this customer holds in the bank.

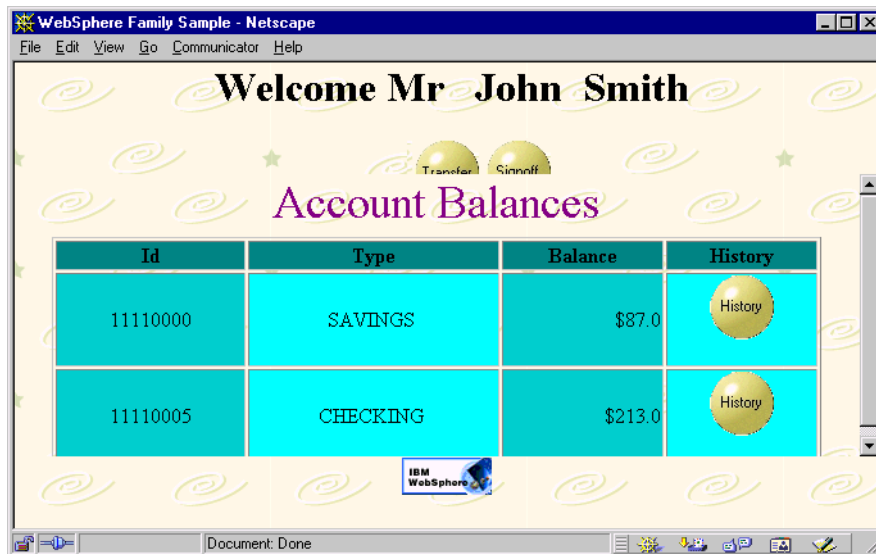


Figure 32. Displaying the account balances

From the Accounts view, a customer can view the transaction history associated with a particular account by clicking the **History** button.

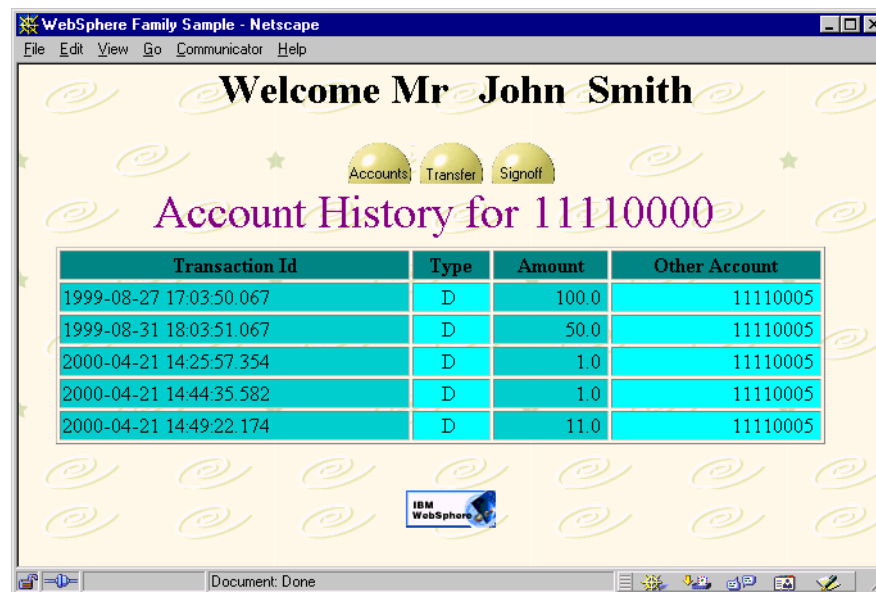


Figure 33. Displaying account history

Selecting the **Transfer** button will allow the customer to transfer money from one account to another.



Figure 34. Transferring funds

Clicking the **Sign Off** button will log the customer out of the application, and take him back to the beginning login screen.

Clearly, a more mature application may require many more views and more complex relationships between views, but the design of the customer's experience with the application will begin with the construction of such storyboards and maps.

Like the List Accounts use case, many of the pages in this application must include data in an attractively formatted page and collect an action to perform from the user. JSP technology provides HTML elements to manage the display of information and buttons, and Java language elements to insert data.

JSP files will be read by a JSP interpreter before being presented to the browser. This interpreter contains the definition of a Java servlet skeleton. The Java language elements of the JSP page will be copied into the servlet skeleton. The HTML elements will be written into `println` statements in the servlet skeleton. The combination of HTML and executed Java code will result in a full HTML page sent back to the browser in a HTTP response.

The JSPs include language elements appropriate to each of the functions of the view - HTML for formatting text and Java for performing the small amounts of logic required by a view for displaying data. JSPs, however, should *not* be used to implement controller logic. JSPs are meant to simplify the task of creating dynamic HTML by bridging the gap between HTML and servlet code. Embedding any kind of controller or business logic inside of a JSP violates the separation of concerns embraced by the MVC design pattern.

We also recommend using a development tool for constructing JSPs. While incorporating two languages in a JSP provides a means for generating dynamic HTML, writing JSP files by hand can become complicated. A development tool, such as WebSphere Studio, can manage these two different programming models behind a simple, consistent graphical interface, and ease the development process.

#### **6.7.2.2 Consumer Banking controller**

The Consumer Banking application controllers are a collection of Java servlets and Command beans that implement some business logic of the application. They will select data from the bank's model of customers, accounts and transaction histories, and will update the model according to the rules laid out by the bank and customer requests.

The Consumer Banking application has five controllers. These are:

Login - authenticates the customer's account ID against the customer model

ShowAccounts - collects account information for all accounts held by a customer

ShowHistory- collects the transaction histories for the accounts identified by the customer's account ID

ShowTransferAccounts - collects the current account information for the accounts identified by the customer's account ID, and displays them as source and destination accounts

TransferAccount- transfers money between selected accounts and updates the transaction history

Each controller is implemented as a servlet that encapsulates a Command bean. For example, the TransferAccount controller is the TransferAccount servlet that encapsulates calls to the TransferAccount\_CMD bean.

The servlet is implemented as part of the flow of control mechanism for the application. It implements doGet and doPost interfaces that will be invoked by the views in response to requests from the customer. After preparing the Command bean and calling it to perform the required function, the servlet will construct an HTTP session containing the result to be presented to the customer and will call the appropriate JSP (view) to be displayed. Remember that in the event of an error, the appropriate view may be an error page.

Each of the controller servlets of the consumer banking application will have a similar structure. Each will collect attributes from the request and construct a Command bean, will communicate that Command bean to the model, and will pass that Command bean and its results via the HTTP session to the view that will display the results.

### **6.7.2.3 Consumer Banking model**

The model of the Consumer Banking application is a collection of enterprise beans that manages access to the persistent data of the business, and provides business logic around this persistent data. The persistent entities of the bank are the customers, accounts, and transaction histories, which are encapsulated behind the Customer, BankAccount, TranRecord Entity beans. Each of these Entity beans are container-managed and use DB2 as their persistent store. A BankTasks Session bean provides business logic around these entities, such as collecting the accounts associated with a customer, and handling the transfer of money from one account to another.

## **6.7.3 Consumer Banking implementation**

Our Consumer Banking application class diagram documents the classes of the application's control and model layers. We have chosen not to include the classes of the view layer in this diagram.

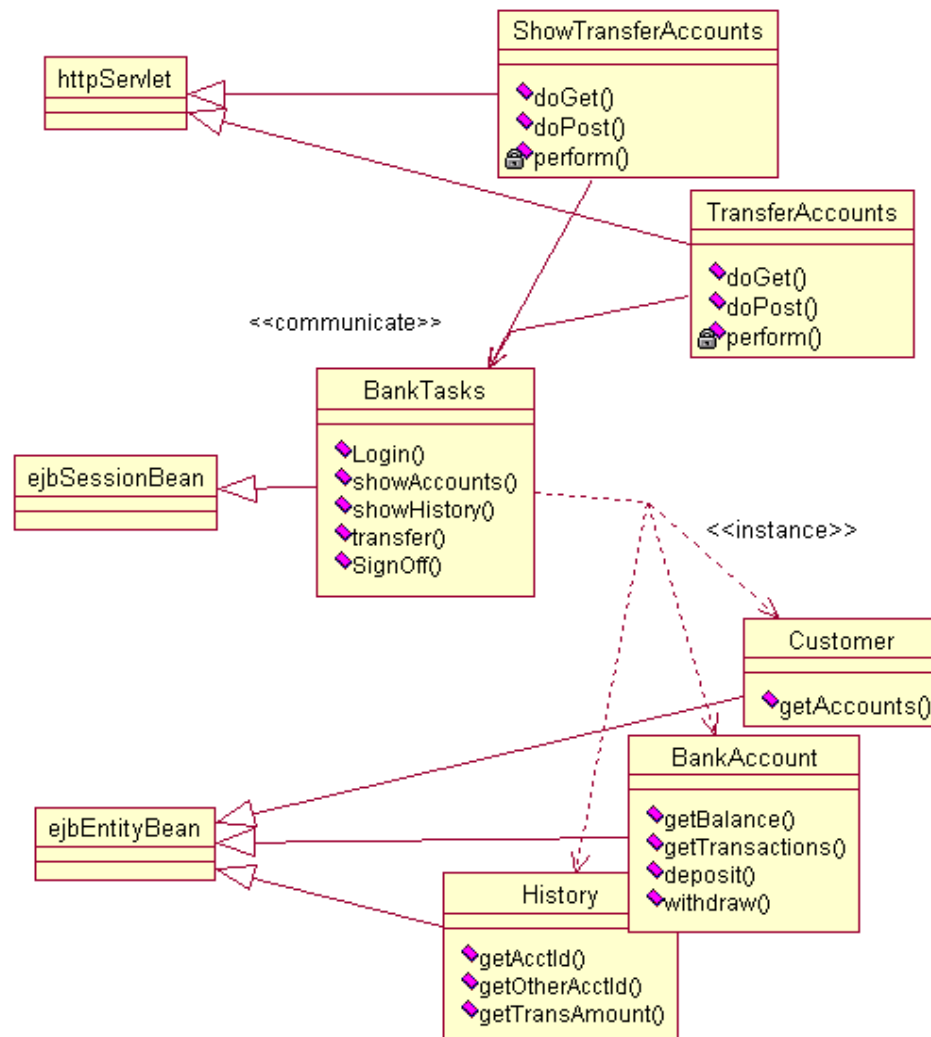


Figure 35. Consumer Banking application class diagram

We will now walk through the execution of our ShowTransferAccounts function. Our ShowTransferAccounts function begins when the user selects the Transfer option from the main menu. The menu is displayed as shown in Figure 36.



Figure 36. Consumer Banking main menu

An example of the main menu HTML is shown in Figure 37.

```
<TABLE>
<TR>
<TD>
    Handle the Show Accounts case
</TD>
<TD nowrap>
<FORM
    name="showTransferAccounts"
    method="POST"
    action=
        "/FamilySample/servlet/WSFamily.ShowTransferAccounts"
    >
<A HREF=
    "document.forms['showTransferAccounts'].submit()">
<IMG src="/FamilySample/WSFamily/transfer_o.gif" border="0" width="50"
    height="50" name="transfer" alt="transfer"> <INPUT
    type="hidden">
</A>
</FORM>
</TD>
<TD>
    Handle the Signoff case
</TD>
</TR>
```

Figure 37. HTML for the main menu

In the second table data entry, bracketed by `<TD>` and `</TD>`, we see the statements that will invoke the `ShowTransferAccounts` control servlet. More precisely we see a form, bracketed by `<FORM>` and `</FORM>`, embedding an anchor, bracketed by `<A>` and `</A>`.

The anchor describes what should be done with the anchored object, in this case, an image, when clicked with the mouse. The `HREF` term of the anchor requires that the form in this document named “`showTransferAccounts`” be



submitted, or executed. The form's action parameter specifies the servlet to execute.

So, when the image is clicked, the browser will construct and submit the following HTTP request to the Web server:

`http://<hostname>/FamilySample/servlet/WSFamily.ShowTransferAccounts`

The customer will then be presented with a list of their accounts so they can select a source and destination account, and a text entry box where they can enter an amount for the transfer.

| Select From Account   |          |          |          |
|-----------------------|----------|----------|----------|
| From                  | Account  | Type     | Balance  |
| <input type="radio"/> | 11110000 | SAVINGS  | \$100.00 |
| <input type="radio"/> | 11110005 | CHECKING | \$200.00 |

| Select To Account     |          |          |          |
|-----------------------|----------|----------|----------|
| From                  | Account  | Type     | Balance  |
| <input type="radio"/> | 11110000 | SAVINGS  | \$100.00 |
| <input type="radio"/> | 11110005 | CHECKING | \$200.00 |

Amount

Figure 38. Transfer funds menu

The controller (the servlet) will need to collect the user's account information from the model and, if all goes well, select an updated view. If all does not go well, it must select an error view.

A servlet will implement a `doGet()`, responsible for returning small amounts of data to the browser, and a `doPost()`, responsible for returning large amounts of data. In the several years that have passed since this feature of the servlet was defined, the number of bytes that pass as a small amount of data has grown to the point that there is not much point in distinguishing between the two. Most servlets will implement a single function to prepare data for the browser, which both `doGet()` and `doPost()` will invoke. So will ours.

Our `ShowTransactionAccount`'s `performTask()` function will implement our `doGet()` and `doPost()`. Very roughly, it will look like Figure 39.

```

public void performTask (
    HttpServletRequest      request,
    HttpServletResponse    response
)
{
    try
    {
        collect the customer's accounts from the Model
        select a display for displaying the customer's
        accounts
    }
    catch (Throwable theException)
    {
        handleError(request, response, theException);
    }
}

```

Figure 39. *ShowTransferAccounts* servlet *performTask()*

Our servlet creates a *ShowAccounts* Command bean that implements the Command pattern. Here we need only to understand that a Command bean is responsible for collecting input data, passing that data to business logic within the model to be executed, and exposing the result data in an appropriate format to be used in the view. This code will look something like Figure 40.

```

// Collect the customer id from the session context
HttpSession session = request.getSession(true);
String customerId = (String)session.getValue("customerId");

// create a ShowAccounts Command Bean
Command.ShowAccounts_CMD showAccounts_CMD = null;
showAccounts_CMD = (Command.CommandBean)java.beans.Beans.instantiate (
    getClass().getClassLoader(),
    "Command.ShowAccounts_CMD"
);

// set command session context and parameters
showAccounts_CMD.setSession(session);
showAccounts_CMD.setCustomerId(customerId);

showAccountsBean.perform();

```

Figure 40. *Creating a Command bean*

When the customer logged in, providing a customer ID and a password, the login servlet would have added this ID to the session context making it available to all the subsequent cases. We will use the customer ID to get the list of customer accounts from the model.

After the Command bean is executed, it is added to the HTTP session so that the result data contained within it is accessible to the JSP.

```
// add to Command bean to the session
session.putValue("showAccounts_CMD", showAccounts_CMD);

// Call the output page. If the output page is not passed
// as part of the URL, the default page is called.
callPage
    (getPageNameFromRequest(request), request, response);
```

*Figure 41. Calling the JSP (view)*

The ShowAccounts\_CMD Command bean communicates with the model and collects the result data in its perform() method, shown in Figure 42.

```

public void perform()
    throws CustNotFoundException, java.lang.Exception
{
    // we need an Access bean for the BankTasks
    BankTasksAccessBean bankTaskBean
        = new BankTasksAccessBean();

    // tell the Access bean where the model is located
    bankTaskBean.setInit_NameServiceTypeName
        ("com.ibm.ejs.ns.jndi.CNInitialContextFactory");
    bankTaskBean.setInit_NameServiceURLName
        ("m23bk62w.itso.ral.ibm.com:910");

    // find all accounts for the customerId
    try
    {
        accountBeanTable =
            bankTaskBean.lookupAccounts(customerId);
    }
    catch (javax.ejb.FinderException notFound)
    {
        throw new CustNotFoundException
            (customerId + " does not exist in our records");
    }
    finally
    {
        bankTaskBean = null;
    }
}

```

Figure 42. *ShowAccounts\_CMD perform()*

The model is implemented with enterprise beans, and it is the BankTasks Session bean that contains most of the business logic for our simple model. The Session bean communicates with Entity beans representing the customer and account records kept by the bank. To get our list of customer accounts from the model we need to fetch the BankTasks Session bean and ask it to collect this information from the bank's customer and account entities.

Following the normal steps for EJB client communication, the perform() method accomplishes the following:

1. Initializes the JNDI context (name server)
2. Looks up the BankTasks home from the JNDI context

3. Creates the BankTasks using the home
4. Invokes the method of the BankTasks to retrieve all accounts for a specific customer.

We use an Access bean generated by VisualAge for Java for client communication with the BankTasks Session bean. Access beans are discussed in further detail in *Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere*, SG24-5754.

The BankTasks Access bean effectively performs the first three steps of standard EJB client programming for us, given the following parameters:

- The address of our JNDI name server and port number:

```
m23bk62w.itso.ral.ibm.com:910
```

- The name of our JNDI Initial Context Factory:

```
com.ibm.ejs.ns.jndi.CNInitialContextFactory
```

- The JNDI name of our BankTasks Session bean home: BankTasks (this value is hard-coded into the Access bean)

The first time a method is invoked on the Access bean, which represents a remote method call on the BankTasks Session bean, the Access bean will use lazy initialization to establish communication with the remote bean. Then the remote method will be invoked.

It is important to note that including these initialization parameters as text literals inline is a really bad idea. At the very least, these literals ought to be embedded in the base class from which the ShowAccountsBean is derived, so that they do not need to be typed into each derived bean. A better idea would be to include these literals, which represent features of the application server configuration, in a properties file so the changes to the configuration could be made independent of the code. For the purposes of explaining the beans, this code is sufficient.

Now, let's take a look at some code inside the BankTasks Session bean that encapsulates the logic of our model.

```

public AccountAccessBeanTable lookupAccounts (
    String                theCustomerId
)
    throws javax.naming.NamingException,
           java.rmi.RemoteException, javax.ejb.FinderException
{
    Customer customer = null;
    java.util.Enumeration accounts = null;

    try
    {
        create an object to contain our results
        collect our results
        populate our results object with result data

        return beanTable;
    }
    catch (CreateException ce)
    {
        throw new java.rmi.RemoteException(ce.getMessage());
    }
}

```

Figure 43. BankTasks lookupAccounts()

Looking a little more closely, we see:

```

// create an object to contain our results
AccountAccessBeanTable beanTable
    = new AccountAccessBeanTable();

```

The AccountAccessBeanTable was built for us by the Access bean feature of VisualAge for Java. Access bean tables simply encapsulate a list (or table) of Access beans, which in turn, represent the actual remote enterprise beans.

Next, we find the Customer Entity bean based on the specified customer ID, and collect its associated Accounts, which are Entity beans themselves.

```

// collect our results
CustomerKey customerKey = new CustomerKey(theCustomerId);
customer = getCustomerHome().findByPrimaryKey(customerKey);
java.util.Enumeration accounts = customer.getAccounts();

```

We now have a list of customer accounts, and these are used to populate the AccountAccessBeanTable to be returned from this method, as shown in Figure 44.

```

// populate our results object with results
while (accounts.hasMoreElements())
{
    Account account = null;
    account = (Account)
        javax.rmi.PortableRemoteObject.narrow
            (accounts.nextElement(), Account.class);
    AccountAccessBean bean
        = new AccountAccessBean(account);
    bean.refreshCopyHelper();
    beanTable.addRow(bean);
}

```

*Figure 44. Populating the results*

Recall that we have received a customer's request to transfer funds. This request came to us as an HTTP request, coded from an HTML form. We need to reply with a page displaying the legal source "from" accounts and destination "to" accounts, and requesting the amount to be transferred.

We have built the initial view using JSP files that contain nothing more than standard HTML. We have built the ShowTransferAccounts servlet at the control layer and it has constructed a ShowAccounts Command bean to send a command to the model to collect the results. The Command bean uses an Access bean, which delegates calls to the actual remote BankTasks Session bean. The BankTasks bean is used to apply the business rules for selecting a customer's accounts and that list is returned to our control servlet. Our servlet places the Command bean, filled with data, into the HTTP session, so that data is accessible to the JSP page which constructs the view to be returned to the browser.

We introduced the following line of code from the servlet in Figure 41, that invokes the JSP to return a resultant HTML view:

```

callPage
    (getPageNameFromRequest(request), request, response);

```

callPage() calls a JSP, which takes on the responsibility of constructing the view, or rather, an HTML file to get returned to the browser. In our example, the servlet we created extends a base servlet provided with WebSphere called `com.ibm.servlet.PageListServlet`, which implements the callPage() method.

callPage() will pass control to the URL given by the first argument or, if there is no name in the first argument, from an XML configuration file. In our case,

there will be no name in the first argument and the call will refer to the configuration file. The callPage code will construct the configuration file name by appending “.servlet” to the class name, such as “ShowAccounts.servlet”. This configuration file was created for us by WebSphere Studio, and looks like this:

```
<servlet>
  <page-list>
    <default-page>
      <uri>
        /FamilySample/WSFamily/ShowTransferAccountsResults.jsp
      </uri>
    </default-page>
    <error-page>
      <uri>
        /FamilySample/WSFamily/ShowTransferAccountsError.jsp
      </uri>
    </error-page>
    <page>
      <uri>
        /FamilySample/WSFamily/Login.jsp
      </uri>
      <page-name>
        Commands.NotLoggedInException
      </page-name>
    </page>
  </page-list>
  <code>
    WSFamily.ShowTransferAccounts
  </code>
</servlet>
```

Figure 45. ShowTransferAccounts servlet XML configuration file

In the page-list, the default page is ShowTransferAccountsResults.jsp. We also see a generic error page and a NotLoggedInException page specified. More information about the PageListServlet can be found at:

<http://as400.rochester.ibm.com/products/websphere/docs/doc/apidocs/com.ibm.servlet.PageListServlet.html>.

The resulting JSP file will define a procedure for generating our page. Very roughly, our ShowTransferAccountsResults.jsp looks something like this:

- Start the HTML document
- Get the Command bean from the HTTP session
- Start the body



- Start the form
- Start the “from” account table with radio buttons
- For each account of the Command bean
  - Build a row in the table
  - End the “from” table
- Start the “to” account table with radio buttons
- For each account of the Command bean
  - Build a row in the table
  - End the “to” table
- Build the amount and submit items
- End the form
- End the body
- End the HTML document

Many of these steps will be programmed by embedding the appropriate HTML text in the source, as though this were a simple HTML file. Consider the following code:

```
<HTML>
get the Command bean from the request
<BODY>
<FORM
    action="/FamilySample/servlet/WSFamily.TransferAccount"
    method="POST" target="_self">
<TABLE>
    for each account of the Command bean
        build a row in the table
</TABLE>
<TABLE>
    for each account of the Command bean
        build a row in the table
</TABLE>
</FORM>
</BODY>
</HTML>
```

Figure 46. *ShowTransfer AccountsResults JSP*

Several of the steps are programmed by writing Java fragments into the source. For each account, fragments may be programmed with the code shown in Figure 46.

```

<%
Object _p0 = showAccounts_CMD.getFrom(0);
Object _p0_0 = showAccounts_CMD.getAccountId(0);
Object _p0_1 = showAccounts_CMD.getAccountType(0);
Object _p0_2 = showAccounts_CMD.getBalance(0);

for (int _i0 = 0; ; )
{
%>
    <TR>
    <TD align="center" bgcolor="#00cccc" class="TBL_ODD">
        <INPUT name="fromAccount" type="radio"
            value="<%= _p0_0 %>">
    </TD>
    <TD bgcolor="#00ffff" class="TBL_EVEN">
        <%= _p0_0 %>
    </TD>
    <TD bgcolor="#00cccc" class="TBL_ODD">
        <%= _p0_1 %>
    </TD>
    <TD align="right" bgcolor="#00ffff" class="TBL_EVEN">
        $<%= _p0_2 %>
    </TD>
    </TR>
<%
    _i0++;
    try
    {
        _p0 = showAccounts_CMD.getFrom(_i0);
        _p0_0 = showAccounts_CMD.getAccountId(_i0);
        _p0_1 = showAccounts_CMD.getAccountType(_i0);
        _p0_2 = showAccounts_CMD.getBalance(_i0);
    }
    catch (java.lang.ArrayIndexOutOfBoundsException _e0)
    {
        break;
    }
}
%>

```

Figure 47. Java code and HTML

Each of the fragments contained between “<%” and “%>” specifies Java code that must be embedded directly into the servlet generated for the JSP. The remaining fragments are HTML, which will be embedded in print statements within the JSP servlet.

We've noted before that the JSP integrates two different languages into a single file. WebSphere Studio simplifies this work by generating much of the Java code fragments embedded in JSP files, such as the one in Figure 47. This JSP file will build an HTML file similar to that seen in the following figures. The resulting display will look like that shown in Figure 38 on page 123.

```
<HTML>
<TITLE>
Select Fund Transfer Accounts
</TITLE>
<BODY>
<FORM
  action="/FamilySample/servlet/WSFamily.TransferAccount"
  method="POST" target="_self">
<CENTER>
<TABLE border="1" width="600">
  <CAPTION>Select From Account</CAPTION>
  <TR>
    <TH>From</TH>
    <TH>Account</TH>
    <TH>Type</TH>
    <TH>Balance</TH>
  </TR>
  <TR>
    <TD align="center">
      <INPUT type="radio" name="fromAccount"
        valueproperty="11110000">
    </TD>
    <TD>
      11110000
    </TD>
    <TD>
      SAVINGS
    </TD>
    <TD align="right">
      $100.00
    </TD>
  </TR>
  remaining rows...
</TABLE>
```

Figure 48. HTML file

```

<TABLE border="1" width="600">
    the To table...
</TABLE>
<BR>
<TABLE border="1">
    <TBODY>
        <TR>
            <TD align="center">
                <INPUT type="submit" value="Transfer"></TD>
            <TD>Amount</TD>
            <TD><INPUT size="20" type="text" name="amount"></TD>
            <TD><INPUT type="reset" name="Reset"></TD>
        </TR>
    </TBODY>
</TABLE>
</FORM>
</BODY>
</HTML>

```

Figure 49. HTML file - continued

## 6.8 Implementing topology 5: the Mortgage Payment System

Web-enabling legacy applications is the distinguishing feature of topology 5. This simply means creating a Web application that exposes the functionality provided by an existing legacy application to a Web browser.

For illustrating topology 5, imagine our bank has inherited an application from a recently purchased mortgage company. This mortgage payment service is an MQSeries-based service that will accept mortgage payments by day and credits them to customer accounts at night after computing interest.

The Mortgage Payment System is a teller's console at the mortgage company offices. It allows the teller to post a customer's mortgage payment to the system, while dropping the customer's money into the cash box. This application had been accessed by clerks at the company offices through dedicated terminals running a fat client. Our goal is to make that application available to our bank's tellers through a Web browser. This Web application will implement the MVC design pattern in a fashion similar to the Consumer Banking application.

Our topology for implementing the Mortgage Payment System as a Web application is illustrated below.

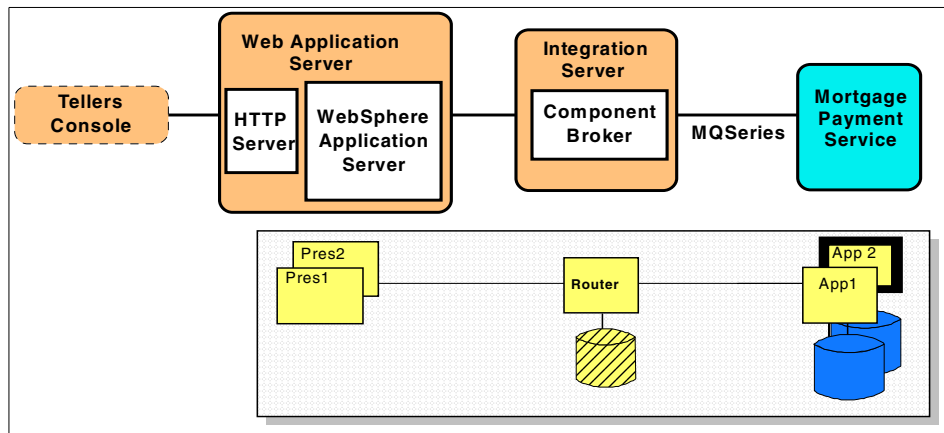


Figure 50. Mortgage Payment System topology

We will use MQSeries for the connection between the router (Component Broker) and the back-end application. MQSeries is an asynchronous guaranteed delivery service. Messages accepted by MQSeries will be delivered to the destination application despite interruptions in network connections or failures of the target application. It is essential to the bank that mortgage payments are accurately logged, and we assume that if a payment message is successfully sent to the MQ queue, then that payment will be processed and logged accordingly.

### 6.8.1 UML and the Mortgage Payment System

The UML activity diagram for the Mortgage Payment System is shown in Figure 51.

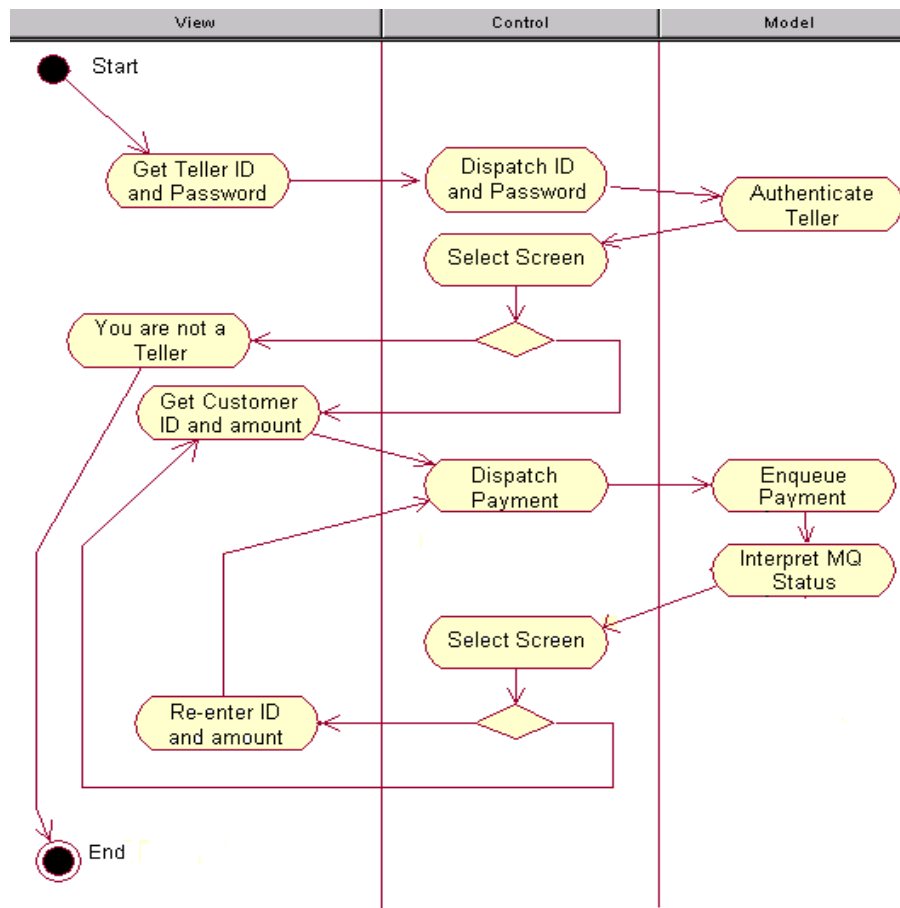


Figure 51. Mortgage Payment System Activity Diagram

### 6.8.2 MVC and the Mortgage Payment System

Our Mortgage Payment System application implements the MVC pattern similar to the Consumer Banking application. The *view* requires JSPs, the *controller* implemented with a servlet, and the *model* will be a Command bean and EJB that sends payment messages to the MQSeries queue to be processed by the existing mortgage application. This implementation and how it fits into the MVC design is illustrated in Figure 52.

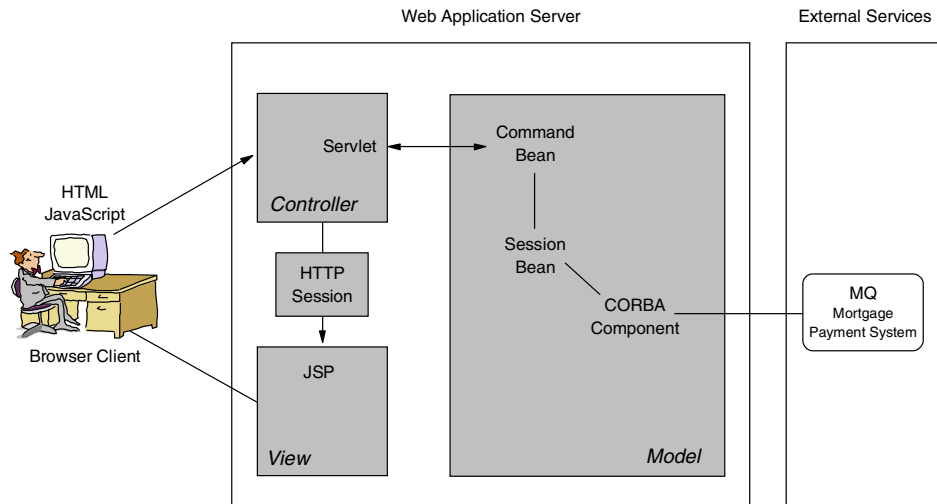


Figure 52. Mortgage Payment System MVC implementation

### 6.8.2.1 Mortgage Payment System view

The views for this application simply consist of input and result pages, implemented using JSPs. The teller enters the customer's mortgage account number and payment amount to be processed.

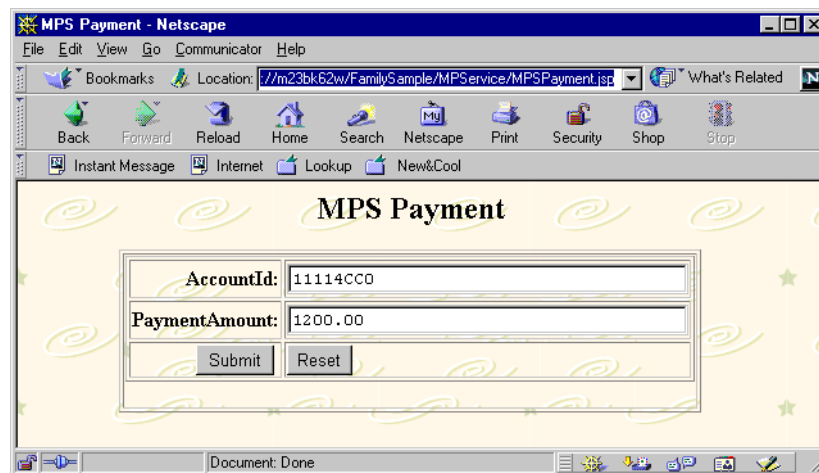


Figure 53. Mortgage Payment Service - screen 1

When the payment is accepted by the application and sent for processing to the MQ queue, a results page will appear.

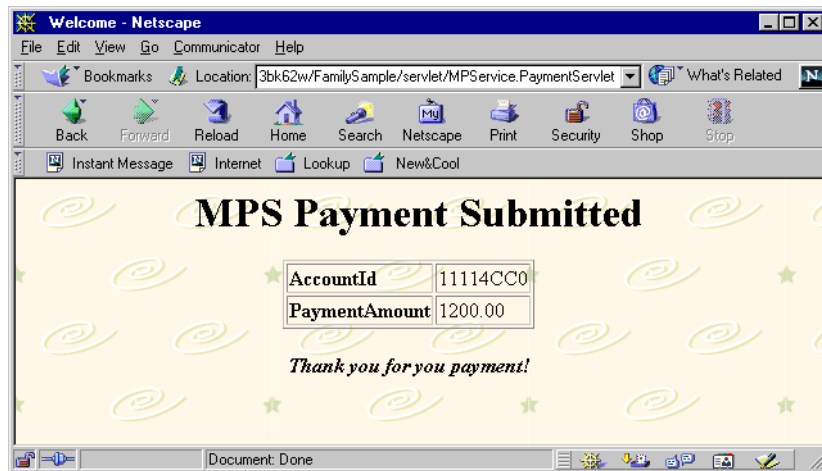


Figure 54. Mortgage Payment Service - screen 2

#### 6.8.2.2 Mortgage Payment System Controller

With the simple functionality required for the Mortgage Payment System, only one controller is needed, the MortgagePayment controller. This controller receives the payment request, passes this request to the model, and returns an appropriate view page indicating that the payment was submitted or failed to be processed.

This controller is implemented in a fashion similar to the controllers in the Consumer Banking application. We implement a servlet (PaymentServlet) that encapsulates a Command bean (Payment\_CMD) that makes the appropriate calls to the model.

#### 6.8.2.3 Mortgage Payment System model

The model for the Mortgage Payment System is an enterprise bean that handles communication to an MQSeries queue. This enterprise bean provides the necessary integration with the MQSeries legacy application to implement topology 5.

IBM provides several ways of communicating to MQSeries from enterprise beans:

- **MQSeries Java API** - A set of Java APIs come with MQSeries, which can be used directly within an enterprise bean to communicate with MQ.
- **MQSeries Connector** - VisualAge for Java provides an MQSeries Connector that provides access to MQSeries through a standard set of interfaces called the Common Connector Framework. VisualAge for Java



also provides tooling to create a Java bean that uses the MQSeries Connector. This bean can then be used within an enterprise bean.

- **MQSeries Application Adaptor** - The Component Broker development environment provides tooling for creating CORBA components and enterprise beans which communicate to MQSeries through the MQSeries Application Adaptor provided with the Component Broker runtime environment.

For this sample, we make use of the third option, the MQSeries Application Adaptor, because of the advantages it provides us. Using this adapter, components can encapsulate outbound and inbound messages, communicate with message queues through pooled connections, and can participate in distributed transactions. The ability to have components that communicate with various disparate back-end systems and participate in distributed transactions is a powerful distinguishing feature of WebSphere Enterprise Edition. Component Broker supports two-phase commit transactional support across “container-managed” components backed by relational databases (such as DB2, Oracle, Informix), legacy resource managers (CICS, IMS), and MQSeries.

We use the tools provided in Component Broker to create the enterprise bean to send outbound payment messages to MQSeries. First of all, this eases the development of this enterprise bean, because the tools will generate nearly all of the code. The only code we must write ourselves is the actual formatting of the message to be sent to the queue. Also, this will allow each payment sent to the back-end mortgage application to be wrapped inside the context of a transaction, meaning that the payment message will not actually be placed on the queue until the transaction is committed.

In this sample, the Command bean (of the controller) explicitly begins and ends the transaction that wraps the calls to the enterprise bean, which results in a “put” to the message queue. Transaction coordination around the MQ enterprise bean component is discussed in more detail in 6.9, “Implementing topology 6: the Consumer Banking Plus application” on page 141.

### 6.8.3 Implementing the Mortgage Payment System

We have already gone into quite some detail on the implementation of the JSPs, servlets, Command beans and EJB client programming of the Consumer Banking application. Here, we will show the interesting differences needed for the Mortgage Payment System application.

When a mortgage payment is submitted from the initial browser view, the servlet MPService.PaymentServlet is called with the following URL:

`http://<hostname>/FamilySample/servlet/MPService.PaymentServlet`

The `PaymentServlet` creates and executes the `Payment_CMD` Command bean. After the Command bean is executed, it is added to the HTTP session so that the result data contained within it is accessible to the result JSP. The result JSP in this sample simply confirms that the payment was submitted properly or failed.

The `Payment_CMD` Command bean calls the MQ EJB component (`MPSOutbound`) to send the outbound payment message. The Command bean explicitly begins a transaction, calls the `put()` method of the `MPSOutbound` bean, and then either commits or rolls back the transaction. If something goes wrong, the transaction is rolled back and the message will not be placed on the queue. If everything goes well, when the transaction is committed, the Component Broker runtime will place the message on the queue. The `Payment_CMD` `perform()` method can be seen in A.2.5, “`Payment_CMD.java`” on page 281.

The `MPSOutbound` bean that communicates to `MQSeries` is developed using the Component Broker toolkit. We used the command line utilities `mqaeejb`, `jetace`, and `cbejb` to generate and deploy the `MPSOutbound` Session bean. The use of these tools is discussed in Chapter 10, “Sample application implementation” on page 179. During this process a CORBA component is created that actually makes the calls to the message queue through the `MQSeries` Application Adaptor. The Session bean simply delegates its calls to this MQ-backed CORBA component. The only code we had to write was the code that formats the actual message that is placed on the queue.

The steps for developing the `MPSOutbound` enterprise bean are described in detail in 10.4, “Creating and Installing the Mortgage Payment System application” on page 205. Also for more information on Component Broker and `MQSeries` integration, see the following guides included in the Component Broker documentation:

- For Component Broker version 3.x, *MQSeries Application Adaptor Quick Beginnings* and *MQSeries Application Adaptor Concepts and Development Guide* can be downloaded from <http://www.ibm.com/software/webservers/appserv/doc/v30ee/cbpdf/>.
- For Component Broker version 3.5, *MQSeries Application Adaptor Development Guide* can be downloaded from <http://www.ibm.com/software/webservers/appserv/doc/v35/ee/cbpdf.html>.

---

## 6.9 Implementing topology 6: the Consumer Banking Plus application

The Consumer Banking Plus application integrates the existing Consumer Banking application with the Mortgage Payment System, giving our customer's the opportunity to make mortgage payments from either their checking or savings account over the Internet.

If we are going to save our customer this trip to the bank, we must find some way to assure the customers and bank officers that money can be reliably moved from one account to the other: that neither the customer nor the bank will come up short no matter what happens to the hardware or software elements of the transaction.

We can satisfy this requirement with Component Broker. As discussed in previous sections of this book, Component Broker's Object Transaction Service (OTS) and various *application adaptors* provide applications with the ability to transactionally coordinate components that access multiple types of disparate back-end systems. By deploying our enterprise beans inside of the Component Broker runtime environment, we can be assured that our transactions across components using DB2 and MQSeries will have two-phase distributed commit transactional coordination. In particular, we are assured that when our customers transfer money from one account to another, either the deduction and deposit will both succeed, or they will both be rolled back correctly.

While applications running in the WebSphere Advanced Edition can start, commit, and roll back transactions for components accessing DB2 and MQSeries, they cannot create a transaction spanning both. It cannot create a transaction that will cover the deduction from a saving account persisted on DB2 and a deposit to a mortgage account that has been persisted with an MQSeries "put" to a back-end application.

A WebSphere Advanced Edition application could create two independent transactions of course and begin the pair of transactions, then test for exceptions on both transactions before deciding to commit or rollback both transactions itself. There are, however, two problems with this solution.

First, it obliges our application to perform this transaction management step at every operation between the two (or more) transaction managers. In our application this will happen exactly once. In other applications this may happen many times and the sets of transactions to be handled may be at opposite ends of long code paths. As the complexity of the application increases, the complexity of the management that the application must do will

increase. The architect should be extremely reluctant to take on a task of arbitrary complexity when a solution already exists.

Second, there is no guarantee that the application itself can make it from the commit of the first transaction to the commit of the second. There is always the possibility that the application will fail precisely at this most vulnerable point. To ensure the completeness of the transaction, the application will need to build some sort of transaction log so that on a restart it can verify that previous executions have not left the transactions it was managing in an incomplete state.

We now see that in order for an application (which accesses both relational and non-relational resource managers) in WebSphere Advanced Edition to manage its transactions correctly, it doesn't simply need to reproduce some of the work of a transaction manager, it must be a transaction manager. We can now see that the architect would not be taking on a task of completely arbitrary complexity. The task is precisely as complex as building a transaction manager, but this should not be an encouraging insight.

Remarkably, we see none of this in our code. We use container-managed persistence for our application. When we ask one of our home objects for a Session bean, it is given a transaction context. When the Session bean then asks for an Entity bean, that transaction context will flow across to the Entity bean. When we set the attributes of an Entity bean or call a business method on the session or Entity bean, the container begins and commits the transaction or rolls back our changes if any part of the transaction fails.

We do not write any new code to take advantage of the Java Transaction Service (JTS) of the Component Broker runtime environment. The JTS implemented in Component Broker is, in fact, a thin interface layer built on top of the CORBA Transaction Service. Another value-add of the Component Broker JTS over that implemented in WebSphere Advanced Edition is the additional CORBA services provided by the Component Broker runtime that work hand-in-hand with the Transaction Service. For example, the Concurrency Service allows you to configure optimistic or pessimistic locking policies, and ensures that simultaneous users of a resource will get mutually consistent results.

What is significant in the Consumer Banking Plus application is that with very little effort, we extended the Consumer Banking application with the Mortgage Payment System and never lost any customer's money between accounts in the Consumer Bank and the Mortgage Company. This is the single feature we were concerned to preserve and demonstrate. And it is important to us that we achieved this end without the construction of special-purpose code.

It is the responsibility of the EJB containers to provide transaction services to the enterprise beans. As the services they count on become more capable, our beans become more capable. This application now spans transactions across components persisted to multiple types of back-end systems without having to introduce new transactional code. The promise of the EJB architecture has to be able to construct our business logic without spending our efforts towards developing code that handles data integrity and data consistency, as these services are to some extent managed by the EJB infrastructure. Component Broker delivers on that promise.

### 6.9.1 MVC and the Consumer Banking Plus application

The integration of the two existing applications to create the Consumer Banking Plus application does not require us to build new components, only extend the existing views, controllers, and model with some slight modifications. This application is essentially the Consumer Banking application, but allows for the additional mortgage account type, by displaying mortgage account information as well as displaying mortgage accounts as an account type that money can be transferred into, but not debited from.

Figure 55 shows a simple overview of integrating the two previously discussed applications to form the extended Consumer Banking Plus application without altering the existing MVC design.

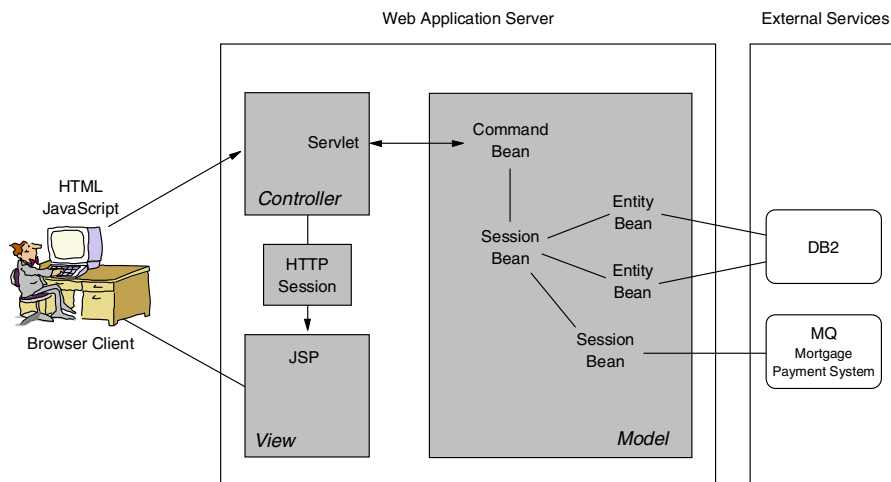


Figure 55. MVC implementation of the Consumer Banking Plus application

### 6.9.1.1 Consumer Banking Plus view

In 6.7.2.1, “Consumer Banking view” on page 114, views are shown for the following use cases:

- Customer Login
- List Accounts
- List Transactions
- Transfer Funds
- Sign Off

Only the List Accounts and List Transactions views have been altered for the Consumer Banking Plus application to include mortgage account types.

When a customer who has a mortgage account has logged in and requests account information, this mortgage account will be displayed along with his other accounts.

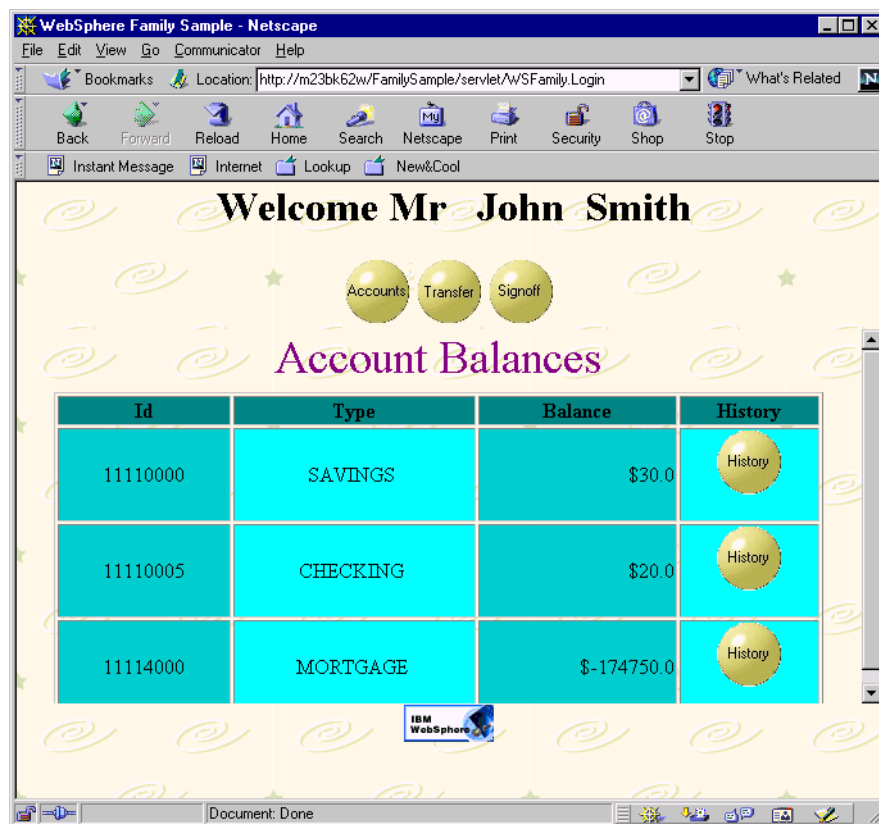


Figure 56. Viewing the account balances

Then when that customer chooses to transfer money from one account to another, the mortgage account will show as a “to” account, but not as a “from” account.



Figure 57. Transferring funds

If money is chosen to be transferred into the mortgage account, once the transaction has completed, this view will display the updated account balances. If the transaction failed for some reason, no updates will occur and the view will be redisplayed with the original information.

#### 6.9.1.2 Consumer Banking Plus controller

The controllers for the Consumer Banking application remain the same. No new controllers are needed to integrate with the Mortgage Payment System.

### **6.9.1.3 Consumer Banking Plus model**

The Consumer Banking Plus model includes all components from the original Consumer Banking application and the Mortgage Payment System. Although no new components need to be created, we inserted the existing customer mortgage account numbers and balance information into the Consumer Banking DB2 database in order for the BankAccount component to recognize these new mortgage account entities, without any code modifications or additions to the existing Model.

Only a few minor modifications to the ShowAccounts\_CMD Command bean are needed in order to provide the JSP view with the appropriate list of “to” and “from” accounts, where the “from” account list is filtered to not include mortgage accounts.

In the Mortgage Payment System application, its one Command bean contained the logic to communicate with the MQ EJB component that sent payment messages to the message queue. In the Consumer Banking Plus application, this logic is moved into a business method of the BankTasks Session bean, providing one more level of integration with the back-end MQ application. Where the topology 5 Mortgage Payment System application provides a simple Web interface to the legacy application from the Command bean, the topology 6 implementation further encapsulates the integration with the MQ EJB component inside of the main application object.

## **6.9.2 Implementing the Consumer Banking Plus application**

The Consumer Banking Plus topology 6 implementation is the integration of the previous two applications - the original Consumer Banking application and the Mortgage Payment System. The BankTasks Session bean provides the point of integration between the Consumer Banking application and the MQ EJB component which provides the access to the legacy Mortgage Payment System MQ application. This integration essentially entails moving the same code that makes the calls to the MQ EJB component from the Command bean in the Mortgage Payment System application (as described in 6.8.3, “Implementing the Mortgage Payment System” on page 139) to the model of the Consumer Banking application. Specifically, the only modifications to the BankTasks bean occur in the transfer() method, shown in the following figures.



```

public void transfer(String fromAcctId, String toAcctId, double amount)
throws javax.naming.NamingException, java.rmi.RemoteException,
BankTransactionException, FinderException {

    try {
        BankAccountAccessBean fromba = lookupAccount(fromAcctId);
        BankAccountAccessBean toba = lookupAccount(toAcctId);

        fromba.withdraw(amount);

        java.sql.Timestamp ts = new
        java.sql.Timestamp(System.currentTimeMillis());
        String id = ts.toString();

        getTranRecordHome().create(id, amount, 'D', ((BankAccountKey) fromba.getEJB
        Ref().getPrimaryKey()).accountId, ((BankAccountKey)
        toba.getEJBRef().getPrimaryKey()).accountId);

        // handle transfers to Mortgage accounts by sending a message to the
        MQ-backed MPSOutbound EJB
        if (toba.getAccountType().trim().equalsIgnoreCase("MORTGAGE")) {

            String homeName = "mps/mq/MPSOutboundEJBHome";

            mps.mq.MPSOutboundAccessBean mpsOutbound = new
            mps.mq.MPSOutboundAccessBean();
            mpsOutbound.setInit_JNDIName(homeName);

            // create mpsOutbound message template
            String queueName = "MPS.QUEUE";
            String paymentAmount = String.valueOf(amount);

            mps.mq.MPSMsgTemplate template = new mps.mq.MPSMsgTemplate();
            template.setQueueName(queueName);
            template.setAccountId(toAcctId);
            template.setPaymentAmount(paymentAmount);

```

Figure 58. BankTasks transfer() method

```

        // put the message
        String correlatorId = mpsOutbound.put(template);
    }

    toba.deposit(amount);

    ts = new java.sql.Timestamp(System.currentTimeMillis());
    id = ts.toString();

    getTranRecordHome().create(id, amount, 'C', ((BankAccountKey)toba.getEJBRef().getPrimaryKey()).accountId, ((BankAccountKey)fromba.getEJBRef().getPrimaryKey()).accountId);

    } catch (BankTransactionException bte) {
        throw new BankTransactionException(bte.getMessage());
    } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
    }
    // handle CB-related exceptions from the MPSOutbound component
    catch (com.ibm.IMessageHome.IMessagePutFailed e) {
        throw new BankTransactionException(e.getMessage());
    } catch (com.ibm.IManagedClient.IInvalidKey e) {
        throw new BankTransactionException(e.getMessage());
    } catch (com.ibm.IManagedClient.IInvalidCopy e) {
        throw new BankTransactionException(e.getMessage());
    }
}

```

Figure 59. BankTasks transfer() method - continued

The transfer() method simply checks if the “to” account is a “MORTGAGE” account, and if so, sends a message to the MQ EJB component and credits the BankAccount Entity bean that stores the current account balance. When a customer transfers money from one account (either savings or checking) to a mortgage account, meaning they want to make a payment on their mortgage, then what occurs is a two-phase commit across two different tables of a DB2 database (ibmwebs.account and ibmwebs.transrecord) and a “put” onto an MQ queue.

The flow that occurs all within the context of a transaction is:

1. Debit/update the “from” account (ibmwebs.account DB2 table)
2. Create transaction record (ibmwebs.transrecord DB2 table)

3. Send message to MQ EJB component (put to MQ queue)
4. Credit/update "to" account (ibmwebs.account DB2 table)
5. Create transaction record (ibmwebs.transrecord DB2 table)

The BankTasks Session bean as deployed in the Consumer Banking application has a transaction policy of "TRANSACTION\_REQUIRED" meaning that if the client of this bean does not begin a transaction, then the container (on the server) will wrap a transaction around method calls on that bean. Notice that in the transfer method implementation, we do not explicitly begin and end a transaction. This is because in our topology 6 implementation, the container controls this two-phase commit transaction across the multiple back-ends. So, say you have only \$1000 in your savings account, but attempt to move \$3000 from that savings account to the mortgage account, then during the processing of the transfer() method of the BankTasks, a BankTransactionException will get thrown. At that point, the container will roll back the transaction, no updates will be made to the DB2 database, no message will get sent to MQ, and an exception will get thrown back to the servlet, and as a result the results page that gets displayed will not show any updates to the account balances.

Now what would really show the rollback of this two-phase commit transaction coordination is if something went wrong during the processing of the MQ EJB component sending the message to MQ. Then an exception will get thrown in the middle of the transaction - updates to the "from" account and the transaction record have already occurred. So say for some reason the MQ process is not running. Then when the MQ EJB component tries to put the message on the queue, the Component Broker MQSeries Application Adaptor will throw an exception because it can't connect to the queue manager, and the updates/inserts that occurred with the "from" account and transaction record in the DB2 database will get rolled back.

Other transaction policies on the BankTasks bean could drive differences in the code. For example, if transaction attribute were set to "TX\_BEAN\_MANAGED", then inside the transfer() method, we would have to begin and commit (or roll back) the transaction explicitly. Or, if the transaction attribute were set to "TX\_MANDATORY", the client of BankTasks would be required to start a transaction before making the call to transfer(). In this case, the TransferAccount servlet would have to explicitly begin and end the transaction.

---

## 6.10 Other design considerations

For more Web application design considerations, including application security and application session management, refer to *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864. Some topics, such as Command beans, are covered in more detail there by showing examples.





## Chapter 7. Component Broker introduction

In this chapter, you will find a quick introduction to IBM's Component Broker. More information can be found in two redbooks:

- *IBM Component Broker Connector Overview*, SG24-2022
- *WebSphere Application Server Enterprise Edition Component Broker 3.0 First Steps*, SG24-2033

Both of these books give more detailed insight into Component Broker, its structure, and application design considerations. We are including this short introduction for those who are unfamiliar with Component Broker and want to continue on to the examples.

### 7.1 What is CBBConnector?

IBM's Component Broker is an integrated package that allows you to develop, deploy, run, and manage a new generation of distributed, multi-tier, object-oriented applications. Its major components are shown in Figure 60.

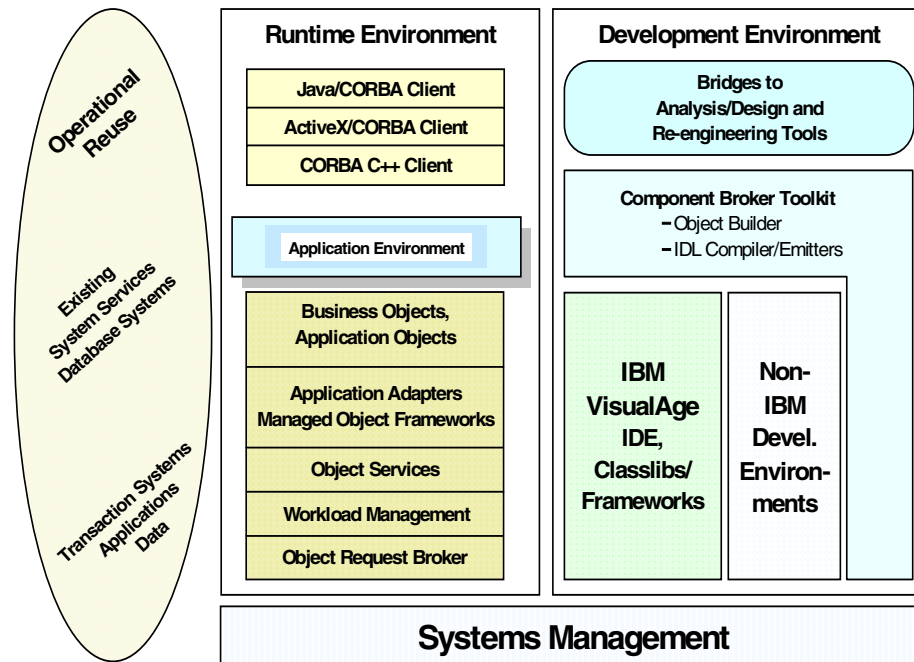


Figure 60. Component Broker building blocks

Component Broker is composed of an industry-leading set of technologies that facilitate distributed object applications. It combines three critical dimensions shown in the figure above:

- Runtime
- Development
- System Management

Component Broker consists of two parts that support these three dimensions. The Component Broker (CBConnector) provides the runtime environment and supports systems management. The Component Broker Toolkit (CBToolkit) contains the tools that application developers use to define and implement the objects running on the middle-tier.

The basics of the runtime dimension were discussed earlier when we described the integration server technology options (see 5.3, “Integration Server” on page 60).

### 7.1.1 Development

Some of the prominent components of the CBConnector development environment are depicted on the right side of Figure 60 on page 153. Of the tools within that environment, the *Object Builder* is the major tool you use to define the structure and CBConnector-specific interfaces of your objects. Using a set of wizards, it guides you through defining and building all the constructs necessary in the CBConnector environment. Object Builder generates and helps you build the scaffolding needed to create deployable install images for servers and clients.

The Object Builder also has a bridging facility to outside analysis and design tools. This allows you to bring your analysis models into the CBConnector world and then extend them for CBConnector as appropriate. You can also import Interface Definition Language (IDL) files or RDBMS Data Definition Language (DDL) files to begin re-engineering your existing applications and databases.

As far as complementary development environments are concerned, the IBM VisualAge family is an obvious choice for Business Object development. For clients, you can continue to use your favorite platforms.

### 7.1.2 System management

To start with, you use CBConnector's system-management facilities to define the network of host machines, and their configurations of servers and application software to parameterize the runtime environment.



At install time, you use its facilities to introduce your applications into the picture, deploying them within those configurations, as needed. At run time, you control the environment, bringing up servers, enabling or disabling applications, and so forth. Rather than being grafted on top as an afterthought, systems management has been designed into CBCConnector from the start. For example, this has been done by instrumenting the code with the appropriate hooks for controlling the system and collecting management information from it. There is a short overview of the facilities built into CBCConnector Release 1 in the redbook, *IBM Component Broker Connector Overview*, SG24-2022.

---

## 7.2 CBCConnector's focus areas

Essentially, CBCConnector is a platform for developing a new generation of flexible software, built using distributed objects technology. It is an ideal base for implementing thin-client topologies, cleanly separating the client-side concerns of ever changing user interface technologies from the very different characteristics of the code needed to support business logic. The Internet is, of course, the most prominent example. CBCConnector is very well suited to that dynamic environment.

However, one main focus area is that of the *operational reuse* of existing code and databases. Talking about CBCConnector to one large customer in the insurance industry lately, one of the main things they were enthusiastic about was CBCConnector's potential for reusing their legacy code. Specifically, they estimate their investment in domain-specific, home-grown software to be approximately ten times their present manpower capacity. In their view, it is ludicrous to even think about throwing all that away and starting any major system from scratch. IBM research has shown this situation to be more the norm than the exception with Fortune 500 companies.

Another typical requirement today is that of integrating separately developed application subsystems. This situation is aggravated by the surge in mergers and acquisitions we are experiencing. Through its middle-tier Business Objects application server, Component Broker offers an integrating platform that can bring together hardware and software systems that were developed without too much concern for interoperation.

---

## 7.3 Component Broker – a member of the Transaction Series family

Component Broker does not stand alone, but rather cooperates with existing transaction and resource managers. It functions as an object server by providing an application environment that lets clients work with mature

back-end systems through object-oriented middleware. It cooperates fully with IBM's industry-leading family of transaction servers.

CBConnector incorporates proven technology from other IBM middleware products such as CICS/ESA, Transaction Series (Distributed CICS and Encina) as well as OSF/DCE. Over time, IBM intends to add Component Broker technology to Transaction Series so that these existing, proven platforms will also benefit from the new computing model.

Future offerings will expand support for other enterprise systems (for instance, IMS) and clients (such as Smalltalk). Further integration with systems management environments, such as the Tivoli Management Environment (TME), is planned. Extensions to Component Broker and corresponding extensions to the *Shareable Frameworks Project* will enable Component Broker and Java/AS400 business frameworks to work together.

Component Broker development spans IBM teams and laboratories. This ensures that the whole product will take advantage of the diverse experiences and many disciplines required to build an integrated solution. We feel strongly that Component Broker is a third-generation object-technology offering. Component Broker's design itself is object-oriented and CORBA-compliant. The ultimate goal is for it to work seamlessly with other CORBA-compliant products in the marketplace. It does so, recognizing that – given today's complex de-facto situation – the best way forward is one of evolution.

---

## Chapter 8. U2B topology 5 and 6 implementation

Part of this project involved “proving” the runtime topologies detailed in Chapter 3, “Choosing the runtime topology” on page 23. This meant setting up these environments in the lab and determining the best product levels to run in each server.

In the course of writing this book, we used a sample application provided by IBM, called the WebSphere Family Sample application. This application illustrates, among other things, the use of WebSphere as a Web application server that calls EJBs residing on Component Broker. Our intent is to give the details on how our lab environment was set up using a modified version of the WebSphere Family Sample application to illustrate the User-to-Business topologies 5 and 6.

These examples are based on Windows NT.

---

### 8.1 Family Sample

The WebSphere Family Sample application is the basis for the examples in these scenarios. The application can be downloaded from:

<http://www.software.ibm.com/websphere/samples/>

The Family Sample application, as downloaded from the Web site, is an online banking application. It allows the users (bank customers) to list their accounts, display their account balances, and transfer money between accounts. The back-end banking data is stored in a DB2 database.

The application can be implemented in four ways:

- EJBs deployed in WebSphere Advanced application server
- EJBs deployed in Component Broker
- Java business objects deployed in Component Broker
- EJBs that communicate with TXSeries Encina

The focus of this book is on the second option, with the EJBs deployed in Component Broker.

#### 8.1.1 Modified version

We took this application and modified it to demonstrate the topology 5 and topology 6 scenarios. The modifications are described in Chapter 6, “Application design guidelines” on page 73. The modified version includes the

original Family Sample application with a few minor changes and a new Mortgage application. This new version of the application has:

- The original Family Sample (CB/EJB) option that is a stand-alone DB2 based banking application showing a flow from browser->jsp/servlet->EJB->DB2.
- A self-contained MQSeries application that illustrates topology 5 by showing a flow from browser->jsp/servlet->EJB ->MQ->MQ application.
- An application that integrates the work of the previous two applications, illustrating topology 6 showing a flow of browser->jsp/servlet->EJBs (from Family Sample)->EJB (from Mortgage application)->MQ->MQ application (mortgage payment system).

## 8.2 Basic topology

The first lab setup is an implementation of the basic topology. This setup has WebSphere advanced in the DMZ. Component Broker is in the secure network.

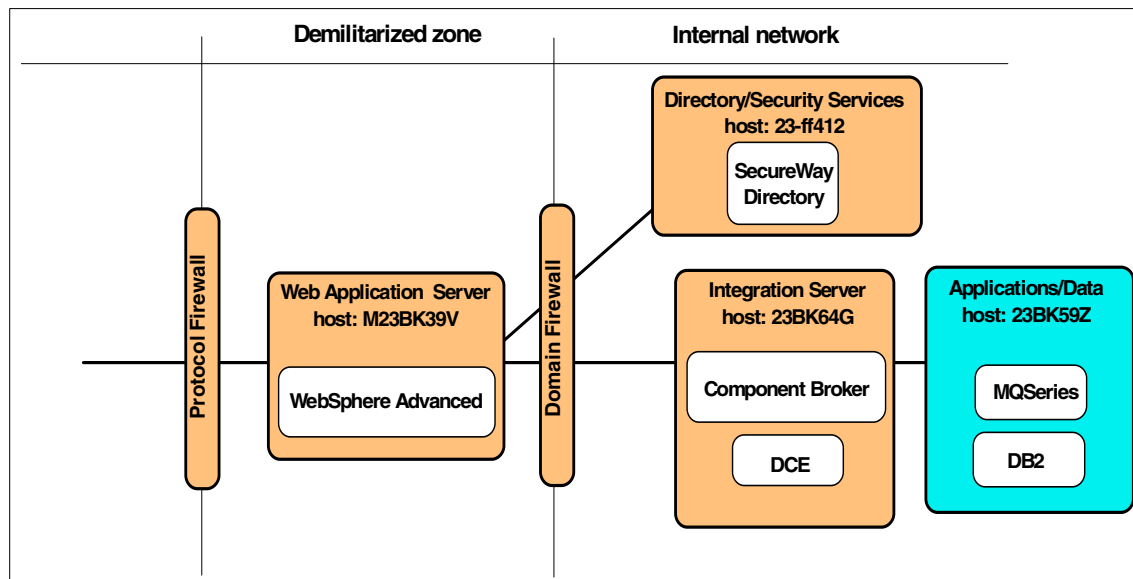


Figure 61. Basic topology

The products installed to support this implementation are listed in the following table.

| Product Installed  | Instructions   |
|--|--|
| <b>Web application server (hostname: M23BK39V)</b><br>Documentation for WebSphere can be found at<br><a href="http://www.ibm.com/software/webservers/appserv/library.html">http://www.ibm.com/software/webservers/appserv/library.html</a> .<br><br>Choose WebSphere Advanced Server 3.0x. For installation, see <i>Getting Started, and Installation Guide</i> .  |  |
| Windows NT 4.0 + SP4   |  |
| JDK 1.1.7<br>(IBM build n1.1.7p<br>19990823)   | Installation instructions are in the <i>Installation Guide</i> .<br>After installation be sure to set up the path and classpath system environment variables:<br><br>Add "c:\jdk1.1.7\lib\classes.zip" to the CLASSPATH variable.<br><br>Add "c:\jdk1.1.7\bin" to the path variable. |
| DB2 5.2 client fixpack 11  | DB2 is needed for the WebSphere administrative repository and for the Family Sample data base. They can be local to WebSphere (install DB2 Enterprise Edition server) or remote (install DB2 Client Application Enabler. In this case, DB2 CAE was used.                             |
| IBM HTTP Server 1.3.6  | Installation instructions are in the <i>Installation Guide</i> .   |
| WebSphere Advanced 3.021   | From the Install Options window, select <b>Custom Installation</b> and select:<br>- Production Application Server<br>- Administrator's Console<br>- Documentation<br>- Samples<br>- Configure admin domain with the default applications<br>- The appropriate Web server plug-in.    |
| Family Sample  | See 10.2, "Installing Family Samples on WebSphere Advanced Edition server" on page 180.  |
| <b>Integration Server (hostname: 23BK64G)</b><br>Documentation for Component Broker can be found at<br><a href="http://www.ibm.com/software/webservers/appserv/library.html">http://www.ibm.com/software/webservers/appserv/library.html</a> .<br><br>Choose WebSphere Enterprise Edition 3.0x and then go to the Component Broker online library. For installation, see the <i>Component Broker Planning, Performance, and Installation Guide</i> . |  |
| Windows NT 4.0 + SP4   |  |

| Product Installed                            | Instructions   |
|--|--|
| (IBM build n1.1.7p 19990823)                 | This is the same as for WebSphere Advanced. Installation instructions can be found in <i>Component Broker Planning, Performance, and Installation Guide</i> .<br><b>Note:</b> It is very important that you have the correct JDK version. Even a difference in date on JDK n117p can cause problems. |
| IBM HTTP Server 1..3.6                       | Not needed for function, but to resolve “skit.dll” error messages.   |
| VisualAge for C++ V3.5.7                     | For C++ development and installation prereq checker.<br><br>See <i>Component Broker Planning, Performance, and Installation Guide</i> . We chose the “typical” install and took all installation defaults.   |
| VisualAge for Java V3.02                     | See <i>Component Broker Planning, Performance, and Installation Guide</i> . We took all installation defaults.   |
| DB2 5.2 CAE (or DB2 5.2 server + FixPack 11) | Install instructions in the <i>Component Broker Planning, Performance, and Installation Guide</i> . Fixpack 8 would work also, but we chose to go with the later version for consistency with WebSphere Advanced.  |
| DCE V2.2 Server                              | Required for Component Brokers naming service.<br><br>See 9.1, “Installing and configuring DCE” on page 165.   |
| Component Broker V3.02                       | See <i>Component Broker Planning, Performance, and Installation Guide</i> for more information.<br><br>Chapter 9, “Setting up Component Broker” on page 165.   |
| MQSeries 5.1 client                          |  |
| MQSeries Application Adaptor                 | See the <i>MQSeries Application Adaptor Quick Beginnings</i> , SC09-4410 for installation instructions. The adaptor is provided with WebSphere Enterprise Edition.   |
| Family Sample                                | See 10.3, “Installing Family Samples on WebSphere Enterprise Edition (CB)” on page 185<br><br>See the readme file that comes with the Family Sample application for more information.  |
| <b>Applications node (hostname: 23BK59Z)</b> |  |

| Product Installed                    | Instructions  |
|--------------------------------------|---|
| Windows NT 4.0 + SP4                 |   |
| DB2 5.2 Server + fixpack 11          | Install DB2 Enterprise Edition server.<br><br>DB2 is needed for the WebSphere administrative repository, the Family Sample data base, and the CB databases. We chose to have one DB2 server set up on a separate machine and installed the DB2 CAE client software on the WebSphere Advanced and CB machines. |
| MQSeries V5.1 Server                 |   |
| <b>Firewalls</b>                     |   |
| Windows NT 4.0 + SP4                 |   |
| IBM SecureWay Firewall 4.1           | The firewalls are described in Chapter 13, "Network security" on page 249. The rules required for this setup are described in 13.6.1, "Base topology" on page 254.  |
| <b>Directory / Security services</b> |   |
| Windows NT 4.0 + SP4                 |   |
| IBM SecureWay Directory 3.1.1        | The configuration is covered in <i>Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition</i> , SG24-5864.  |

### 8.3 Second topology

This scenario uses a servlet redirector in the DMZ to send requests to WebSphere. Both WebSphere Advanced and Component Broker are in the secure network.

The main difference in the installation of this scenario from the first (besides network positioning) is that we need a third machine to act as a redirector, which will be implemented using the Remote OSE feature of WebSphere Advanced. Using a redirector means that the WebSphere Advanced machine no longer needs an HTTP server installed on it. The firewall rules will also change.

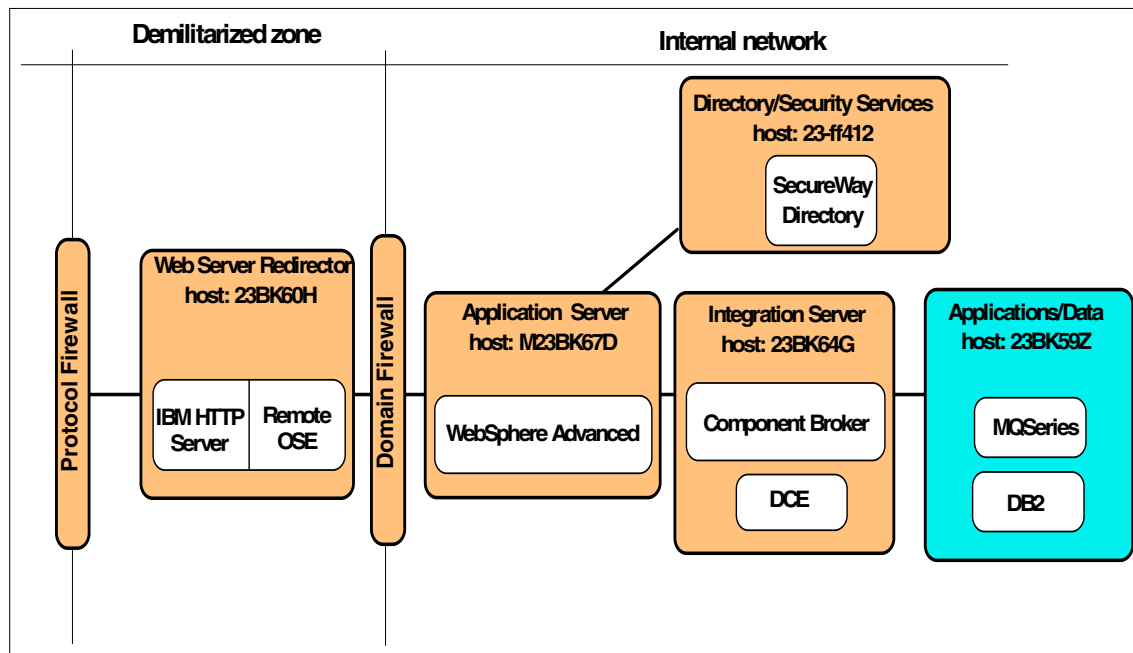


Figure 62. 2nd topology

The products installed are listed in the following table:

| Product Installed   | Instructions   |
|---|--|
| <b>Web server redirector (hostname: 23bk60h)</b><br>Documentation for WebSphere Advanced Server 3.0x can be found at<br><a href="http://www.ibm.com/software/webserver/appserv/library.html">http://www.ibm.com/software/webserver/appserv/library.html</a> . |  |
| Windows NT 4.0 + SP4  |  |
| JDK 1.1.7 IBM build<br>n117p 19990823   | Installation instructions are in the <i>Installation Guide</i> .<br>After installation be sure to set up the path and classpath system environment variables:<br><br>Add "c:\jdk1.1.7\lib\classes.zip" to the CLASSPATH variable.<br><br>Add "c:\jdk1.1.7\bin" to the path variable. |
| IBM HTTP Server 1.3.6   | Installation instructions are in the <i>Installation Guide</i> .   |



| Product Installed   | Instructions   |
|---|--|
| WebSphere Advanced configured for Remote OSE  | See Chapter 11, "Redirecting using OSE Remote" on page 227.  |
| <b>Application server (hostname: M23BK67D)</b><br>Same as the Web application server in the basic topology. The HTTP server is optional, and if you don't have it installed, you do not need to install the plug-in during WebSphere Advanced installation. For testing purposes, we found it was handy to have a Web server installed. |  |
| <b>Integration Server (hostname: 23BK64G)</b><br>Same as in the basic topology.   |  |
| <b>Firewalls</b>  |  |
| Windows NT 4.0 + SP4  |  |
| IBM SecureWay Firewall 4.1  | The firewalls are described in Chapter 13, "Network security" on page 249. The rules required for this setup are described in 13.6.2, "Variation 1" on page 261. |
| <b>Directory / Security services</b><br>Same as in the basic topology   |  |

---

## 8.4 User IDs used in the installation

For development and testing purposes, it is easy to fall into the trap of using one administrative user ID for everything. While this can be very convenient, it is generally a bad idea. For this reason we are listing the user IDs and capabilities used for these lab exercises.

`pattern` - an account with system administrator capabilities defined at every Windows NT node to be used to install our applications

`dbAdm` - an account with user capabilities at the back-end server to administer the DB2 Server

`wsAdm` - an account with user capabilities at the back-end server to own the WAS and Component Broker databases

`cbnkAdm` - an account with user capabilities at the back-end server to own the ConsumerBank databases

`cbnkUsr` - an account with user capabilities at the back-end server to use the ConsumerBank databases

`wsAdm` - an account with user capabilities at the application server to execute the Websphere Advanced application server

`wsAdm` - an account with user capabilities at the integration server to execute the WebSphere Enterprise Application Server and our DCE applications, servers and clients;

---

## Chapter 9. Setting up Component Broker

This chapter takes you through the steps we followed to install and configure the DCE and Component Broker software. DCE and Component Broker should be the last software installed on the system. The complete list of products installed on this system are shown in 8.2, “Basic topology” on page 158.

More complete instructions can be found in the *Component Broker Planning, Performance, and Installation Guide*. We are including this information so there is no confusion about which options we chose to make these scenarios work.

For more information on the user IDs used in the scenarios, see 8.4, “User IDs used in the installation” on page 163.

---

### 9.1 Installing and configuring DCE

Component Broker's naming service is implemented using Distributed Computing Environment (DCE) Cell Directory Services (DCS). There is one DCE cell defined for a Component Broker System Manager and all the application servers it manages.

DCE can be installed locally or remote.

#### 9.1.1 Installing the DCE server or client

The install is done using a user ID with administrative authority. In our scenarios, we used the user ID of “pattern”.

To begin the install, insert the IBM DCE for Windows NT Version 2.2 with the data encryption standard (DES) CD. Double-click setup.exe and follow the installation path. During the installation we made the following selections.

1. Take the standard install.
2. **For a DCE server installation** select the following components:
  - DCE Runtime Services
  - DCE Cell Directory Server (CDS)
  - DCE Security Server (SS)
  - Additional Documentation

**For a DCE client installation** select the following component:

- DCE Runtime Services

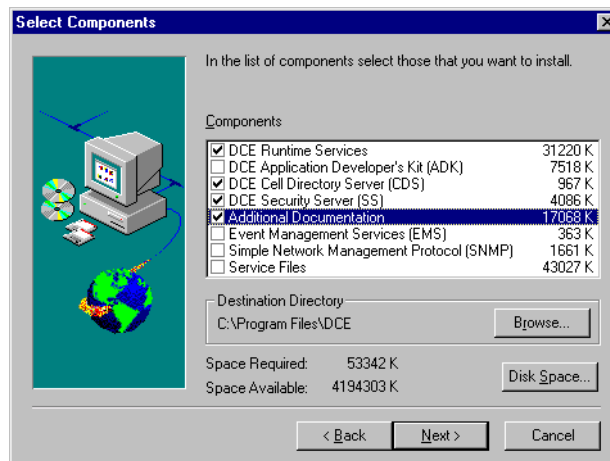


Figure 63. DCE server install

3. We took the default path of C:\DCE22.
4. We chose "English in US OEM CP" for the cultural convention.

After the installation completes, the system will reboot.

### 9.1.2 DCE server configuration

First you need to create a user ID for use with DCE. For our example, we used "wsAdm", which has user (not administrative) capabilities. You do not need to switch to this ID. We stayed logged on as the administrative ID, "pattern", for the DCE configuration.

1. Start DCEsetup by clicking **Start->Programs->DCE for Windows NT V2.2->DCEsetup**.
2. Start the server configuration process by clicking **Configuration->Create->Server**.
3. From the Server Configuration Wizard fill in the DCE cell and principal account information. We use the following values:
  - Cell name: ConsBank
  - Principal Account Name: wsAdm
  - Principal password: wsWrd

**Note:** the cell name must be unique in your network. If you choose a name that is already in use, the configuration will not be successful.

Server Configuration Wizard

DCE Host name: 23bk64g.itso.ral.ibm.com Cell name: ConsBank

☒ Directory Server  
☒ Master  
☐ Replica

☒ Security Server  
☒ Master  
☐ Replica  
 Server name:

☒ Distributed Time Service  
☐ DTS Client  
☒ DTS Local Server  
☐ DTS Global Server

☐ Cell spans multiple LANs LAN Profile name:

Principal Account Name wsAdm  
 Principal Password  
 Confirm Password

< Back Next > Cancel Help

Figure 64. DCE configuration - panel 2

4. From the Server Configuration Wizard:

- Select whether to auto-start DCE services at system startup or not. If you choose to auto-start the services, check **Clean before Auto Start**. If you choose not to auto-start the services, you will need to start DCE manually before starting Component Broker. This option can be reconfigured later.
- Disable Integrated Login.
- Click **Next**.

Server Configuration Wizard

Security Server Master Host Name or IP Address

Directory Server Host Name or IP Address

☐ Automatically sync local time with cell time  
Preferred DTS Server to Sync with

☐ Auto Start DCE services at system startup ☐ Clean before Auto Start

| Enable                | Disable                          |                                |
|-----------------------|----------------------------------|--------------------------------|
| <input type="radio"/> | <input checked="" type="radio"/> | Integrated Login               |
| <input type="radio"/> | <input checked="" type="radio"/> | Password Strength Server       |
| <input type="radio"/> | <input checked="" type="radio"/> | Audit Server                   |
| <input type="radio"/> | <input checked="" type="radio"/> | Name Service Interface Gateway |
| <input type="radio"/> | <input checked="" type="radio"/> | Global Directory Agent         |
| <input type="radio"/> | <input checked="" type="radio"/> | Identity Mapping Server        |
| <input type="radio"/> | <input checked="" type="radio"/> | Event Management Service       |
| <input type="radio"/> | <input checked="" type="radio"/> | DCE SNMP                       |

☐ Certificate Based Login

Entrust Initialization File

Entrust Profile for the DCE Security Server

Entrust Profile Password

< Back Next > Cancel Help

Figure 65. DCE installation - panel 3

5. In the next window, click **Finish**. The messages generated by the configuration will show up in the left panel of the DCEsetup window. If there are no errors, you should see something like the messages in Figure 66.

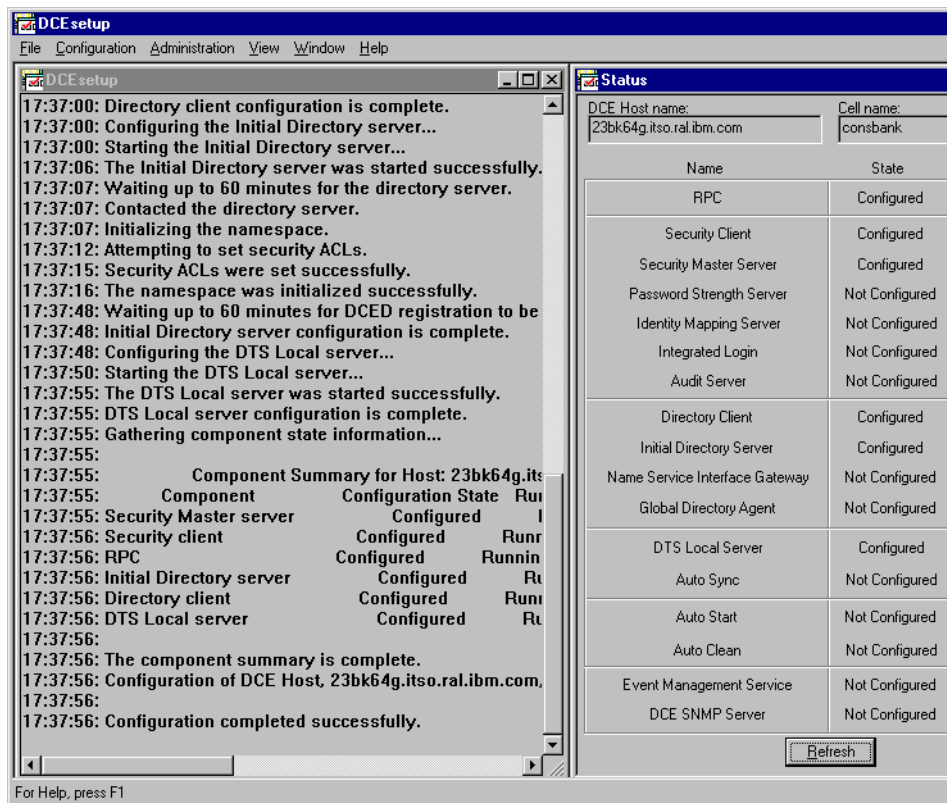


Figure 66. DCE client configuration

6. Close the DCEsetup window.

7. Open a Windows NT command prompt to log in to DCE. Enter:

```
dce_login <principal_account_name principal_id>
```

In our case this would be:

```
dce_login wsAdm wsWrd
```

8. You will see a message indicating the password should be changed. Issue the following commands to modify the password validity.

```
dcecp
```

```
account modify wsAdm - pwdvalid yes
```

```
exit
```

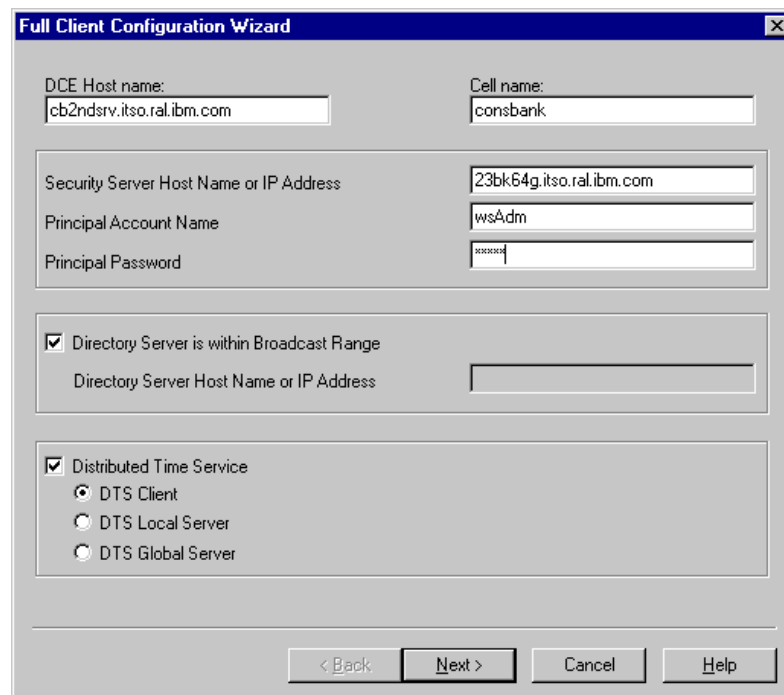
### 9.1.3 DCE client configuration

If the DCE server is not on the Component Broker machine, you will need to install the DCE client.

First you need to create a user ID for use with DCE. For our example, we use “wsAdm” which has user (not administrative) authority. You do not need to switch to this ID. We stayed logged on as “pattern” for the DCE configuration.

1. Start DCEsetup by clicking **Start->Programs->DCE for Windows NT V2.2->DCEsetup**.
2. Start the server configuration process by clicking **Configuration->Create->Full Client**.
3. From the Server Configuration Wizard, set
  - Cell name: ConsBank
  - Security server name is the host name or IP address of the DCE server
  - Principal Account Name: wsAdm

**Note:** the cell name must be the name specified for the DCE server.



The image shows a Windows-style dialog box titled "Full Client Configuration Wizard". It contains several input fields and checkboxes. The "DCE Host name" field is filled with "cb2ndsrv.itso.ral.ibm.com". The "Cell name" field is filled with "consbank". The "Security Server Host Name or IP Address" field is filled with "23bk64g.itso.ral.ibm.com". The "Principal Account Name" field is filled with "wsAdm". The "Principal Password" field is filled with "wsadm". There are two checkboxes: "Directory Server is within Broadcast Range" (checked) and "Distributed Time Service" (checked). Under "Distributed Time Service", there are three radio buttons: "DTS Client" (selected), "DTS Local Server", and "DTS Global Server". At the bottom, there are four buttons: "< Back", "Next >", "Cancel", and "Help".

Figure 67. DCE client configuration



4. From the Server Configuration Wizard:

- Select **Automatically sync local time with cell time** and fill in the name of the DCE server. (**Note:** when we did not select this option, we got errors during the DCE configuration saying that the authorization as “wsAdm” failed.)
- Select **Auto Start DCE services at system startup** and check **Clean before Auto Start**.
- Disable Integrated Login.

Click **Next**.

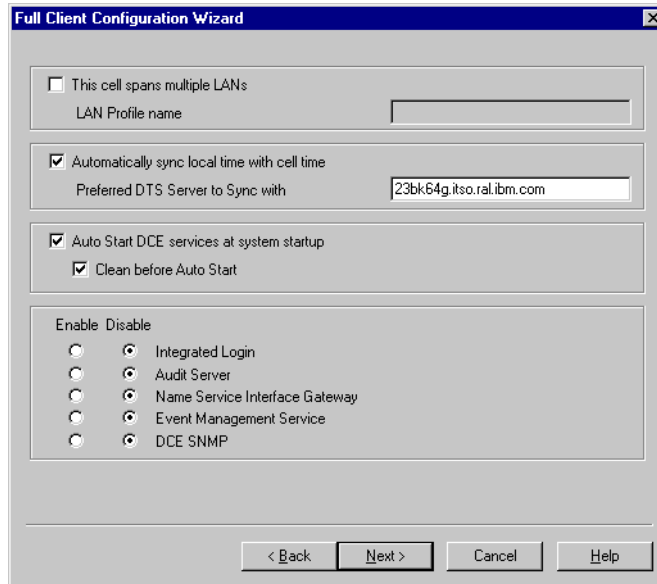


Figure 68. DCE client configuration - panel 2

5. In the next window, click **Finish**. The messages generated by the configuration will show up in the left panel of the DCEsetup window. If there are no errors, you should see something like the messages in Figure 66.

6. Close the DCEsetup window.

7. Open a Windows NT command prompt to log in to DCE. Enter:

```
dce_login <principal_account_name principal_id>
```

In our case this would be:

```
dce_login wsAdm wsWrd
```

If you see a message indicating the password should be changed. Issue the following commands to modify the password validity:

```
dcecp
account modify wsAdm - pwdvalid yes
exit
```

---

## 9.2 Component Broker installation and configuration

This section outlines the installation and configuration steps for the Component Broker (CB) server that will act as the system manager (CB SM) in our CB network.

The first step is to install the required software using the instructions in the *Component Broker Planning, Performance, and Installation Guide*. To begin, we defined a user ID called “pattern” on the system to have Component Broker installed. We gave the ID administrative authority and installed Component Broker while logged on as this ID.

In our lab we installed Component Broker in the following order:

1. Prereqs (see Chapter 8, “U2B topology 5 and 6 implementation” on page 157).
2. Component Broker 3.01 (this is the release that came in our package). The installation type was “System manager (First multi-host server)”.

**Note:** During the installation process the level of DB2 will be verified. The minimum prereq is Fixpack 8. If you have installed a higher level of Fixpack (we had Fixpack 11), click the **Yes** button to bypass the prereq.

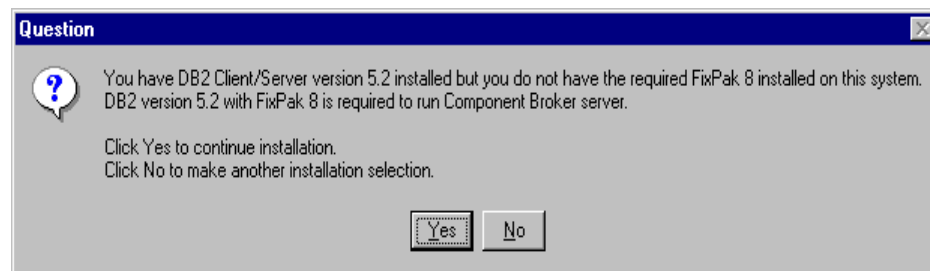


Figure 69. Component Broker installation warning message

3. Additional components from the Component Broker Tools CD. We installed the following:
  - Server SDK

- C++ Client SDK
  - Java Client SDK
  - CB Toolkit
  - Samples
  - MQSeries Application Adaptor SDK
4. Component Broker upgrade to 3.02. Even though 3.021 is available, the Family Samples were created for 3.02 and we found a few problems when we tried using them at 3.021. Upgrades for CB can be downloaded from <http://www.ibm.com/software/webservers/appserv/cb/support/>.
  5. Create the databases used by CB. If you installed DB2 server on the CB machine, this will be done for you during CB configuration. If the DB2 server is remote you will need to follow the instructions in the *Component Broker Planning, Performance, and Installation Guide*, to create the databases first.

We created a user ID for accessing the DB2 databases. We called the ID “wsAdm” and used the DB2 Control Center to give it “Grant All” access to the databases.

### 9.2.1 Configuring CB SM V3.0.2

Once CB is installed, you will need to go through a configuration process. The steps are:

1. Ensure the DCE has started. If you chose not to auto-start the processes during configuration, start DCEsetup and then start the DCE configuration by selecting **Administration -> Start**. After DCE is started, you can close the DCEsetup window.
2. From the Windows NT Start menu, select **Programs -> IBM Component Broker -> Component Broker Configuration Tool**.
3. On the first panel select **New Install Configuration**.
4. If you are installing Component Broker on a DB2 client machine you will be asked to enter the information required to find and access the Component Broker databases on the remote DB2 server. In our case, the DB2 server resides on a machine called 23bk59z.itso.ral.ibm.com. The four CB databases have been created on this server. BACKEND is the name to be used in cataloging the databases.

The screenshot shows a window titled "Component Broker Configuration Tool". Inside, there's a section titled "Enter Remote DB2 Server Information". Below this title is a instruction: "Enter your remote DB2 server host name, local node name, user id and password to access the remote DB2 server." There are five input fields with labels: "User ID" (value: wsAdm), "Password" (value: \*\*\*\*\*), "Server Host" (value: 23bk59z.itso.ral.ibm.com), "Local Node" (value: BACKEND), and "Service Name" (value: DB2). At the bottom of the window, there's a status bar that says "DB2 Client Setup Page" and a set of buttons: "<<Back", "Next>>", "Close", "Cancel", and "Help".

Figure 70. Remote DB2 access

5. Take the defaults in the next two windows.

**Note:** Sometimes it is desirable to install CB and WebSphere Advanced on the same machine, mainly for development and testing purposes. The only problem is that both products use port 900. Though both products can be configured to use alternate ports, we found it easier to change the port number here for CB.

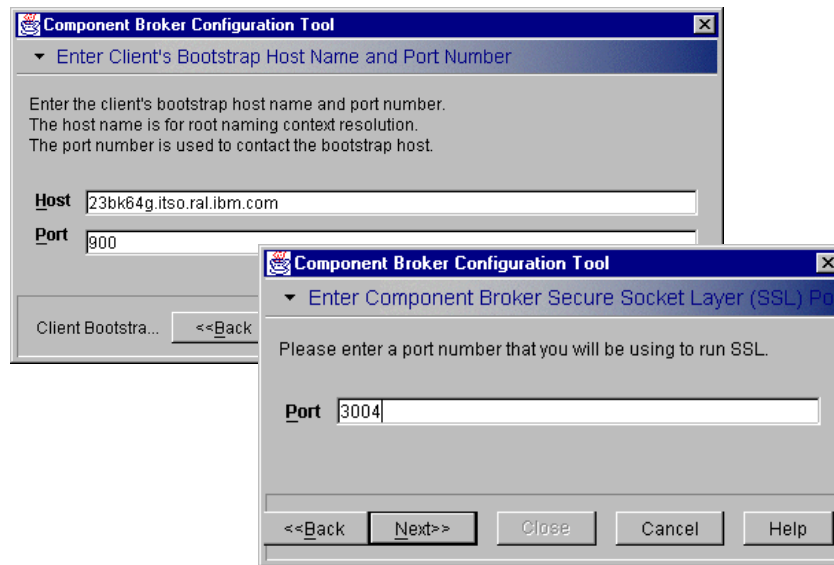


Figure 71. CB configuration

6. The next panel asks for the user ID to be used to start the CB service. We are using the administrative ID we are installing the product with (“pattern”). See the notes for choosing the user ID in the *Component Broker Planning, Performance, and Installation Guide*.

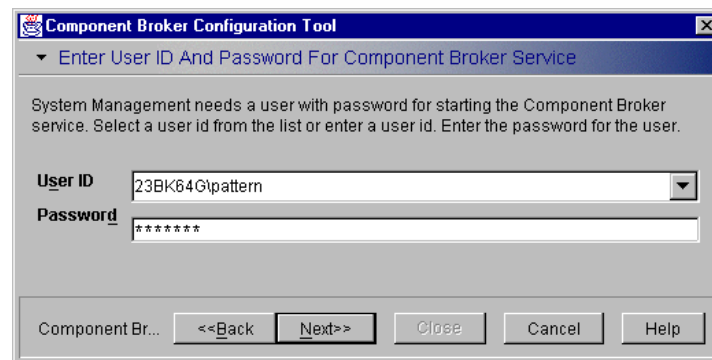


Figure 72. CB installation

7. The next window will ask for the user ID and password for the DCE cell administrator. During DCE installation, you had to specify a principle account name and password. These are the values you must enter here.

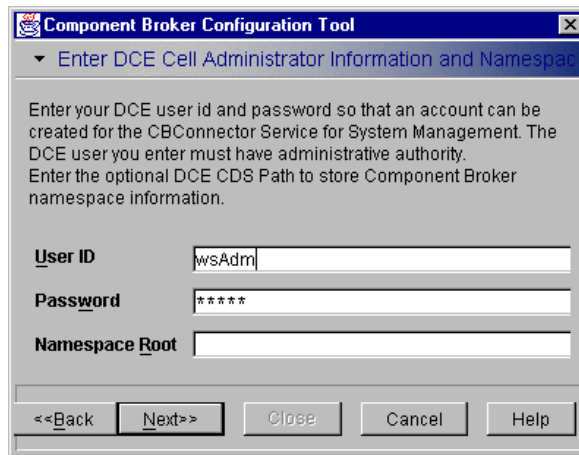


Figure 73. Specifying the DCE information

8. Since this is the System Manager (SM) machine, take the default host name and port number in the next window.

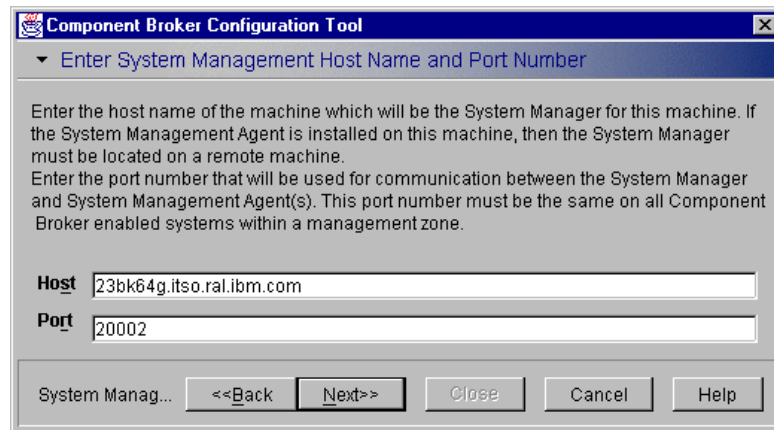


Figure 74. Specifying the System Manager information

9. The last window will allow you to verify your settings before proceeding with the configuration. Once you have verified that they are correct, the configuration will complete.

During the configuration, the DB2 databases will be created (if CB is on the DB2 server) and bindings will be done. Check the status of the configuration steps carefully. If there are any errors with the DB2

databases or bindings, see the *Component Broker Planning, Performance, and Installation Guide* for information on how to set up the DB2 databases and connections correctly.

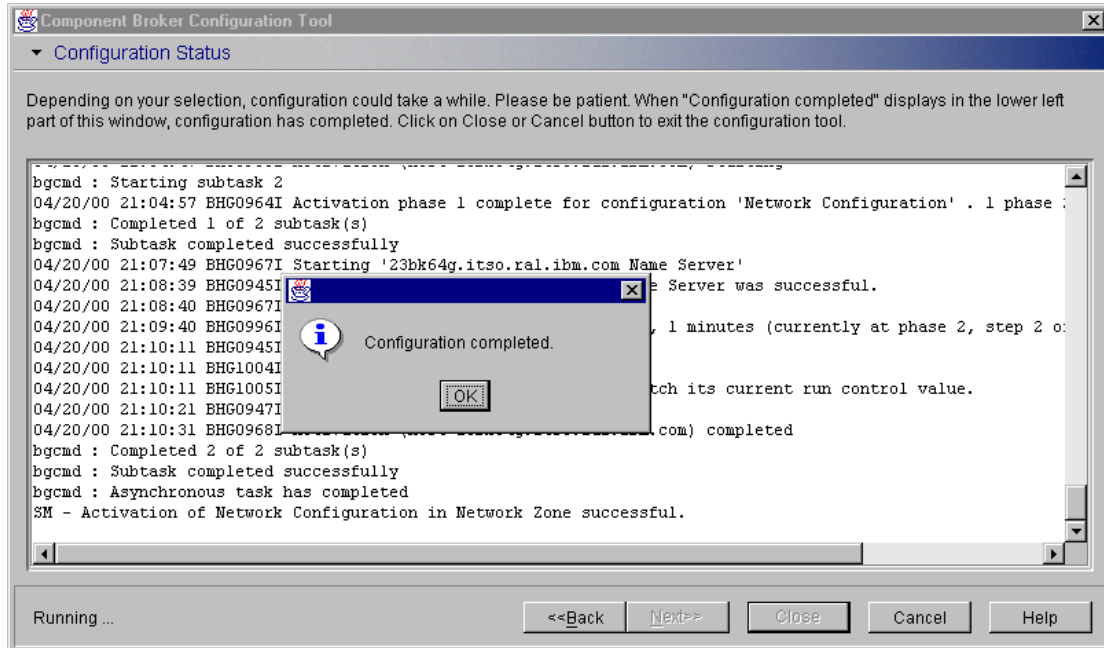


Figure 75. Component Broker configuration messages

10. Close the DB2 command windows and the Component Broker Configuration Tool window.

After Component Broker has been configured you can use the DCE Director to browse the DCE namespace. You will find entries for CB resources in the CDS.

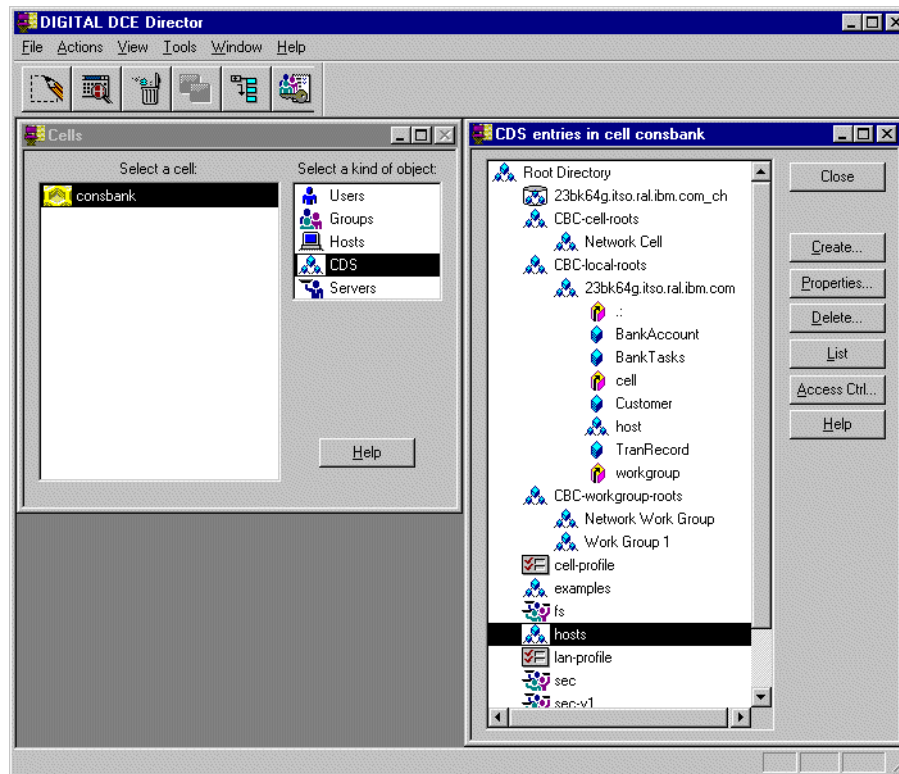


Figure 76. DCE CDS

### 9.3 Installing MQ Application Adaptor in CB

The MQSeries application adaptor provided by Component Broker is used to provide integration between CB business applications non-CB applications that are based directly on MQSeries.

The MQSeries application adaptor is provided on a separate CD along with the CICS and IMS, and Oracle adaptors in WebSphere Enterprise Edition. Follow the instructions in the *MQSeries Application Adaptor Quick Beginnings*, SC09-4410 to install and configure the adaptor.



---

## Chapter 10. Sample application implementation

Our scenarios are based on the IBM WebSphere Family Sample application. We describe this application as the Consumer Banking Plus application in Chapter 6, “Application design guidelines” on page 73.

The Web application that comes with the Family Sample is provided as a published version of a WebSphere Studio archive. The VisualAge for Java repository is also included. This Family Sample application can be downloaded from:

<http://www.ibm.com/software/webservers/samples/>

The Family Sample application comes with instructions that you should follow to install it. We will not provide the full details here, but will discuss them in general to give you an overview of the process. The idea is to introduce you to the implementation steps required for building a Web application using WebSphere Advanced and Component Broker.

We have taken the original Family Sample and modified and enhanced it slightly to illustrate both application topology 5 and topology 6. We did this by introducing integration with MQSeries. The second part of this chapter will take a look at the steps required to implement these enhancements. By reviewing these steps, you will see how to build an enterprise bean that communicates with MQSeries and how to deploy it in Component Broker.

---

### 10.1 Install the DB2 banking database

The Family Sample application accesses a user DB2 database called IBMWEBS that contains account data. This database can be local or remote. In the Family Sample application as shipped, both the WebSphere Advanced system and the Component Broker (CB) system must have access to this database. For our purposes, we are only illustrating the CB implementation and the only database access performed is from CB.

The application comes with a script to create and populate the database with banking information for three customers. In addition to the banking records, we added mortgage records to the database for the topology 6 scenario, by executing the following bat file:

```
rem This assumes you are running in
rem a db2 command line processor
call db2 -t -f mortgage.dml
```

Figure 77. *db2setup.bat*

The input SQL executed to add the mortgage records is shown in Figure 78.

```
CONNECT TO IBMWEBS;

INSERT INTO IBMWEBS.ACCOUNT (accid, custid, acctype, balance) VALUES
('11114000', '1111', 'MORTGAGE', -175000.00);
INSERT INTO IBMWEBS.ACCOUNT (accid, custid, acctype, balance) VALUES
('12344000', '1234', 'MORTGAGE', -100000.00);
INSERT INTO IBMWEBS.ACCOUNT (accid, custid, acctype, balance) VALUES
('22224000', '2222', 'MORTGAGE', -200000.00);

CONNECT RESET;
```

Figure 78. *mortgage.dml*

---

## 10.2 Installing Family Samples on WebSphere Advanced Edition server

The Family Sample application offers several implementation choices. The particular option we are interested in uses WebSphere Advanced Edition to run the servlets and JSPs. The servlets call EJBs on Component Broker.

To begin installation of the samples, the Family Sample zip file must be unzipped on the WebSphere Advanced machine.

**Note:** These steps include setting up WebSphere Advanced to use EJBs. This is one of the possible scenarios included with the samples. In our application, we will be using a different scenario offered with the samples, using EJBs on the Component Broker system instead. When we installed the samples, we did not strip out the WebSphere Advanced EJB functions, but rather left them in as a point of interest.

### 10.2.1 Initializing the application's properties file

WebSphere and Component Broker can be on the same machine or on separate machines. The application on WebSphere Advanced knows where the Component Broker system is by looking in a properties file that comes with the application. This file must be edited to indicate the host name and

the port number for the WebSphere Advanced system and the Component Broker system.

The properties file is in:

```
\family\website\FamilySample\servlets\WSFamily\FamilySampleStrings.properties
```

```
# This file specifies the hostnames and the port numbers for the
# different WebSphere hosts in the network.
# The default port number is 900.

# Uncomment the following lines and change the hostnames to define your
# WebSphere hosts. Use the fully-qualified name for the hostnames.
#   WSA.Host is where WebSphere Advanced Application Server resides.
#   WSE.Host is where WebSphere Enterprise (CB) EJB Server resides.
#   JBO.Host is where WebSphere Enterprise (CB) JBO Server resides.
#   ENC.Host is where WebSphere Advanced Application Server resides.

WSA.Host=m23bk67d.itso.ral.ibm.com
WSE.Host=23bk64g.itso.ral.ibm.com
JBO.Host=was1.sl.dfw.ibm.com
#ENC.Host=enc.austin.ibm.com
WSE.Port=900
WSA.Port=900
ENC.Port=900
JBO.Port=910
```

*Figure 79. Updating the FamilySampleStrings.properties file*

The port number for the WSE (WebSphere Enterprise, that is, Component Broker) machine must match the number set during CB configuration (see Figure 71 on page 175).

### 10.2.2 Defining the application to the WebSphere Application Server

The Family Sample application must be defined to WebSphere. This can be done manually using the WebSphere Administrative Console, or by using the WebSphere XMLConfig tool with the XML files shipped with the Family Samples. The Family Samples documentation provides instructions to do it either way.

The `ivjeb302.jar` file contains run-time Java classes to support Access beans. This file is provided with VisualAge for Java 3.02 and a copy is also supplied with the Family Sample application. Before deploying the EJBs in the WebSphere Advanced server, you will need to add the `ivjeb302.jar` file to the classpath in the XMLConfig utility, found in `\WebSphere\AppServer\bin\XMLConfig.bat`. This is required to support Access beans provided by VisualAge for Java as client-side EJB programming convenience classes.

```
@echo off
call setupCmdLine.bat

set WAS_CP=%WAS_HOME%\lib\ibmwebas.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\xml4j.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ujc.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ejb.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\console.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\admin.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\repository.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\sslight.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\properties
set WAS_CP=%WAS_CP%;%JAVA_HOME%\lib\classes.zip
set WAS_CP=%WAS_CP%;c:\family\vaj\ivjeb302.jar

%JAVA_HOME%\bin\java %CLIENTSAS% -classpath %WAS_CP%
com.ibm.websphere.xmlconfig.XMLConfig %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Figure 80. Updating the XMLConfig classpath

Once the XMLConfig tool is updated with the classpath, it can be used with the XML files to create the following definitions:

- The datasource and JDBC driver to access the database (IBMWEBS) that has the banking data.
- The definitions for the application.
- Add JAR files to the classpath for the application server.

#### 10.2.2.1 Define the DataSource

The `FamilyServerDataSource.xml` file will define and install a JDBC driver for DB2 access and a DataSource definition defining the sample database. The JDBC driver defines the class file to use for the database access. The DataSource defines the database name and the JDBC driver definition to use.

```

<websphere-sa-config>
  <jdbc-driver name="DB2 Driver" action="update">
    <implementation-class>com.ibm.db2.jdbc.app.DB2Driver</implementation-class>
    <url-prefix>jdbc:db2</url-prefix>
    <jta-enabled>>false</jta-enabled>
    <install-info>
      <node-name>m23bk67d</node-name>
    <jdbc-zipfile-location>c:\sqllib\sdsep$java$sdsep$db2java.zip</jdbc-zipfile-location>
    </install-info>
  </jdbc-driver>
  <data-source name="ibmwebs" action="update">
    <database-name>ibmwebs</database-name>
    <jdbc-driver-name>DB2 Driver</jdbc-driver-name>
    <minimum-pool-size>1</minimum-pool-size>
    <maximum-pool-size>30</maximum-pool-size>
    <connection-timeout>300</connection-timeout>
    <idle-timeout>1800</idle-timeout>
    <orphan-timeout>1800</orphan-timeout>
  </data-source>
</websphere-sa-config>

```

Figure 81. Defining the DataSource

You can confirm that the update was made by opening the WebSphere administrative console and switching to the Types tab. Click **JDBC** and you should see the new JDBC driver called DB2 Driver. Click **DataSource** and you should see the new DataSource called ibmwebs.

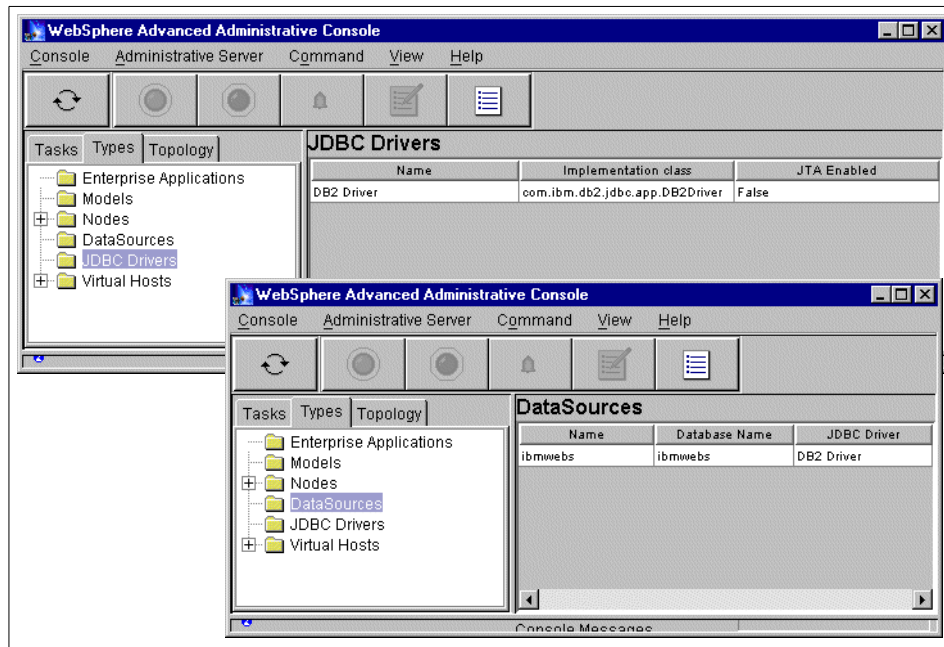


Figure 82. JDBC driver and DataSource

#### 10.2.2.2 Define the server

The FamilyServer.xml file creates the WebSphere Advanced definitions for the application. It creates a new application server, the servlets, and EJBs. Once the XML has run, you can go to the WebSphere administrative console and switch to the Topology tab. You should see the new application server called FamilyServer and the new definitions under it.

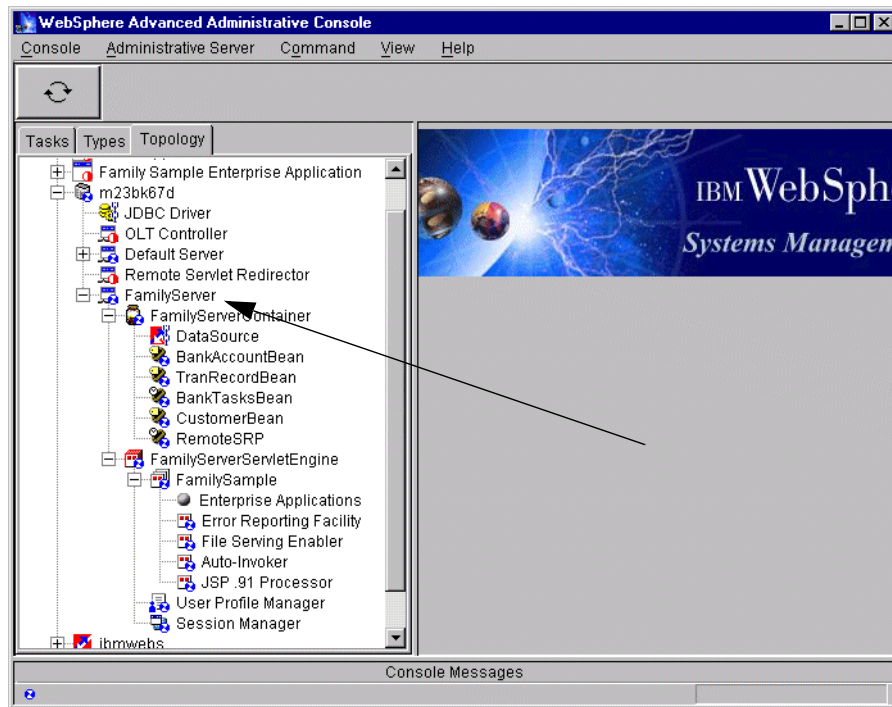


Figure 83. WebSphere Advanced Family Server definitions

#### 10.2.2.3 Extend the classpath for Component Broker

The last step in setting up the WebSphere Advanced system is to ensure the classpath for the application server is correct. The client ORB file, `somajor.zip`, needs to be copied from the directory `<CBroker>\lib` on the Component Broker machine to the `\family\lib` directory on the WebSphere Advanced machine.

The XML file adds `somajor.zip` and `jcbAccountC.jar` to the server class path. The `jcbAccountC.jar` file is for the Java BO (JBO) implementation (which we are not doing in this example). The `somajor.zip` file is required for communication to EJBs running inside of Component Broker.

### 10.3 Installing Family Samples on WebSphere Enterprise Edition (CB)

Setting up the definitions for the application on Component Broker is a little more involved. Assuming that Component Broker is already up and running, the first step is to unzip the Family Sample zip file onto the Component

Broker machine. A series of batch files are supplied to assist with the configuration.

### **10.3.1 Step 1 - generating the code**

The first batch file provided with the samples takes the VisualAge for Java exported JAR file and emits FinderHelpers. The Finder Helpers are required to support lookup commands in the homes for the BankAccount, Customer and TranRecord beans. After the FinderHelpers are emitted, the batch file updates the generated code, compiles the Java classes, and adds them to the ibmwebsEJBCB.jar file. It then invokes the cbejb tool. We found that we needed to make a few modifications to the batch file, which are shown in bold in Figure 84 and Figure 85.



```

setlocal

set FAMILY_DIR=c:\family

echo Delete the current ibmwebsEJBCB.jar, copy the
echo original one and then compile the finder classes

del /q ibmwebsEJBCB.jar

echo Copy the ibmwebsejb.jar to this directory
copy ..\..\vaj\ibmwebsejb.jar ibmwebsejbcj.jar

set
classpath=ibmwebsEJBCB.jar;%FAMILY_DIR%\vaj\ivjejb302.jar;%classpath%

echo Generate the helper classes
call ejbfhgen com.ibm.ibmwebs.sample.CustomerHome
call ejbfhgen com.ibm.ibmwebs.sample.TranRecordHome
call ejbfhgen com.ibm.ibmwebs.sample.BankAccountHome

echo At this time, modify the Customer, BankAccount and TranRecord
FinderHelpers
echo as described in the docs. When completed, press any key to
continue
echo with the compilation and zip

pause

echo Delete com directory
rmdir /s/q com
mkdir com\ibm\ibmwebs\sample

```

Figure 84. *ejb\_to\_cb batch file - part 1*

```

echo Copy java to subdirectories
copy *.java com\ibm\ibmwebs\sample
del *.java

set
classpath=%FAMILY_DIR%\cb\cdejb\ibmwebsEJBCB.jar;%FAMILY_DIR%\lib\somobj
or.zip;%FAMILY_DIR%\vaj\ibmwebsEJS.jar;%classpath%

echo Compile the helper classes
javac com\ibm\ibmwebs\sample\*.java
echo Zip up the new classes into the jar

REM zip -u -r ibmwebsEJBCB.jar com\ibm\ibmwebs\sample\*.class
jar -xf ibmwebsejbcj.jar
jar -cvfm ibmwebsejbcj.jar Meta-inf\Manifest.mf
com\ibm\ibmwebs\sample\*.class com\ibm\ibmwebs\sample\*.ser

echo Invoke the cbejb tool
echo When exiting OB, have the code emitted but do not start the build
call cbejb ibmwebsEJBCB.jar -ob . -cacheddb2
com.ibm.ibmwebs.sample.Customer:com.ibm.ibmwebs.sample.BankAccount:com.
ibm.ibmwebs.sample.TranRecord -queryable
com.ibm.ibmwebs.sample.Customer:com.ibm.ibmwebs.sample.BankAccount:com.
ibm.ibmwebs.sample.TranRecord -dllname Custome
com.ibm.ibmwebs.sample.Customer -dllname BankAcc
com.ibm.ibmwebs.sample.BankAccount -dllname TranRec
com.ibm.ibmwebs.sample.TranRecord -serverdep ..\..\vaj\ivjejb302.jar
-clientdep ..\..\vaj\ivjejb302.jar
copy *.hpp working\nt
pause
endlocal

```

Figure 85. *ejb\_to\_cb batch file - part 2*

### 10.3.2 Step 2 - importing enterprise beans into Object Builder

The `ejb_to_cb.bat` file supplied with the Family Samples uses the `cbejb` tool to deploy the enterprise beans in the EJB server and to generate a model that Component Broker's Object Builder can use to create the necessary files.

#### 10.3.2.1 Object Builder

The Object Builder is a development tool designed to support construction of server objects. It defines the behavior and attributes of server objects and

generates the necessary code for them to function within the Component Broker run-time environment. Object Builder accepts IDL, Rational Rose object models, ERWin, Data Definition Language (DDL) models, and other Object Builder object models defined using XML.

During execution, the cbejb tool opens the Object Builder tool.

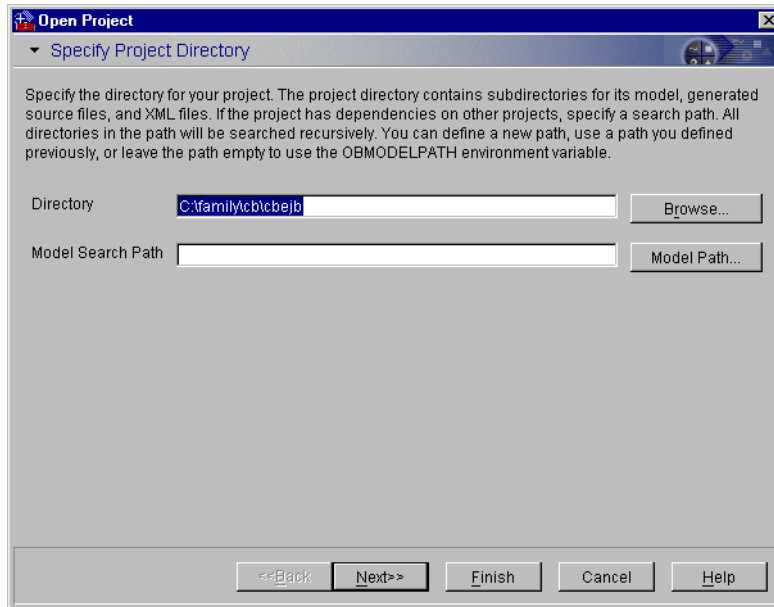


Figure 86. Object Builder tool

Clicking **Finish** will take you into the tool.

### **Defining schemas**

A schema is a description of an object type that is understood and stored in a persistent data store. It is a structural and behavioral abstraction of the real physical data and focuses on information relevant to users of the applications that use an existing database (database schema) or access an existing procedural system (procedural adaptor schema). In this case we are defining a database schema, which is created by importing an SQL file into Object Builder. To do this, we used the following steps:

1. Select **DBA-Defined Schemas**.
2. Open the pop-up menu, select **Import->SQL**.

Schemas imported from an SQL file exist in Object Builder within a schema group, which is just a way of organizing different schemas imported from the DDL file.

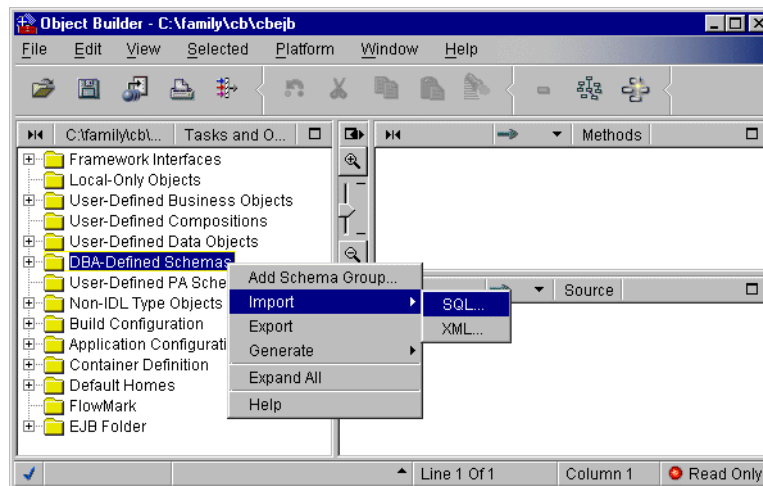


Figure 87. Defining the database schema

The location of the SQL file, the database name (ibmwebs), and the schema group name are required on the next panel.

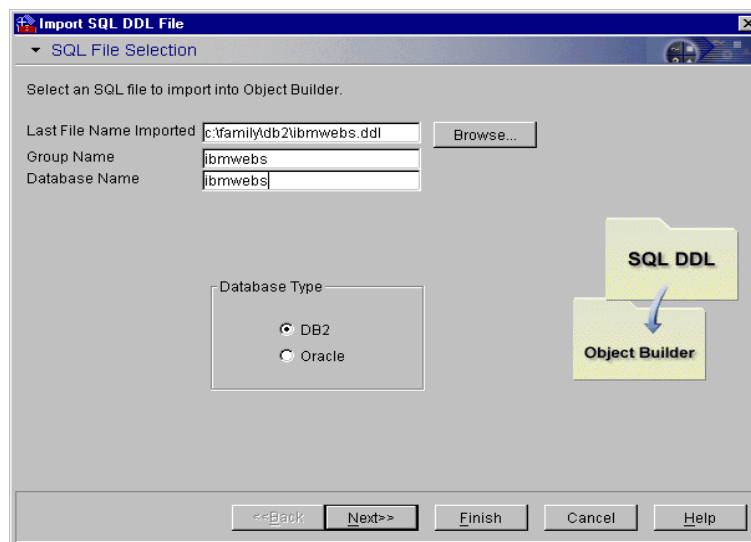


Figure 88. SQL file selection

The ibmwebs.ddl file is just a collection of SQL statements defining our ibmwebs database:

```
CONNECT TO IBMWEBS;

CREATE TABLE IBMWEBS.CUSTOMER (
  custid      CHAR( 4) NOT NULL,
  title       CHAR( 3) NOT NULL,
  fname      CHAR(30) NOT NULL,
  lname      CHAR(30) NOT NULL,
  PRIMARY KEY (CUSTID) );

CREATE TABLE IBMWEBS.ACCOUNT (
  accid      CHAR( 8) NOT NULL,
  custid     CHAR( 4) NOT NULL,
  acctype    CHAR(10) NOT NULL,
  balance    DOUBLE NOT NULL,
  PRIMARY KEY (ACCID));

CREATE TABLE IBMWEBS.TRANSRECORD (
  transid    CHAR(26) NOT NULL,
  accid      CHAR( 8) NOT NULL,
  transtype  CHAR( 1) NOT NULL,
  transamt   DOUBLE NOT NULL,
  traccid    CHAR( 8),
  PRIMARY KEY (TRANSID));

CREATE SYNONYM IBMWEBS.TRANS FOR IBMWEBS.TRANSRECORD;

ALTER TABLE IBMWEBS.TRANSRECORD
  ADD CONSTRAINT "AccountTran" FOREIGN KEY (ACCID) REFERENCES
  IBMWEBS.ACCOUNT ON DELETE RESTRICT;

ALTER TABLE IBMWEBS.ACCOUNT
  ADD CONSTRAINT "CustAccount" FOREIGN KEY (CUSTID) REFERENCES
  IBMWEBS.CUSTOMER ON DELETE RESTRICT;

CONNECT RESET;
```

Figure 89. ibmwebs.ddl

3. Click **Finish** to create the schema.

There are now three schemas defined.

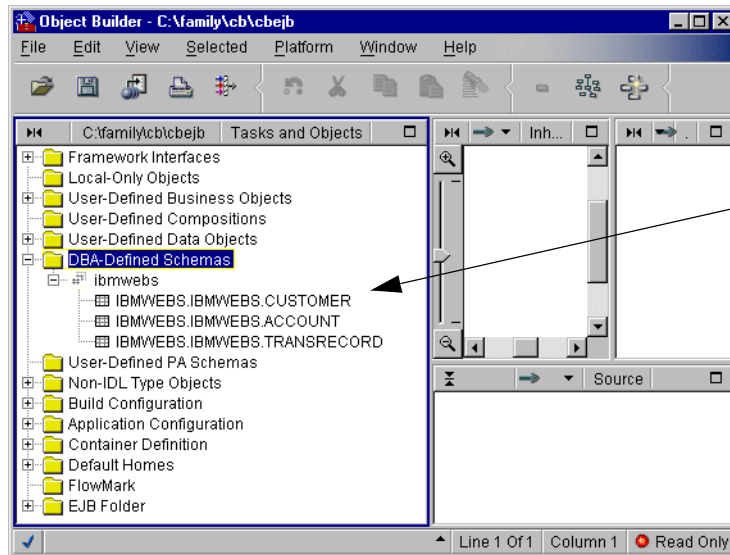


Figure 90. Database schema

### **Adding persistent objects**

Next, the persistent objects are defined. A persistent object is a C++ object that provides a mechanism for storing the state of the component in a data store. Database persistent objects store the state data of the component in the relational database. Multiple persistent objects can be associated with a database schema. To define the persistent objects for each schema do the following:

1. Open the pop-up menu and select **Add Persistent Object**
2. In the Add Persistent Object dialog, select **DB2 Cache Service** for Type of Persistence. This gives you a cached copy of the data object held in the CB server. Leave the other fields set to their default values. Figure 91 shows this for the CUSTOMPO schema. The same is done for ACCOUNPO and TRANSRPO.

**Add Persistent Object**

Names and Attributes

Type a name for the persistent object, and select the type of persistence it provides. A default attribute is defined for each schema column. You can change the name and type for each PO attribute, and indicate whether it is a PO key. Any attribute flagged as Mapped becomes part of the current persistent object.

Name:  ☒ Table is updatable

Package File:  Type of Persistence: ☒ DB2 Cache Service ☐ Embedded SQL

Schema: ibmwebs.IBMWEBS.CUSTOMER

| Column Name | SQL Type | PO Attribute | Attribute Type | Size | PO Key                              | DB Key                              | Not Null                            | Mapped                              |
|-------------|----------|--------------|----------------|------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| custid      | CHAR(4)  | custid       | char[]         | 5    | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| title       | CHAR(3)  | title        | char[]         | 4    | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| fname       | CHAR(30) | fname        | char[]         | 31   | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| lname       | CHAR(30) | lname        | char[]         | 31   | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |

Double-Click for editing

<<Back Next>> Finish Cancel Help

Figure 91. Defining the persistent objects

3. Click **Finish**. Your results should look like those in Figure 92.

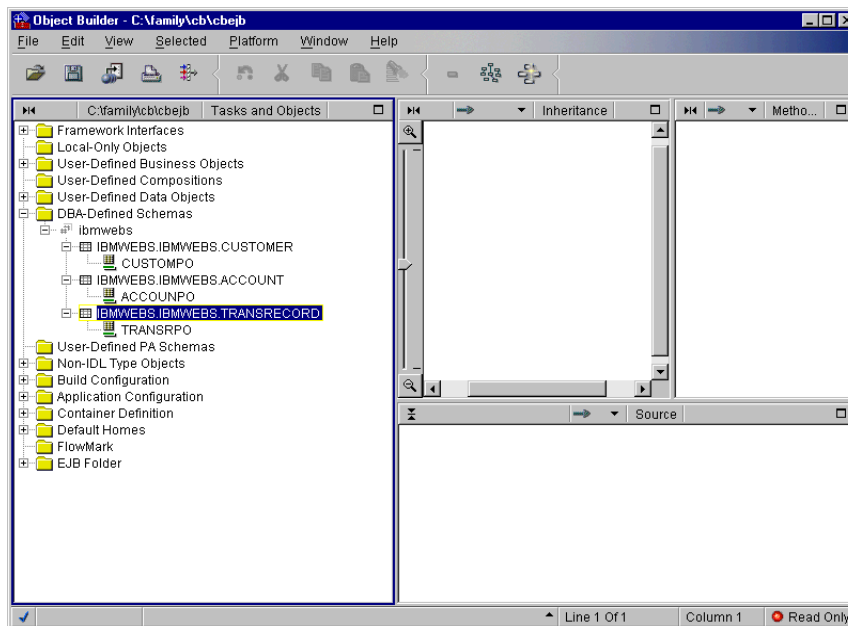


Figure 92. Persistent objects added to schema

### Defining data objects

There are four data objects (DO) under User-Defined Data Objects. We need to map the relationship between these data objects and the persistent objects we just defined.

For each DO except CBBankTasksDO, do the following:

1. Expand the tree and highlight the \*DOImpl leaf.

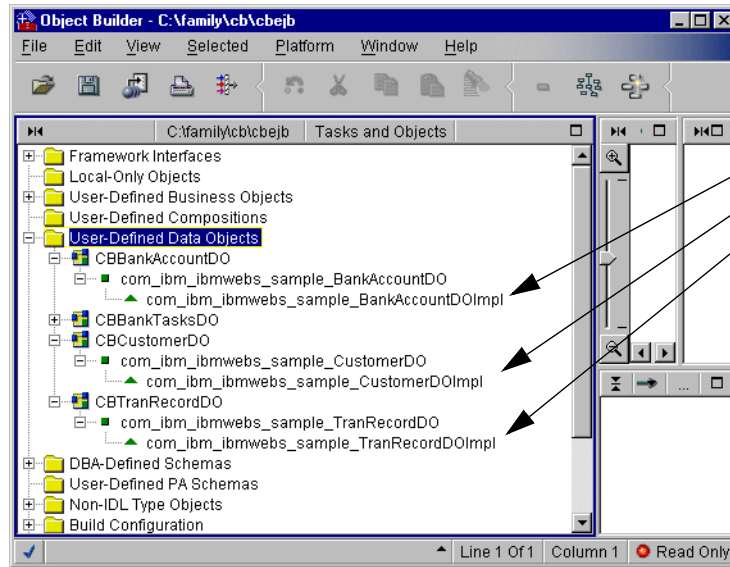


Figure 93. screen capture of the doimpl leaf expanded.

2. Open the pop-up menu and select **Select Persistent Object and Schema**.
3. In the Data Object Implementation dialog box, click **Add Another**.
4. On the Associated Persistent Objects page, select the appropriate persistent object. For example, if you are doing the CBCustomerDO data object, the associated persistent object would be CUSTOMPO.
5. Click **Next**.



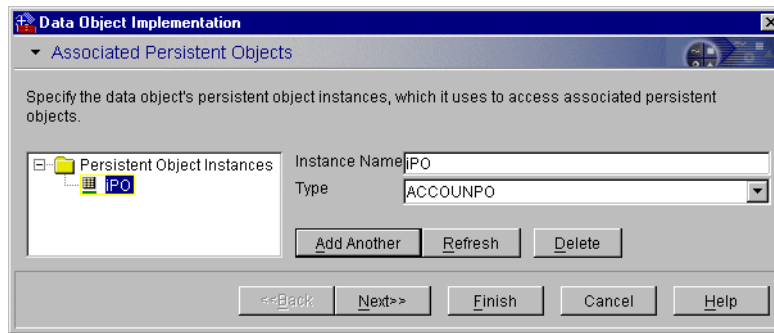


Figure 94. Associated Persistent Objects page for BankAccount

6. In the next screen, the Attributes Mapping page, object attributes are listed in a folder called Attributes. For each attribute that is not mapped (no + sign) do the following:
  - Select the attribute
  - Right-click the mouse and select **Primitive**. A combo box labeled Persistent Object Attribute is displayed.
7. Select the attribute that matches the name in the combo box. For example, the attribute balance matches the primitive IPO.balance from the combo box on the right. These attributes can be seen in Figure 91 on page 193.

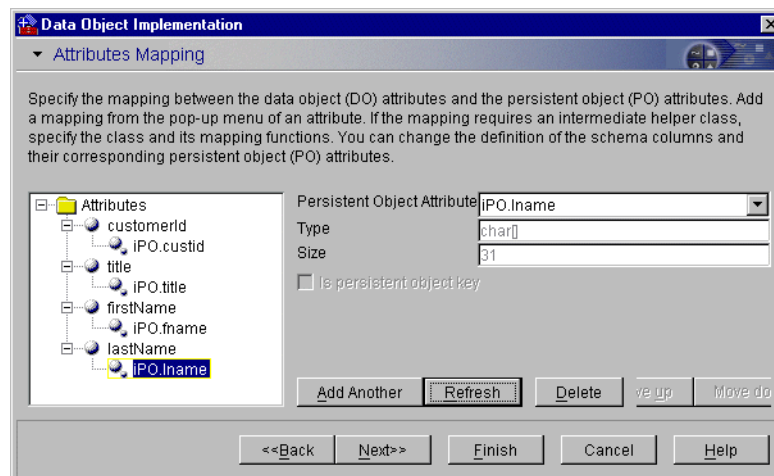


Figure 95. Data object mapping

- When you have mapped all of the attributes, click **Finish**.

The TranRecord.transType attribute requires a mapping helper, which is an intermediate helper class used to define the mapping between the data object and persistent object. The mapping helper is supplied with the application.

You will be prompted to enter the information for the helper when you click **Finish**.

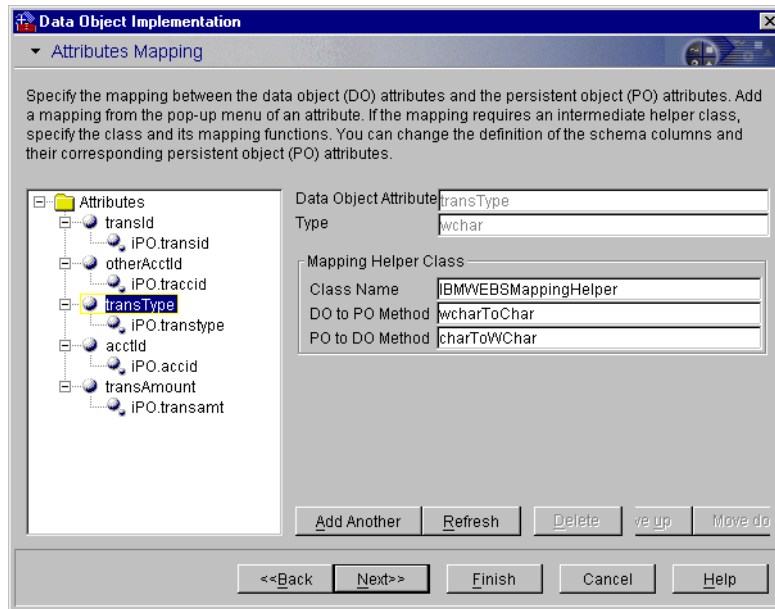


Figure 96. Mapping helper

The finished definitions will look like those in Figure 97.

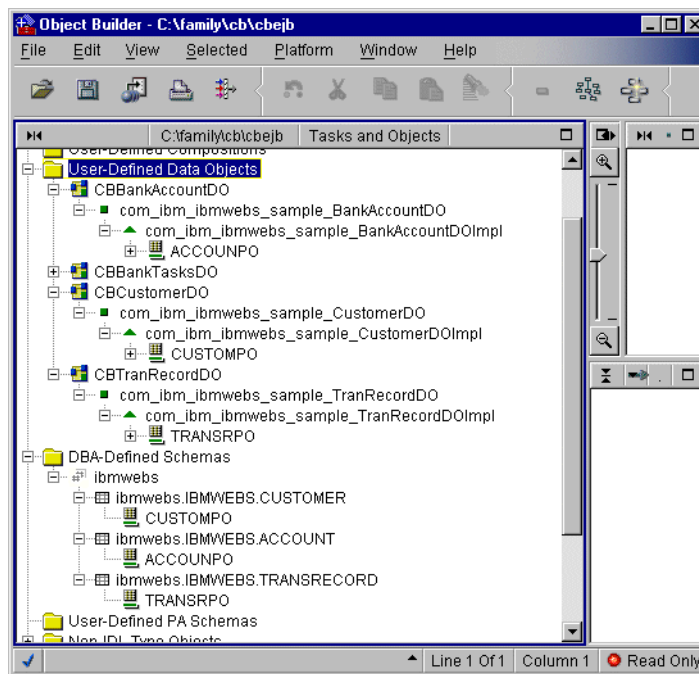


Figure 97. Completed definitions

We are ready to exit the Object Builder.

- Exit (File/Exit) the Object Builder tool. Choose to save the latest changes.
- In the next dialog (Do All Dialog), do the following:
  - Unset **Run Consistency Checker**
  - Check **Generate all code**
  - Uncheck **Build all targets**.

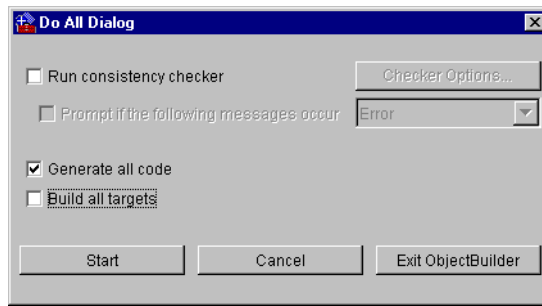


Figure 98. Exiting Object Builder

The Object Builder tool will generate the necessary code to run on Component Broker. When Object Builder completes, choose **Exit ObjectBuilder**.

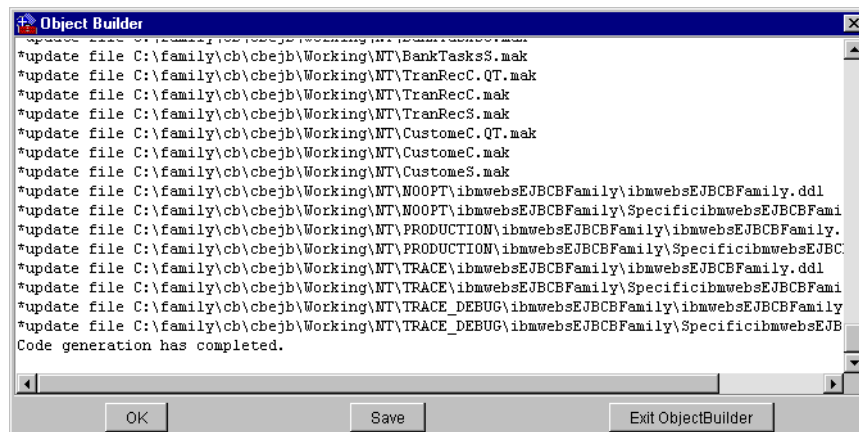


Figure 99. Object Builder generation messages

### 10.3.3 Step 3 - compiling the generated code

Next, we have to compile the Object Builder generated code by using `nmake` (`make` on AIX). All required classes must be in the classpath.

- `nmake -f all.mak`

The `all.mak` file was created by the Object Builder during the code generation step.

### 10.3.4 Step 4 - loading and activating the applications

The next step is to load and activate the applications. This requires Component Broker to be running, with the Location Service Daemon, name server, and SM server running. The processes associated with these services are somorbd and two processes named somsrsm.

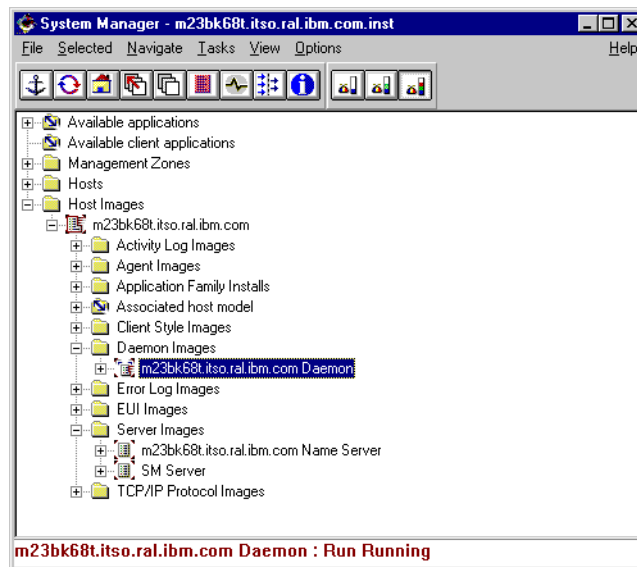


Figure 100. Activating the applications

The Family Sample application provides a batch job (loadCBEJBApplication) to create and activate the definitions for CB on Windows NT. On an AIX platform it must be done manually. The tasks to be done are:

- Create the server group
- Create the server
- Relate the host to the server
- Load the applications
- Add the applications to the configuration
- Add the DB2 services
- Relate the applications to the server group
- Activate the configuration

After the batch file has completed (or the definitions have been added manually), you will see the new server from the CB System Management Interface:

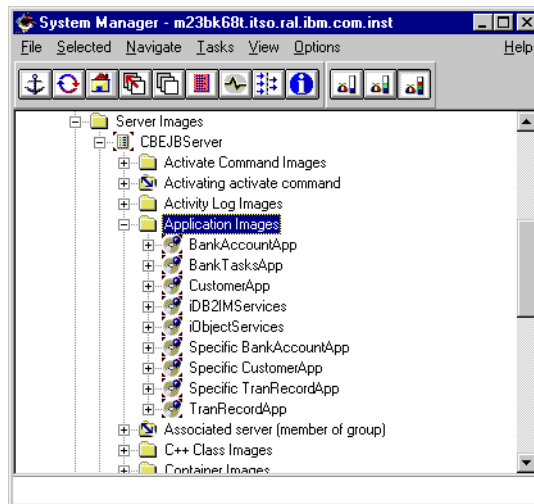


Figure 101. CB definitions for the application

### 10.3.5 Step 5 - preparing for DB2 access

If the application database is located on a remote DB2 server or if the user ID CB is operating under does not have access to the database, you will need to do the following:

- From the Windows NT Start menu, select **Programs->IBM Component Broker->System Manager User Interface**.
- In the System Manager window, set the view level to **Control**.
- Under Host images, find the CBEJBServer and choose **Stop Immediate** from the options.
- Expand CBEJBServer and the RDBConnection Images.

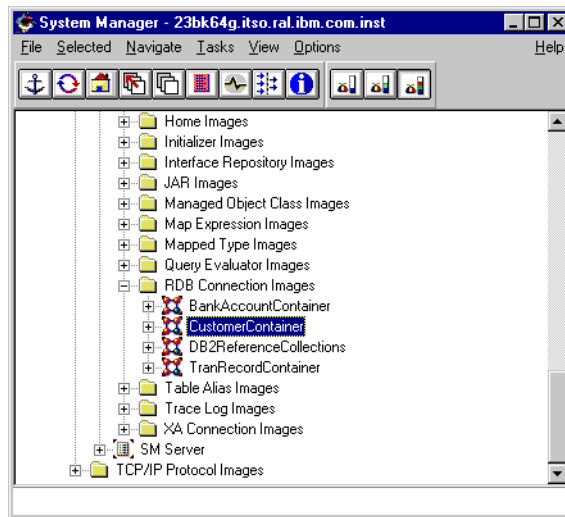


Figure 102. RDB connections

- For each application RDB connection do the following:
  - From the pop-up menu, open **Properties**.
  - Go to the Main notebook page.
  - Append the DB2 user ID/password, separated with commas, to the end of the Open string and click **Apply**.

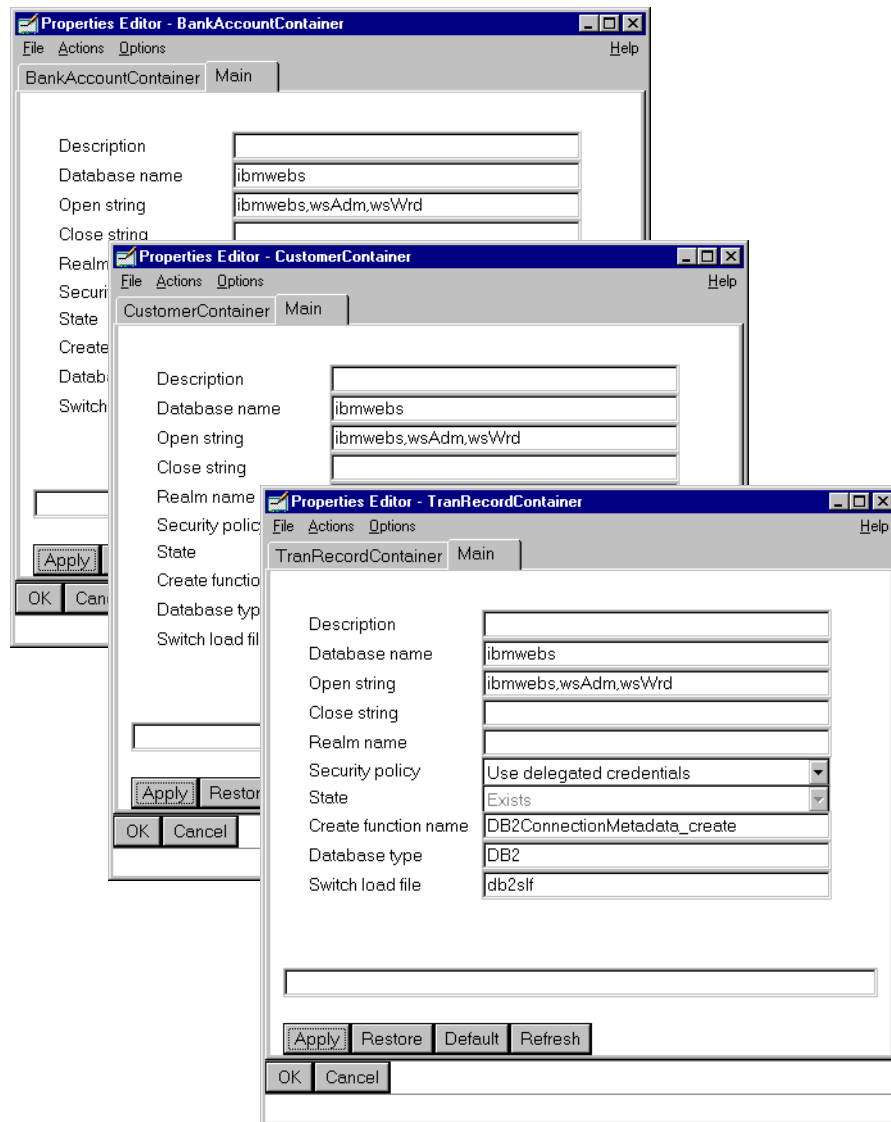


Figure 103. Adding the DB2 user ID and password to the RDB connections

#### 10.3.5.1 Binding the databases

To improve performance and concurrency for applications, Component Broker has a cache service. To configure DB2 to use the Component Broker data cache facility, bind the files db2cntrr.bnd and db2cntcs.bnd to your application database.



Open the DB2 command window, go to the c:\CBroker\etc directory, and issue commands:

```
db2 catalog database ibmwebs as ibmwebs at node BACKEND
db2 connect to ibmwebs user wsAdm using wsWrđ
db2 bind db2cntcs.bnd
db2 bind db2cntrr.bnd
db2 connect reset
```

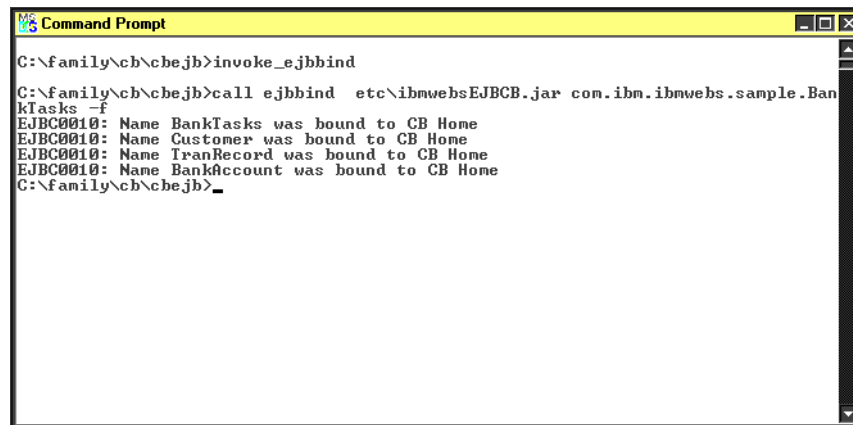
### 10.3.6 Step 6 - update the JNDI naming service

The last configuration step is to update the Java Naming and Directory Interface (JNDI) naming service. Scripts are provided with the application to run the ejbbind tool to bind the sample enterprise bean names to the JNDI name tree, which is implemented using the underlying Component Broker COS Naming service.

- invoke\_ejbbind.bat (NT)
- invoke\_ejbbind.sh (AIX).

```
call ejbbind etc\ibmwebsEJBCB.jar com.ibm.ibmwebs.sample.BankTasks -f
call ejbbind etc\ibmwebsEJBCB.jar com.ibm.ibmwebs.sample.Customer -f
call ejbbind etc\ibmwebsEJBCB.jar com.ibm.ibmwebs.sample.TranRecord -f
call ejbbind etc\ibmwebsEJBCB.jar com.ibm.ibmwebs.sample.BankAccount -f
```

Figure 104. Binding the enterprise beans to the naming service



```
Command Prompt
C:\family\cb\chejb>invoke_ejbbind
C:\family\cb\chejb>call ejbbind etc\ibmwebsEJBCB.jar com.ibm.ibmwebs.sample.Ban
kTasks -f
EJBC0010: Name BankTasks was bound to CB Home
EJBC0010: Name Customer was bound to CB Home
EJBC0010: Name TranRecord was bound to CB Home
EJBC0010: Name BankAccount was bound to CB Home
C:\family\cb\chejb>
```

Figure 105. Running ejbbind

### 10.3.7 Step 7 - running the application server

To start the CBEJBServer, you can go to the System Management Interface, right-click on the server and select **Run**.

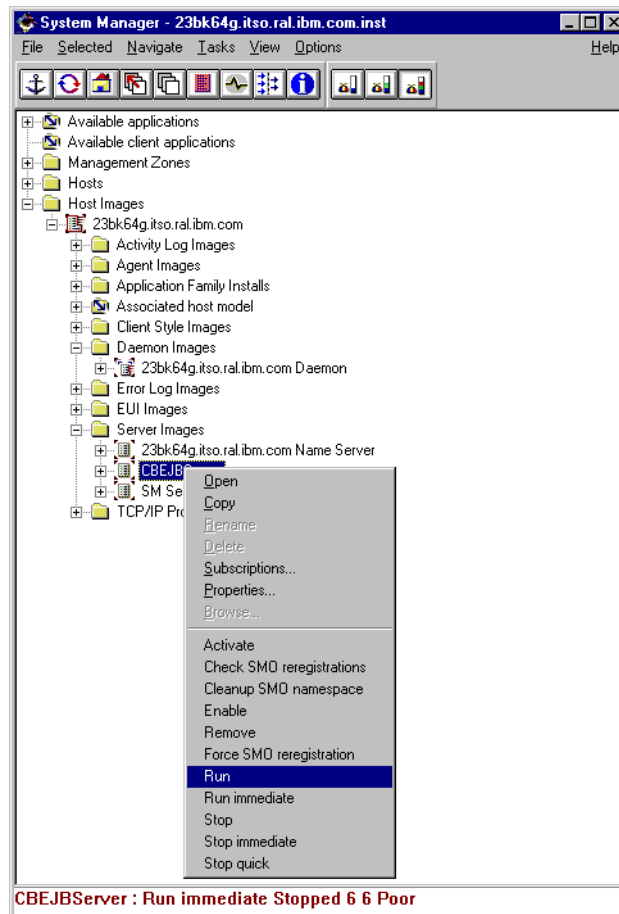


Figure 106. Starting the server

For testing purposes, it may be helpful to start the server from a command prompt. This allows you to have an open window with the server console messages appearing as activity takes place. To start the server from a command prompt issue the following commands:

```
Set SOMCBENV=S:CBEJBServer
somsrsm
```

---

## 10.4 Creating and Installing the Mortgage Payment System application

The design of the mortgage payment application is discussed in Chapter 6, “Application design guidelines” on page 73. This is a quick overview of how we implemented the application and is intended to give you an idea of how to install an MQSeries application into CB.

### 10.4.1 Step 1 - generating the code

Creating an enterprise bean to communicate with MQSeries requires the use of three tools:

- mqaaejb
- jetace
- cbejb

To make things simple, we created a file (build.bat) to do all three steps. These steps are explained in the following sections.

```
echo running mqaaejb command for outbound EJB
call mqaaejb -f MPSOutbound.properties -n MPS -p mps.mq
echo
echo add deployment descriptors to ejb-jar file
call jetace -f MPSejb.xml
echo
echo running cbejb command to deploy mqaa ejbs
call cbejb MPSejb.jar -ob ..
```

*Figure 107. build.bat*

In this case, we are building the MPSOutbound Session bean, which is introduced into the application and described in 6.8, “Implementing topology 5: the Mortgage Payment System” on page 134.

#### 10.4.1.1 mqaaejb

The mqaaejb tool generates a Session bean that wraps a Component Broker MQSeries-backed business object (BO). The mqaaejb command will generate Java source files and the corresponding compiled class files, a JAR file containing the compiled files that compose the enterprise bean, and an XML file containing the enterprise bean’s deployment information which will be

used as input to the jetace tool to build the deployment descriptor for the enterprise bean.

**Note:** The mqaaejb.bat file shipped with CB 3.0 and 3.02 set the classpath incorrectly. Use the following:

```
setlocal

set EJB_ROOT=%SOMCBASE%\lib
set CLASSPATH=%CLASSPATH%;%EJB_ROOT%\developEJB.jar
java com.ibm.ejb.cb.tools.mqaa.MQAAEJB %*

rem Restore the original CLASSPATH setting:
endlocal
```

Figure 108. mqaaejb.bat

The mqaaejb tool requires a properties file containing the message type specification and a list of message field specifications. The message type can be Inbound, Outbound, or InOut. When InOut is chosen, a pair of enterprise beans, instead of a single one, are created to accommodate paired inbound and outbound message queues. Each message field is listed by name with the field type.

The properties file required for our mortgage application is shown in Figure 109.

```
messageType=Outbound
queueName=java.lang.String
accountId=java.lang.String
paymentAmount=java.lang.String
```

Figure 109. MPSOutbound.properties

The format of the mqaaejb command is:

```
mqaaejb -f propertiesFile -n baseBeanName [-p packageName] [-b
BOInterface]
```

Where:

- *propertiesFile* specifies the name of the properties file
- *baseBeanName* specifies the base name of the enterprise bean to be generated

- *packageName* specifies the package name of the enterprise bean; if not specified, the package name defaults to mytest.ejb.mqaa.
- *BOInterface* tells the tool that there is an existing Component Broker BO for the MQ queue and to connect the generated enterprise bean to that BO, rather than to create a duplicate BO.

In our example we call the mqaaejb tool with the following command:

```
call mqaaejb -f MPSOutbound.properties -n MPS -p mps.mq
```

The output of this is the following files:

- MPSMsgTemplate.java
- MPSOutboundBean.java
- MPSOutboundEJBHome.java
- MPSOutboundEJBObject.java
- MPSMsgTemplate.class
- MPSOutboundBean.class
- MPSOutboundEJBHome.class
- MPSOutboundEJBObject.class
- MPSejb.xml
- MPSejb.jar

#### 10.4.1.2 jetace

We will use the jetace tool to create a deployment descriptor for the enterprise bean and package this deployment descriptor into the EJB jar file. The resulting EJB JAR file contains the class files, deployment descriptor, and an EJB-compliant manifest file.

The format of the jetace command is:

```
jetace -f beanName.xml
```

For our example we used:

```
call jetace -f MPSejb.xml
```

MPSejb.xml was generated by the mqaaejb tool. You can see it in Figure 110.

```

<?xml version='1.0' encoding="ISO-8859-1"?>
<!-- (C) COPYRIGHT International Business Machines Corp. 1998 -->
<!-- ALL Rights Reserved -->
<!-- Generated on Fri Jul 07 14:32:45 EDT 2000 -->
<!-- by utility MQAAEJB -->

<ejb-jar>
<input-file>MPStmp.jar</input-file>
<output-file>MPSejb.jar</output-file>

<session-bean dname="MPSOutboundBean.ser">

<session-timeout>0</session-timeout>
<state-management>STATEFUL_SESSION</state-management>

<jndi-name>mps/mq/MPSOutboundEJBHome</jndi-name>
<transaction-attr value="TX_MANDATORY"/>
<isolation-level value="READ_COMMITTED"/>
<run-as-mode value="CLIENT_IDENTITY"/>

<enterprise-bean>mps.mq.MPSOutboundBean</enterprise-bean>
<remote-interface>mps.mq.MPSOutboundEJBObject</remote-interface>
<home-interface>mps.mq.MPSOutboundEJBHome</home-interface>

</session-bean>

</ejb-jar>

```

Figure 110. MPSejb.sml

#### 10.4.1.3 cbejb

The cbejb tool is used to deploy the enterprise bean contained in the EJB JAR file. It works with Object Builder to create and compile the necessary files needed by CB to manage the enterprise bean.

### 10.4.2 Step 2 - importing the enterprise beans into Object Builder

During the execution of the cbejb tool, the Object Builder will start.

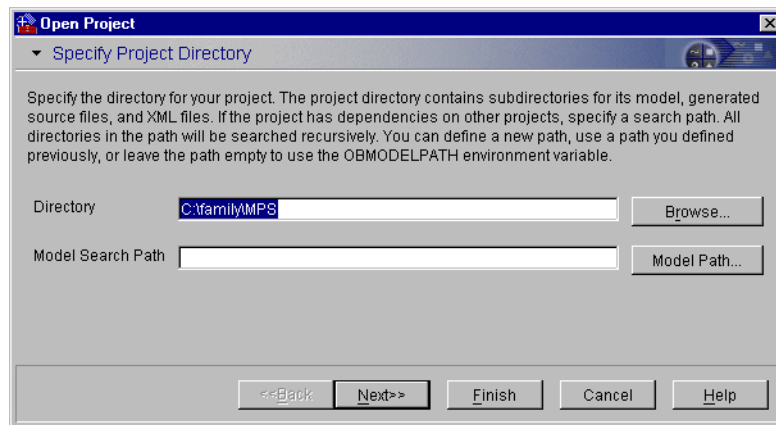


Figure 111. Object Builder start panel

1. Click **Finish** to move to the next screen.
2. The DO implementation will need to be altered to suit our needs. The code required for the insert() method has been automatically generated, but needs a few adjustments for our application.

Expand User-Defined Data Objects until you get to the MPSOutbound DOImpl leaf:

- In the Methods pane, select **Framework Methods->public void insert()**.

You can see the default code in the source window. We took that code, copied it to a new file called do\_insert.tde, and modified it. The modifications made included hard coding the MQ queue name and adding a few lines of code, which performs the formatting of the actual message that will be put onto the message queue. The code for this can be seen in A.1, “MPSOutbound DO include()” on page 271.

- Right-click and select **Properties**.

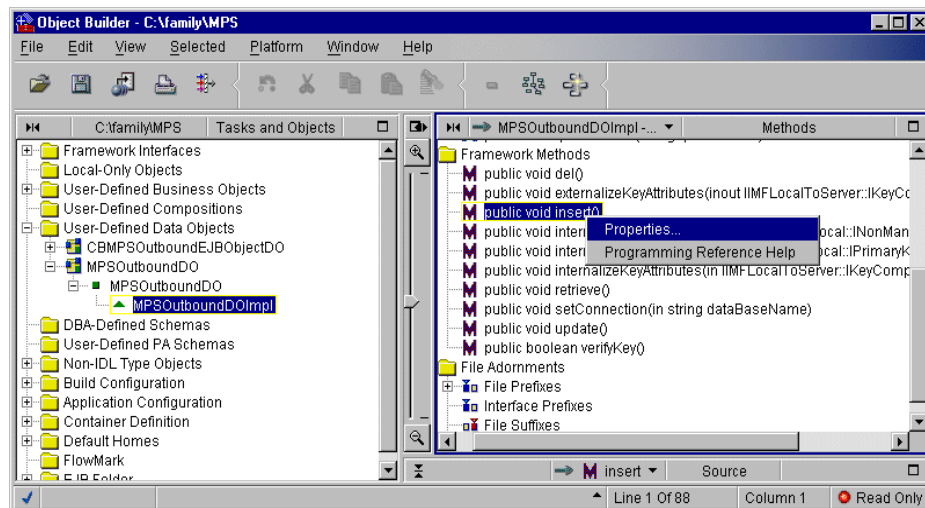


Figure 112. Framework methods - public void insert() properties

3. In the properties window, select **Use an external file** and browse to select **family\mps\source\do\_insert.tde**. Click **Finish**.

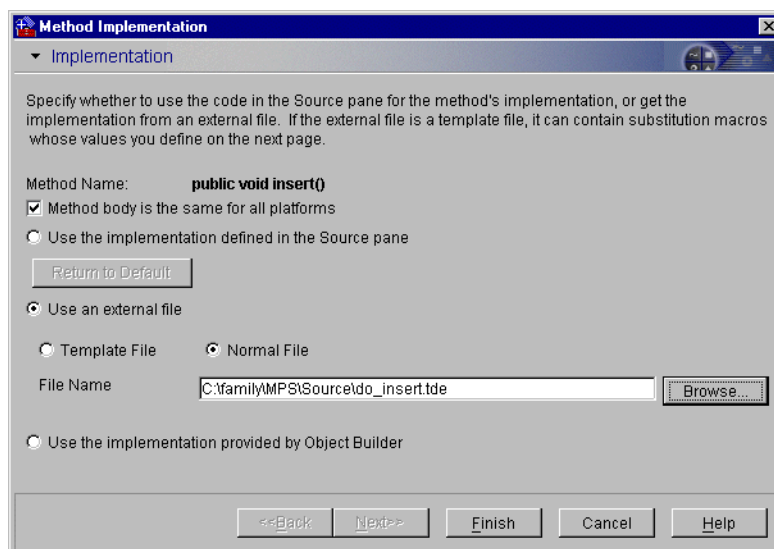


Figure 113. Adding the modified insert() code



- Some of the code added will require the `stdio.h` C++ file in order to compile correctly. To do this, click **File Adornments->File Prefixes**. Right-click and select **Add**.

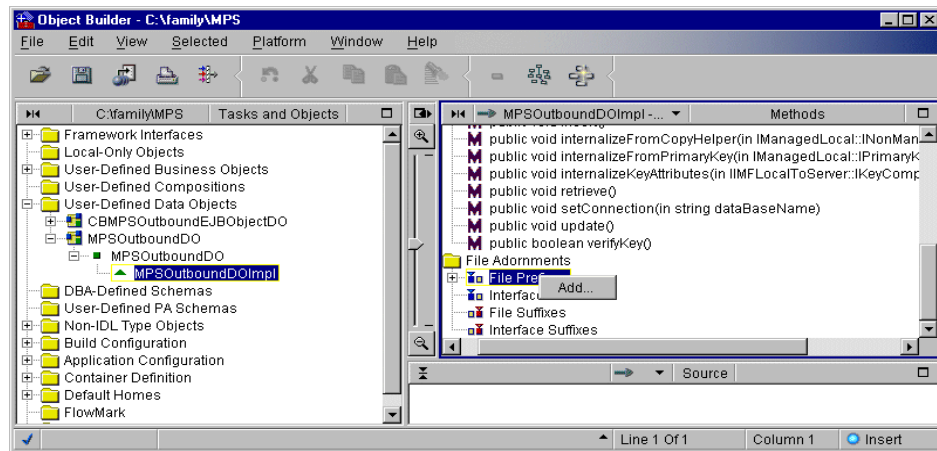


Figure 114. Including required source files

- Enter “includes” and keep all defaults.

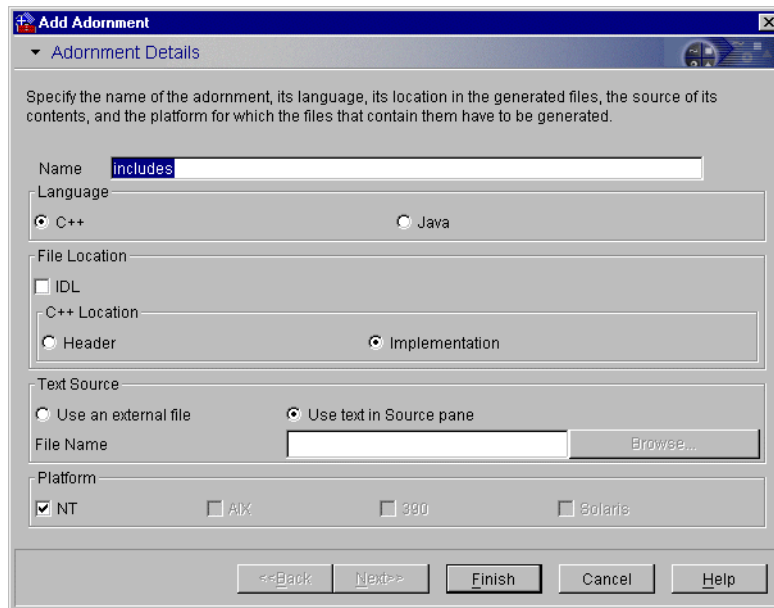


Figure 115. Adornment details

Click **Finish**.

6. Back in the Source pane type `#include <stdio.h>`

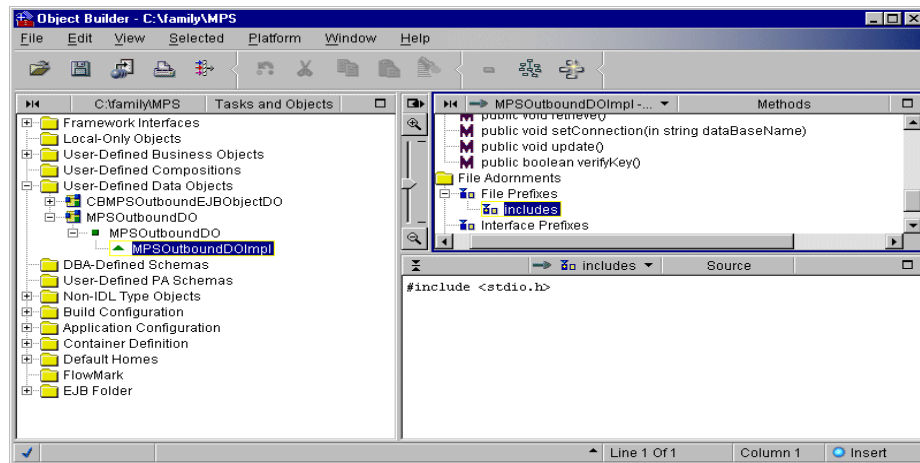


Figure 116. Including `stdio.h`

7. The last thing to modify is the MQSeries queue manager name. In the Tasks and Objects pane, expand the Container Definition folder. The container represents and defines the characteristics of a specific queue manager. The queue manager is treated as a persistent data store in which data can be stored and retrieved within the scope of one or more transactions.

Select the `MPSOutboundContainer`, right-click and select **Properties**.

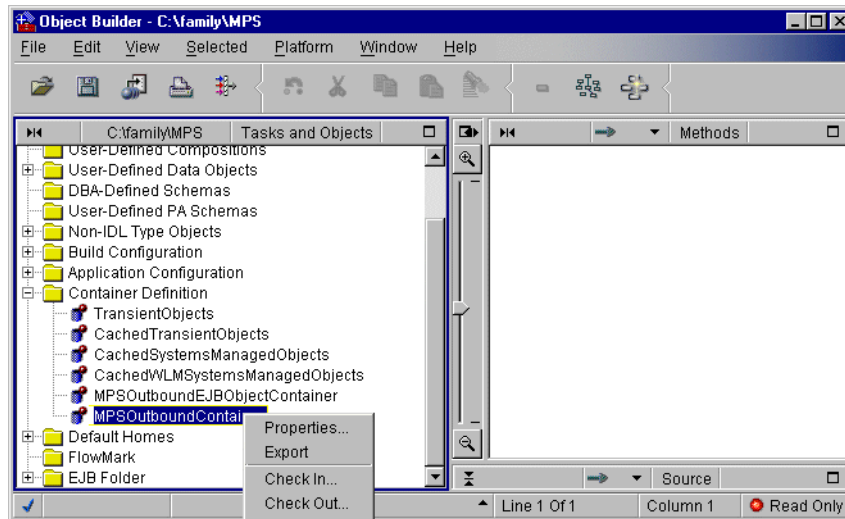


Figure 117. MPSOutboundContainer properties

8. Select **Next** until you come to Services Details page. Enter "MPS.QUEUE.MANAGER" as the queue manager name and select **Finish**.

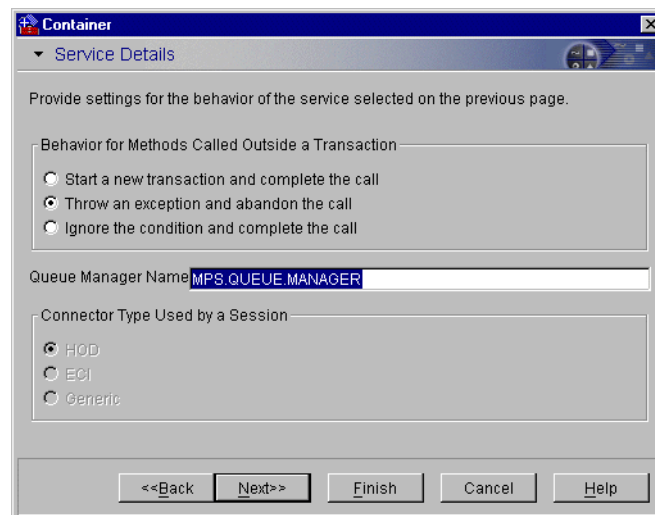


Figure 118.

9. From the main panel select **File->Do All**.
  - In the Do All Dialog, check only **Generate all code**

- Click **File->Save**
- Click **File->Exit**

### 10.4.3 Step 3 - compiling the generated code

Next, we need to compile the object model. Navigate to the directory in which you initially generated the code in Step 1 and run the following command.

```
cd ..\working\nt
nmake -f all.mak
```

Figure 119. compile.bat

You now have the MQ-backed EJB component, called MPSOutbound, fully built and ready to be installed into the CB runtime environment.

### 10.4.4 Step 4 - Loading and activating the applications

The next step is to load and activate the applications. Once this step is completed, the MQ-EJB component will be installed and running in Component Broker.

1. Ensure that the SM server is running.
2. The applications can be loaded manually or by running a batch file. The CBEJBSG server group and CBEJBServer were defined with the Family Sample application. For the Mortgage Payment System, we need to do the following:

- Load the DDL file under the MPS Object Builder project directory:

\Working\NT\PRODUCTION\MPSObjFamily\MPSObjFamily.ddl

DDL files are CB-specific application deployment definitions, and these application definitions were created by the `cbejb` command in Step 1.

- Add the MPSOutboundApp, and MPSOutboundEJBObjectApp applications.
- Relate the following applications to the CBEJBSG server group in the Sample Configuration of the Sample Application Zone.
  - MPSOutboundApp
  - MPSOutboundEJBObjectApp
  - iMQAAServices (installed with the MQ Application Adaptor)
  - iDXAAAServices (installed with the MQ Application Adaptor)

- Activate the configuration

We created a batch file to do this for us. The file can be seen in A.6, “Loading the MQ application in CB” on page 320. The results of the definitions are shown in Figure 120.

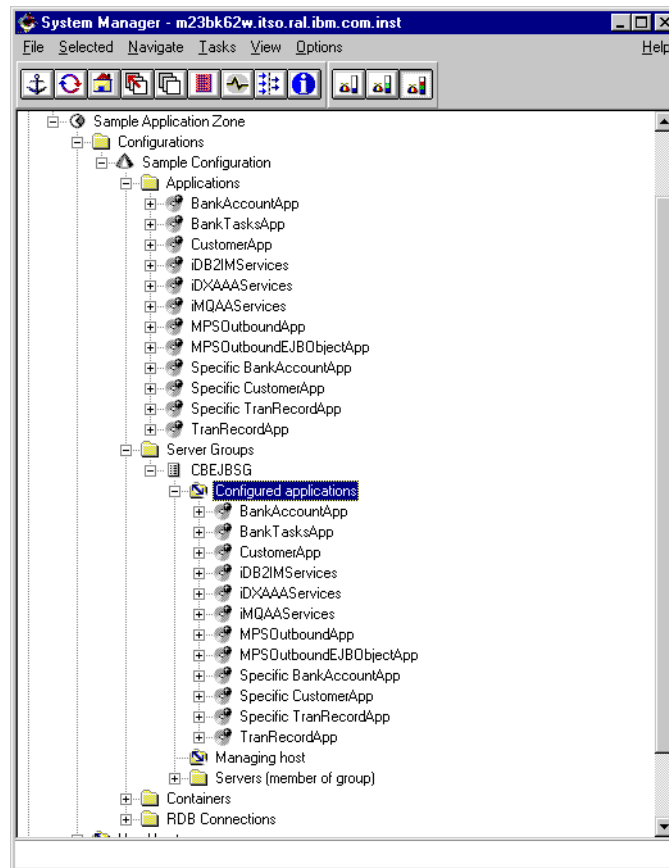


Figure 120. New application definitions

3. Start the CBEJBServer, if not already running.
4. Call ejbbind to update the JNDI naming services. The ejbbind tool allows you to specify the host name and port number. If you have changed the port (for example, if you are running WebSphere Advanced and CB on the same machine), use the -ORBInitialHost and -ORBInitialPort parameters to specify the host and port. Otherwise, they default to the local host and port 900.

```
call ejbbind MPSejb.jar mps.mq.MPSOutboundEJBObject
```

Figure 121. *bind.bat*

#### 10.4.5 Create the MQSeries queue manager and queues

This process assumes that MQSeries has been installed. The MQSeries server will act as a queue manager for the Mortgage Payment System. Installing the MQ Server is described in *MQSeries for Windows NT: Quick Beginnings*.

The following bat file can be used to start MQSeries and define the MQ manager and queue:

```
net start "IBM MQSeries"  
crtmqm -q MPS.QUEUE.MANAGER  
runmqsc MPS.QUEUE.MANAGER < MPSMQcfg.txt > MPSMQcfg.lst  
strmqm  
cd .\MPSPostingService  
cls  
MPSPostingService
```

Figure 122. *setup.bat*

Note that the queue manager name matches the name you specified in step 8 on page 213. The setup.bat file uses a file (MPSMQcfg.txt) for the queue definitions.

```

*****/
* Define the initiation queue */
*****/
DEFINE QLOCAL (MPS.TRIGGER.QUEUE) REPLACE +
    LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE) +
    DESCR('Queue for triggering MPS Posting Service')
*****/
* Define the application queue for the MPS Posting Service */
* triggering. Messages placed on this queue cause */
* trigger messages to be placed on the MPS.TRIGGER.QUEUE. */
*****/
DEFINE QLOCAL (MPS.QUEUE) REPLACE +
    LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE) +
    DESCR('Queue for triggering MPS Posting Service') +
    INITQ('MPS.TRIGGER.QUEUE') +
    PROCESS('MPS.TRIGGER.PROCESS') +
    TRIGGER +
    TRIGTYPE(EVERY) +
    DEFPSIST(YES)
*****/
* Define the process which will be part of the trigger message */
* for the MPS Posting Service triggering. */
*****/
DEFINE PROCESS (MPS.TRIGGER.PROCESS) REPLACE +
    DESCR('MPS Posting Service Trigger Process') +
    APPLTYPE (WINDOWSNT) +
    APPLICID ('MPS Posting Service')

```

Figure 123. MPSMQcfg.txt

#### 10.4.6 Creating an Access bean for the MQ-EJB component

The Family Sample application uses Access beans for all client access to EJBs. We create an Access bean for the MQ EJB component (MPSOutbound) so that our EJB client code will follow the same coding practice set forth in the existing Family Sample application upon which we are building. The Access bean created here is used in the code which communicates with the MPSOutbound bean in the Payment\_CMD Command bean (described in 6.8.3, “Implementing the Mortgage Payment System” on page 139 and in the BankTasks transfer() method (described in 6.9.2, “Implementing the Consumer Banking Plus application” on page 146.

In order to create the Access bean, we need to import the MPSOutbound Session bean into VisualAge for Java.

The following steps assume that you have already imported and added the VisualAge for Java repository for the Family Sample into your VisualAge for Java Workbench.

1. In VisualAge for Java, import the .java files from the directory where you executed the `mqaaejb` command into the IBM WebSphere Family Samples project. These files were created by the `mqaaejb` tool (see 10.4.1.1, “mqaaejb” on page 205) and are the source files for the generated MPSOutbound enterprise bean.
2. In the EJB tab, add a new EJB group called `IBM_WebSphere_Family_Samples_MPS`.
3. Add an enterprise bean using the following values:

Name: MPSOutbound  
Type: session  
Use an existing bean class  
package: mps.mq  
class: mps.mq.MPSOutboundBean  
home interface: mps.mq.MPSOutboundEJBHome  
remote interface: mps.mq.MPSOutboundEJBObject

4. Select properties of MPSOutbound to set the deployment descriptor:

JNDI name: mps/mq/MPSOutboundEJBHome  
transaction attribute: TX\_MANDATORY  
Isolation level: TRANSACTION\_READ\_COMMITTED  
State Management: #STATEFUL

5. Add an Access bean for the MPSOutbound bean. Select **MPSOutbound**, right-click, and select **Add->Access Bean**. Accept all defaults and click **Finish**.

#### 10.4.7 MPSPostingService “legacy” application

To integrate MQSeries into the application, we made a simple application to take the messages sent by the EJB and open a window, showing a message reflecting the transaction. This is a simple but effective way of verifying that the MQSeries portion of the process is working.

The application, MPSPostingService, is written in C++. The source, along with the mak file is shown in A.7, “MPSPostingService application” on page 323.



### 10.4.8 Testing the MQ-EJB component

As a test to see if the MPSOutbound component is working correctly, we created a small test client. The code for this test client is shown in A.5, “MPSOutboundTest.java” on page 314.

To run this test:

- Start the CBEJBServer from the CB SM, or run it from the command line to see output from the MQ component.
- Run the simple test client using the commands shown in `run.bat` shown in Figure 124 below.

```
set local

set FAMILY=C:\family
set MPS=%FAMILY%\mps
set
classpath=%CLASSPATH%;%MPS%\Working\NT\PRODUCTION\JCB\jcbMPSOutboundEJB
ObjectC.jar;%FAMILY%\vaj\ivjejb302.jar

java mps.mq.MPSOutboundTest m23bk62w.itso.ral.ibm.com 910 %1

endlocal
```

Figure 124. `run.bat`

If everything goes well, Notepad should open a file called `MPSReport.txt` with the text `11114000's payment is 1000.00`.

---

## 10.5 Running Consumer Banking and Consumer Banking Plus

From a user's point of view, our Consumer Banking Plus application and the Family Sample (Consumer Banking) application will look much alike. The difference will appear in the screens that present accounts, with the addition of the mortgage account. We have not provided you with the source code for this application, but if you have installed the Family Sample application, you will see the same initial interface.

To test our application, we first started the WebSphere Advanced server, the Component Broker servers (SM server, CBEJBServer, and Name Server), the MQSeries Queue Manager, and the MPSPostingService application.

A user opening a Web browser to the following URL:

`http://<hostname>/FamilySample/WSFamily/`

would see the screen presented in Figure 125.

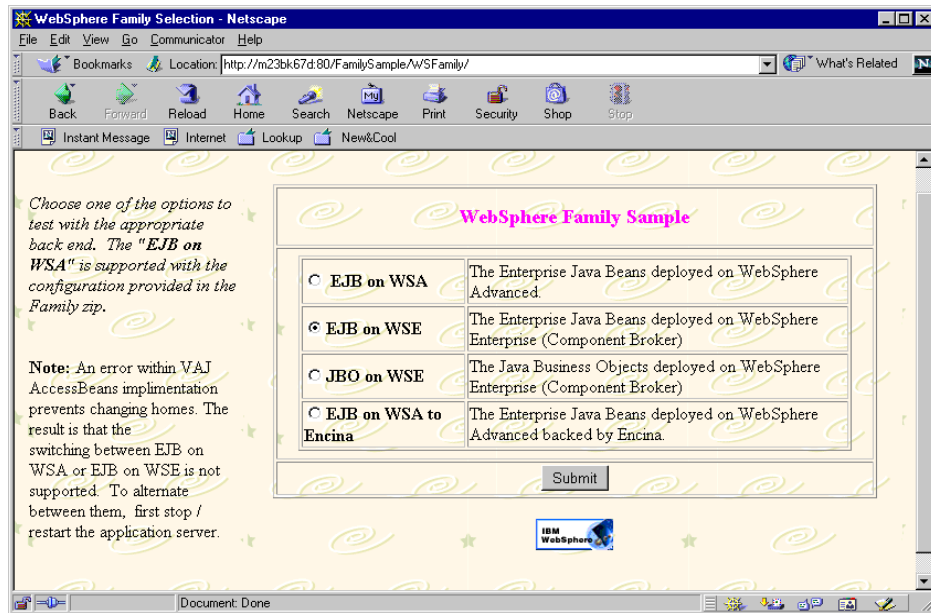


Figure 125. Sample application

- Select the **EJB on WSE** option.

If you have implemented security at the WebSphere Advanced Web application (as we do later in Chapter 12, "Application security" on page 237), you will be challenged for a user ID and password

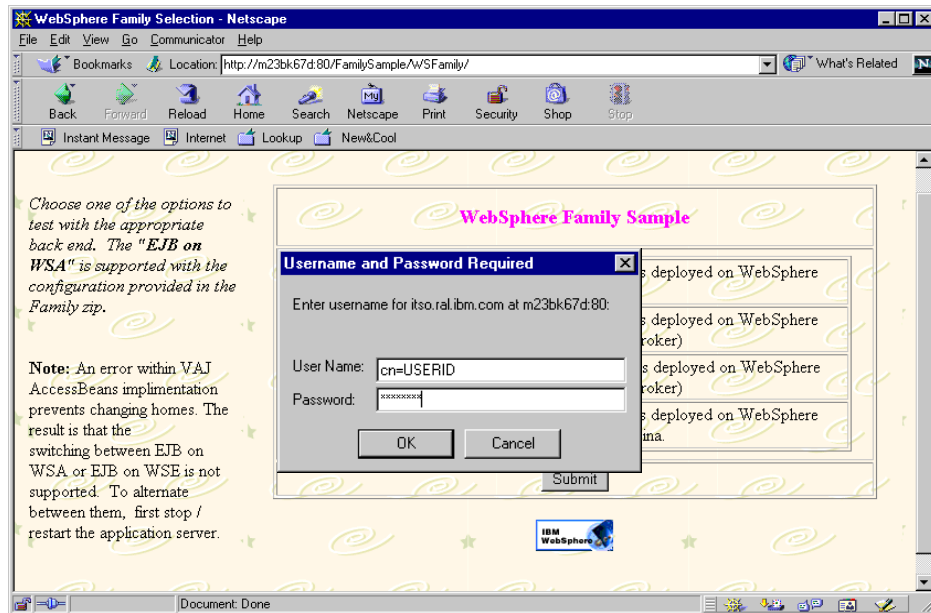


Figure 126. Sample application

- Log in as user ID 1111, 1234 or 2222. If an invalid user ID is entered, an error message appears stating that the ID does not exist in the records.

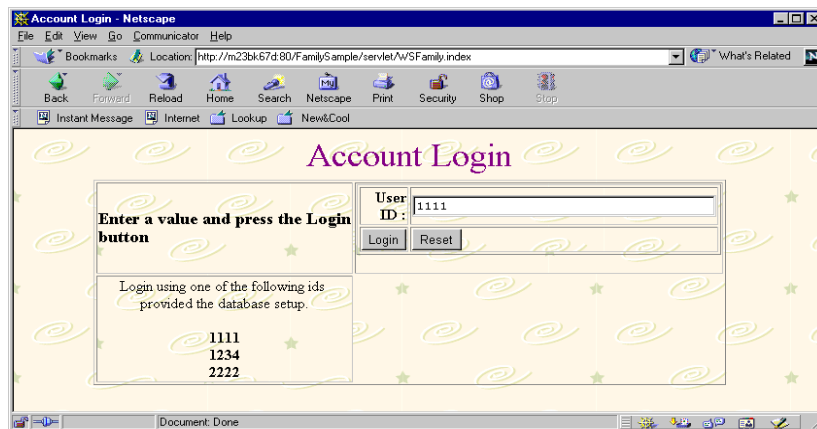


Figure 127. Sample application

- There are three options for the user:
  - Accounts: This will show all the accounts that the user has.

- Transfer: This will allow the user to transfer funds from one account to another.
- Signoff: Log off the account.



Figure 128. Sample application

The next logical step is to click the **Accounts** button. From this point on, the application appears as described in 6.9.1.1, “Consumer Banking Plus view” on page 144.

## 10.6 Topology 5 implementation

We developed an MPService package that contains the Command beans and a servlet for processing the mortgage payment from a browser to communicating with the MQ-backed EJB component. These are described in 6.8, “Implementing topology 5: the Mortgage Payment System” on page 134.

The code for the servlet and Command bean for this sample include:

- `PaymentServlet.java`

This is the servlet class that communicates with the Command bean and sends back the reply to the browser by forwarding to a JSP. See A.2.1, “`PaymentServlet.java`” on page 274.

- `PaymentServlet.servlet`

This is the XML parameter file for the `PaymentServlet`. See A.2.2, “`PaymentServlet.servlet`” on page 278.

- `Util.java`

This is a simple utility class that handles the creation of the Command bean. See A.2.3, “`Util.java`” on page 279.

- `CMD.java`

This is intended to be a super class of all Command beans used for this sample. See A.2.4, “`CMD.java`” on page 280.

- `Payment_CMD.java`

This is the Command bean used by the `PaymentServlet` to communicate with the `MPSOutbound` enterprise bean. See A.2.5, “`Payment_CMD.java`” on page 281.

- `MPSTransactionException.java`

This is the general exception that gets thrown back to the `PaymentServlet` if something goes wrong during the communication with the `MPSOutbound` enterprise bean inside of the `Payment_CMD` Command bean. See A.2.6, “`MPSTransactionException.java`” on page 286.

The code for the JSPs for this sample include:

- `MPSPayment.jsp`

This is the initial view for submitting mortgage payments. See A.3.1, “`MPSPayment.jsp`” on page 287.

- `MPSPaymentSubmitted.jsp`

This is the result view for mortgage payments submitted successfully. See A.3.2, “`MPSPaymentSubmitted.jsp`” on page 288.

- `MPSPaymentError.jsp`

This is the result view for mortgage payments submitted unsuccessfully. See A.3.3, “`MPSPaymentError.jsp`” on page 289.

### 10.6.1 Installing the application in WebSphere Advanced

To install this application into the WebSphere Advanced system:

1. Add the following to the classpath of the FamilyServer (the application server configured when installing the Family Sample application into WebSphere Advanced) in the command line arguments:

jcbMPSOutboundEJBObjectC.jar

This file is located under the directory in which you created the Object Builder object model for the MPSOutbound enterprise bean. The object model project directory was created in the current directory in which you ran the `cbejb` command. Under this directory, this file is located in `\Working\NT\PRODUCTION\JCB`. This file has the EJB client bindings generated for use with the MPSOutbound bean installed into Component Broker.

2. Place the following compiled class files in the classpath of the (WebSphere Advanced) FamilyServer. This can be a directory in the classpath of the system or specified from the command line arguments of the FamilyServer.
  - The `mps.mq.MPSOutboundAccessBean` (see 10.4.5, “Creating an Access Bean for the MQ-EJB Component”).
  - The classes associated with the MPS service described above, except for the servlet (`CMD`, `Payment_CMD`, `Util`, `MPSTransactionException`).
3. Publish the JSP files and the servlet (including its XML configuration file - `PaymentServlet.servlet`) to the servlet directory specified by the FamilySample Web Application in the WebSphere Advanced Admin Console.
  - Publish the JSP files to `\family\website\FamilySample\web`
  - Publish the servlet files to `\family\website\FamilySample\servlets`

### 10.6.2 Running the topology 5 application

The application can be called by accessing the JSP:

`http://<hostname>/FamilySample/MPSservice/MPSPayment.jsp`

The interface would appear as described in 6.8.2.1, “Mortgage Payment System view” on page 137.

---

## 10.7 Topology 6 Implementation

In 6.9.2, “Implementing the Consumer Banking Plus application” on page 146, you will find a high-level overview of the enhancements made to the Consumer Banking application to implement the Topology 6 design. Here, we go into more detail about exactly what modifications were made.

There are four pieces of functionality that need to be added to the original Family Sample (Consumer Banking) application code. These are:

- Update the ShowAccounts\_CMD Command bean to expose to the user a list of all accounts, “from” accounts, and “to” accounts. The reasoning behind this is that mortgage accounts should never be listed as a “from” account, because customers are not allowed to withdraw money from mortgages.
- Update the ShowAccountsResults JSP so that it retrieves the complete account list from the ShowAccounts\_CMD Command bean.
- Update the ShowTransferAccountsResults JSP so that it retrieves the appropriate “to” and “from” account lists from the ShowAccounts\_CMD Command bean.
- Update the BankTasks Session bean to handle the transfer of money to a mortgage account.

### 10.7.1 Update the ShowAccounts\_CMD Command bean

In the class Commands.EJB.ShowAccounts\_CMD, the following changes were made:

- Added the methods getTo() and setToIndex() to separate the “to” and “from” account lists.
- Modified the method setIndex() to make the “from” account list not contain “MORTGAGE” accounts.
- Added the methods getAll() and setAllIndex() to provide a way to obtain a list of all accounts, regardless of account type.

See A.4.1, “Commands.EJB.ShowAccounts\_CMD.java” on page 290 for the full code listing for this updated class.

In the class Commands.ShowAccounts\_CMD, we added the methods getTo() and getAll() so that they will be exposed to the JSPs which need them. The Commands.ShowAccounts\_CMD class is a subclass of Commands.EJB.ShowAccounts\_CMD and provides the implementation of these methods.

See A.4.2, “Commands.ShowAccounts\_CMD.java” on page 294 for the full code listing for this updated class.

### **10.7.2 Update the ShowAccountsResults JSP**

In the ShowAccountsResults.jsp file we updated the loop for the accounts to use the property showAccounts\_CMD.getAll() instead of the getFrom() property

See A.4.3, “ShowAccountsResults.jsp” on page 295.

### **10.7.3 Update the ShowTransferAccountsResults JSP**

In the ShowTransferAccountsResults.jsp file, we updated the second loop for the "to" accounts to use the property showAccounts\_CMD.getTo() instead of the getFrom() property

See A.4.4, “ShowTransferAccountsResults.jsp” on page 298.

### **10.7.4 Update the BankTasks Session bean**

In the class com.ibm.ibmwebs.sample.BankTasksBean, the following change was made:

- Added some code in the transfer() method to send a payment message to the mps.mq.MPSOutbound enterprise bean.

This change is shown in 6.9.2, “Implementing the Consumer Banking Plus application” on page 146, and for a complete code listing for this class, see A.4.5, “com.ibm.ibmwebs.sample.BankTasksBean” on page 304.



## Chapter 11. Redirecting using OSE Remote

The WebSphere Advanced Edition server is designed to work closely with a Web server. The Web server can be located on the same machine as WebSphere, or it can be on a remote system. The most likely scenario where these two would be separated is the case where the WebSphere Advanced server will reside in a secure network, separated from the Web server by a firewall.

In order to split the Web server from WebSphere, a function must be implemented on the Web server to forward requests to WebSphere. This chapter will show you how to configure the OSE Remote feature (also referred to as Remote OSE), available in WebSphere Advanced V3.021.

### 11.1 Overview of the configuration

The system providing the Web server and OSE remote function will be referred to as the “redirector”, since its main purpose is to take requests and redirect them to the application servers in WebSphere. In our scenario, we have the redirector node, implemented using IBM HTTP Server and OSE Remote on a Windows NT 4.0 machine in the DMZ.

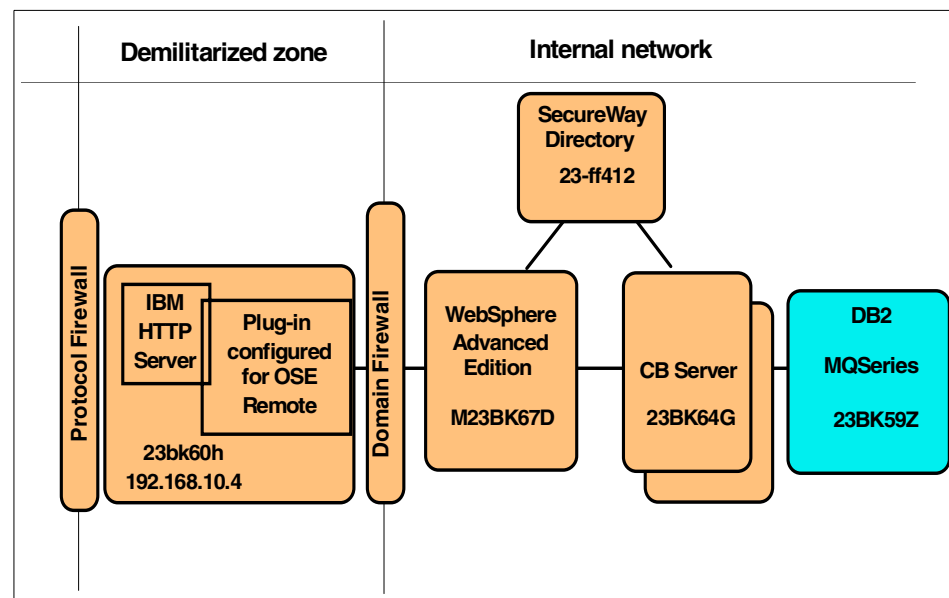


Figure 129. OSE Remote

To prepare the redirector node, you will need to install the following:

- A Web server

The following are supported by WebSphere Advanced Edition V3.021:

- IBM HTTP Server Version 1.3.6.2
- Apache Server Version 1.3.6
- Domino Version 5.02b
- Lotus Domino Go Webserver Version 4.6.2.5 or 4.6.2.6
- Microsoft Internet Information Server Version 4.0
- Netscape Enterprise Server Version 3.51 or Version 3.6.3

- WebSphere Advanced Edition, V3.021

When installing, use the custom installation path and choose the Production Application Server and the appropriate Web server plug-in.

---

## 11.2 Configuring the Web application server

The WebSphere Advanced Edition on the Web application server, “m23bk67d”, needs to be prepared for remote OSE communication. This involves two steps:

- Adding the host alias entries for the redirector node in the default\_host
- Configuring the transport type for the servlet engines

### 11.2.1 Adding host alias entries

Host alias entries must be added to the default\_host entry in WebSphere Advanced so the virtual host will accept requests redirected from the redirector, “23bk60h”. The alias list will include the host name, fully qualified host name, and IP address, in short, any type of addressing a client could use in a URL to address the Web server.

In our example, this means we need to add the following three entries for our Web server:

- 23bk60h
- 23bk60h.itso.ral.ibm.com
- 192.168.10.4

Host aliases can be added from the Administrative Console:

1. Click the **Topology** tab.

2. Click the **default\_host** in the Topology tree.
3. Click the **Advanced** tab. Add aliases in the **Host Aliases** and click the **Apply** button as shown in Figure 130.

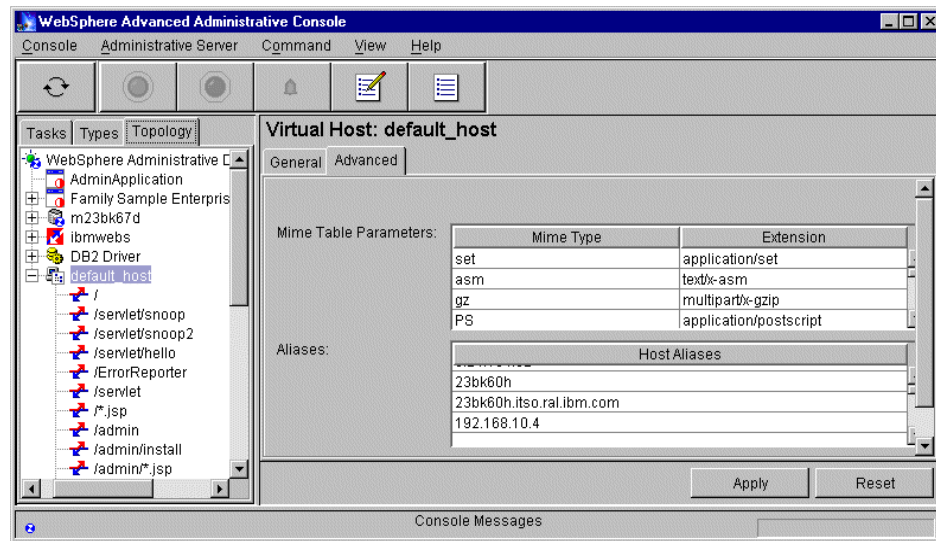


Figure 130. Adding an alias to the default\_host

After adding the alias, you need to restart all application servers using default\_host.

### 11.2.2 Configuring the servlet engine transport type

Each servlet engine needs to be configured to use sockets instead of local pipes. This is done by specifying INET Sockets for the transport type of the OSE queue. This is the default on Solaris but not on AIX or Windows NT.

1. From the Topology tag, select the servlet engine. In our example, we would select the FamilyServerServletEngine under the FamilyServer application server. Later, we would repeat these steps for the servletEngine in the Default Server.
2. Select the Advanced tag inside the servlet engine panel, and choose **OSE** for the Queue Type and click the **Settings** button.
3. You will get the Edit Servlet Engine Transport window. Then choose **INET Sockets** for Transport Type.

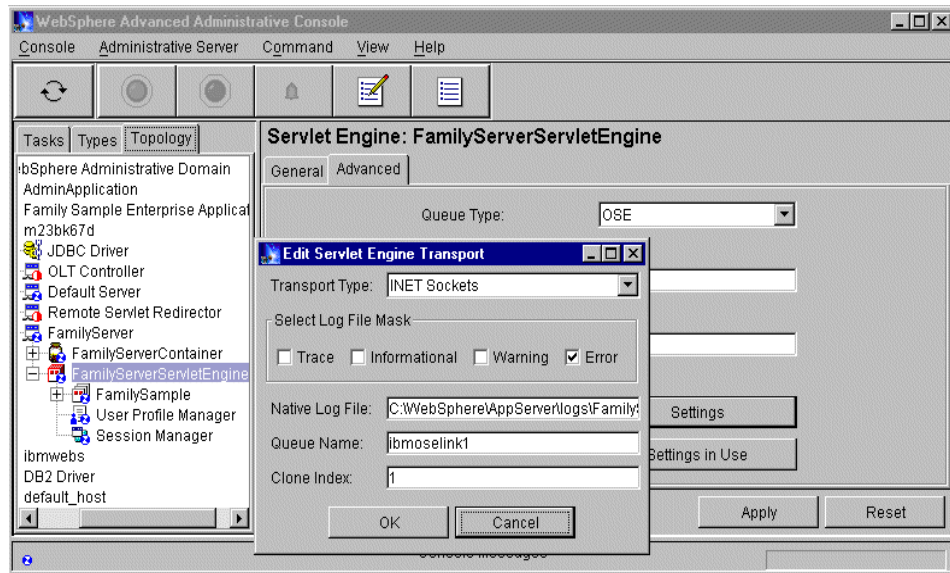


Figure 131. Transport Type Settings

4. Then click the **OK** button. You will be returned to the Administrative Console.
5. Click the **Apply** button.

### 11.3 Configure the plug-in for OSE Remote

The WebSphere plug-in for the Web server uses three properties files generated by WebSphere to define its configuration:

- queues.properties
- rules.properties
- vhosts.properties

These files need to be generated on the machine that will be used as the Web server.

#### Note

Any changes to the application server such as adding a Web application, servlet, or EJB, should be made now. Any changes you make after you generate the plug-in files require you to recreate them.

The plug-in can be configured two ways:

1. Configure the plug-in manually

Manual configuration requires that the files be copied from the WebSphere Advanced machine to the Web server machine. Some changes to the copied files are required. This is the preferred method if you are using WebSphere security or if the redirector is in the DMZ.

2. Configure the plug-in using a script

The script uses IIOP to connect to the administrative server to get the configuration information. This requires that IIOP traffic be allowed through any firewall that is between the Web server and WebSphere machines. Ports 900 (bootstrap) and 9000 (LSD) must be open on the firewall to run the plug-in configuration script.

Some changes to the generated files are required when using the script.

**Note:** Stop the Web server before configuring the plug-in.

### 11.3.1 Configure the plug-in manually

**Note**

This is the method you will need to use if you have application security turned on in WebSphere Advanced.

1. Copy the following files from WebSphere/AppServer/temp on the WebSphere Advanced machine, “m23bk67d”, to the same directory on the Web server machine, “23bk60h”.
  - vhosts.properties
  - rules.properties
  - queues.properties
2. Edit the queues.properties file on the redirector machine (“23bk60h”) and add an entry for the host the application servers reside on. This will be the name of the machine that is running the application server. There will be one queue for each application server.

The form of the entry is:

```
ose.srvgrp.<queue>.<clone>.host=system
```

```

#IBM WebSphere Plugin Communication Queues
#Thu May 18 14:25:03 EDT 2000
ose.srvgrp.ibmotelink.clonescount=1
ose.srvgrp.ibmotelink1.clonescount=1
ose.srvgrp.ibmotelink1.type=FASTLINK
ose.srvgrp=ibmoselink,ibmoselink1
ose.srvgrp.ibmotelink.type=FASTLINK
ose.srvgrp.ibmotelink.clone1.host=m23bk67d
ose.srvgrp.ibmotelink1.clone1.host=m23bk67d
ose.srvgrp.ibmotelink.clone1.port=8110
ose.srvgrp.ibmotelink1.clone1.port=8111
ose.srvgrp.ibmotelink1.clone1.type=remote
ose.srvgrp.ibmotelink.clone1.type=remote

```

Figure 132. <was\_dir>/temp/queues.properties for OSE Remote

You can correlate the entries above with the application server by looking at the servlet engine transport settings. In Figure 131, you can see that the FamilyServerServletEngine has a queue name of ibmoselink1. The Default Server servlet engine has a queue name of ibmoselink. There will also be entries for each clone. In this example we have only the application server (automatically designated as clone1), with no clones defined.

3. If you are using application security on the WebSphere advanced machine, you will need to modify the bootstrap file on both machines.

#### Note

These changes apply to port 8081. This port is used when the Web server is accessing a Web server resource (HTML) on the Web application server. Port 8110 is used when the resource is a WebSphere resource such as a JSP or servlet. If you are only securing WebSphere resources (no HTML) you do not need to do this.

The updates are different, so you will need to update each file individually (do not copy from one to the other).

c:\WebSphere\AppServer\properties\bootstrap.properties.

On the redirector, go to the bottom of the file and change the ose.srvgrp.ibmappserve.clone1.type from local to remote. Change

ose.srvgrp.ibmappserv.clone1.host from localhost to the host name of your WebSphere Advanced machine.

```
# Admin Server Properties
#
ose.adminqueue=ibmappserve
ose.max.concurrency=1
ose.srvgrp.ibmappserve.type=FASTLINK
ose.srvgrp.ibmappserve.clonescount=1
ose.srvgrp.ibmappserve.clone1.port=8081
ose.srvgrp.ibmappserve.clone1.type=remote
ose.srvgrp.ibmappserve.clone1.host=m23bk67d
```

Figure 133. Editing the bootstrap properties on the Web server redirector

On the WebSphere Advanced machine, change the ose.srvgrp.ibmappserv.clone1.type from local to remote.

```
# Admin Server Properties
#
ose.adminqueue=ibmappserve
ose.max.concurrency=1
ose.srvgrp.ibmappserve.type=FASTLINK
ose.srvgrp.ibmappserve.clonescount=1
ose.srvgrp.ibmappserve.clone1.port=8081
ose.srvgrp.ibmappserve.clone1.type=remote
ose.srvgrp.ibmappserve.clone1.host=localhost
```

Figure 134. Editing the bootstrap properties on the Web application server

#### 4. Stop and start WebSphere and the Web server.

You will need to repeat steps 1 and 2 every time the following changes are made to the WebSphere configuration:

- Adding a new URL (Web resource) to the environment
- Securing or unsecuring a URI (when you add security to a URI, an "A" is added to it, changing its name)
- Configuring a new virtual host alias
- Changing the queue properties of a servlet engine (name, port)
- Adding a new servlet engine
- Adding a new clone

### 11.3.2 Configure the plug-in using a script

**Note**

If you have security turned on in the application server, the batch file will not work. You will need to do the manual configuration.

To run the batch configuration, create a file (shell script for AIX or batch file for Windows NT) that sets the required environment variables and then runs the OSE Remote plug-in configuration program.

The configuration program uses information found in the WebSphere Advanced installation to build the temporary files needed by the plug-in. The file (we called it configOSE.bat) points to the WebSphere Advanced machine in two places:

- The -adminNodeName parameter
- The -nameServiceNodeName parameter

After you create and update the file you need to run it to generate the plug-in files. These temporary files (queues.properties, rules.properties, and vhosts.properties) will reside in the subdirectory <was\_dir>/temp on the WEB server machine.

The script requires the

`com.ibm.servlet.engine.oselister.systemsmgmt.OSERemotePluginCfg` class.

This is available in WebSphere Version 3.021.



```

@echo off
rem  configure OSE remote
setlocal
call setupCmdLine.bat
rem setup the classpath
set WAS_CP=%WAS_HOME%\lib\ibmwebas.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\properties
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\servlet.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\webtlsrn.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\lotusxsl.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ns.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ejb.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ujc.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\repository.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\admin.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\swingall.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\console.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\tasks.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\xml4j.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\x509v1.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\vaprt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\iioprt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\iioprttools.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\dertrjrt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\sslight.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ibmjndi.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\deployTool.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\databaseans.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\classes
set WAS_CP=%WAS_CP%;%JAVA_HOME%\lib\classes.zip
set CLASSPATH=%WAS_CP%

java com.ibm.servlet.engine.oselister.systemsmgmt.OSERemotePluginCfg
-traceString com.ibm.servlet.engine.*=all=enabled -tracefile
c:\websphere\appserver\logs\configRegen.log -serverRoot
c:\WebSphere\AppServer -adminNodeName m23bk67d
-nameServiceNodeName m23bk67d -nameServicePort 900
endlocal

```

Figure 135. configOSE.bat

**Note:** The line beginning with java.com.ibm.servlet should all be on one line. It is broken into multiple lines here for printing purposes.

Don't forget to start the Web server after the configuration is complete.

### Using the Remote script across a firewall

If you will be using the plug-in script when a firewall separates the redirector and WebSphere, you will also need to open port 900, port 9000, and a CORBA listener port. The listener port will be allocated dynamically unless you specify it directly. To do this, you need to add the parameter to the `-Dcom.ibm.CORBA.ListenerPort` to the line beginning with `com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs` in the `<was_dir>/bin/admin.config` file.

For example:

```
com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs=-mx128m  
-Dcom.ibm.CORBA.ListenerPort=33000
```

The line has been divided with a line break for easier reading. In your actual `admin.config` file, ensure that the line is on a single line. The administrative server has to be restarted for this to take effect.

---

## Chapter 12. Application security

In these scenarios, the applications were secured using the WebSphere Advanced Server application security features. Component Broker also offers application security. These options are discussed in 4.3, “Security” on page 37.

**Note:** Due to time constraints we did not implement Component Broker security using the Policy Director. All security was done at the WebSphere Advanced server by protecting the Web application. For information on Component Broker security, refer to *Application Server Solution Guide: Enterprise Edition*, SG24-5320.

---

### 12.1 Securing the WebSphere Advanced Web application

The next step in our example is to secure our application. The example will use basic authentication (user ID and password) and the IBM SecureWay directory as the user registry.

Defining security for the new application involves:

- Configuring the IBM Secureway Directory. We will not describe this here. If you are interested, you can see an example of this in *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864.
- Enabling global security in WebSphere and configuring it to use the SecureWay Directory.
- Configuring application security in WebSphere.

---

### 12.2 Enabling application security in WebSphere

WebSphere security is enabled and configured using the Security task. Using the WebSphere administrative console, switch to the Tasks tab and expand the security item.

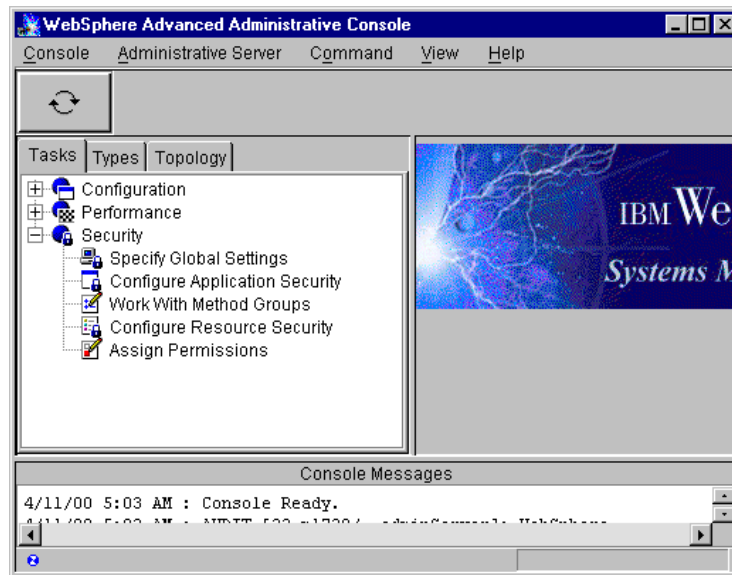


Figure 136. Security configuration task

There are five tasks listed under security:

- Specify Global Settings
- Configure Application Security
- Work with Method Groups
- Configure Resource Security
- Assign Permissions

Each of these tasks is designed to be performed in the order listed to enable WebSphere security.

---

## 12.3 Enabling WebSphere global security

The first step is to define the global security settings for WebSphere. At the completion of this step, the administrative server will be protected from unauthorized access.

#### Notes

- Once you have turned on global security and restarted the administration server, you must have a working security registry, using either LDAP or the operating system, in order to bring the administrative console back up. Do not perform this step until this is done!
- If you are running WebSphere Advanced on AIX, turning on global security will cause WebSphere to use SSL on the IIOP connection to Component Broker and you will need to enable SSL on that system.

In this example, an IBM SecureWay Directory LDAP server is going to serve as the user registry.

1. Make sure the LDAP server is running.
2. Bring up the WebSphere administrative console. Under the Tasks tab, highlight **Specify Global Settings** under the Security item. Then click the green light to start the security task.
3. In the following window, under the General tab, check the **Enable Security** checkbox.

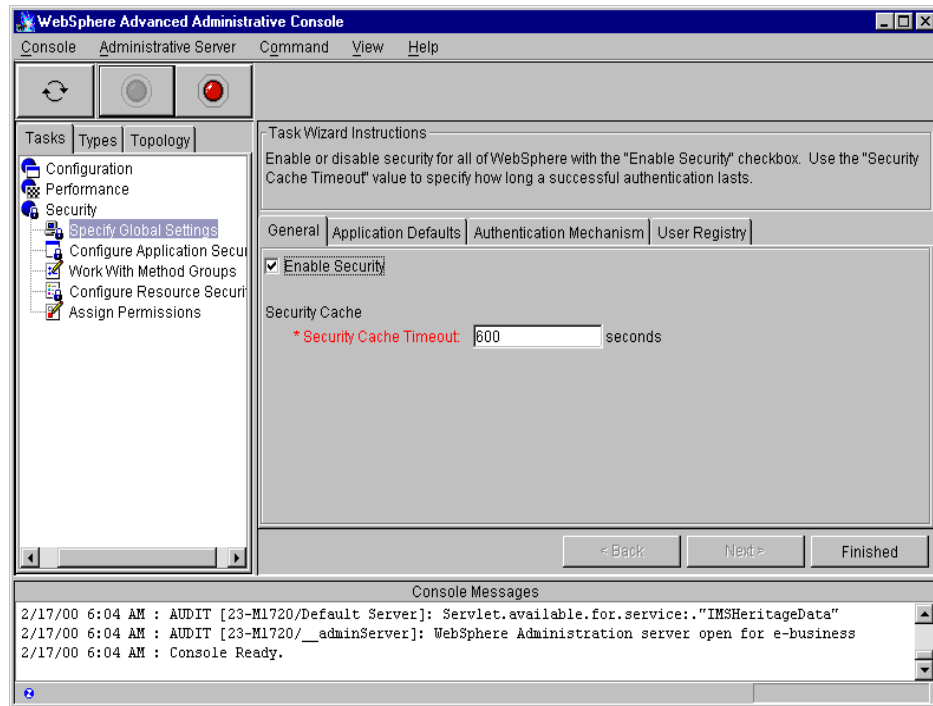


Figure 137. WebSphere global security settings

4. In the Application Defaults tab, a realm name is automatically entered. The security realm is the domain in which a security system operates. You can name the realm anything you like. All applications will need to belong to this security realm. Under Challenge Type, select the **Basic** radio button.

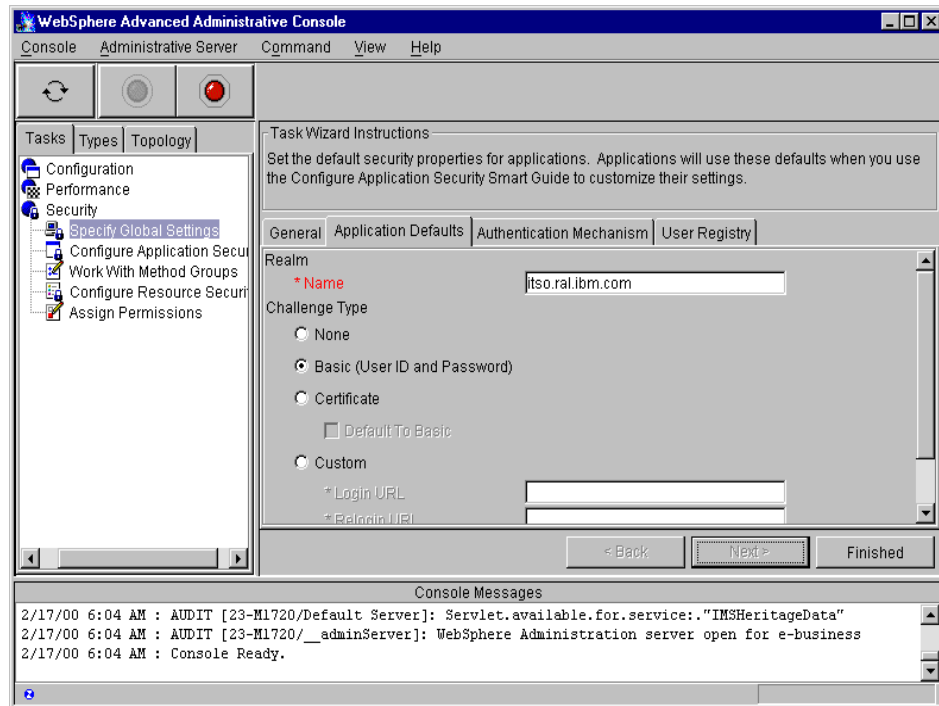


Figure 138. WebSphere security: application defaults

5. In the Authentication Mechanism tab, specify **Lightweight Third Party Authentication**. (For Lightweight Third Party Authentication (LTPA) testing use the default token expiration of 30 minutes.)

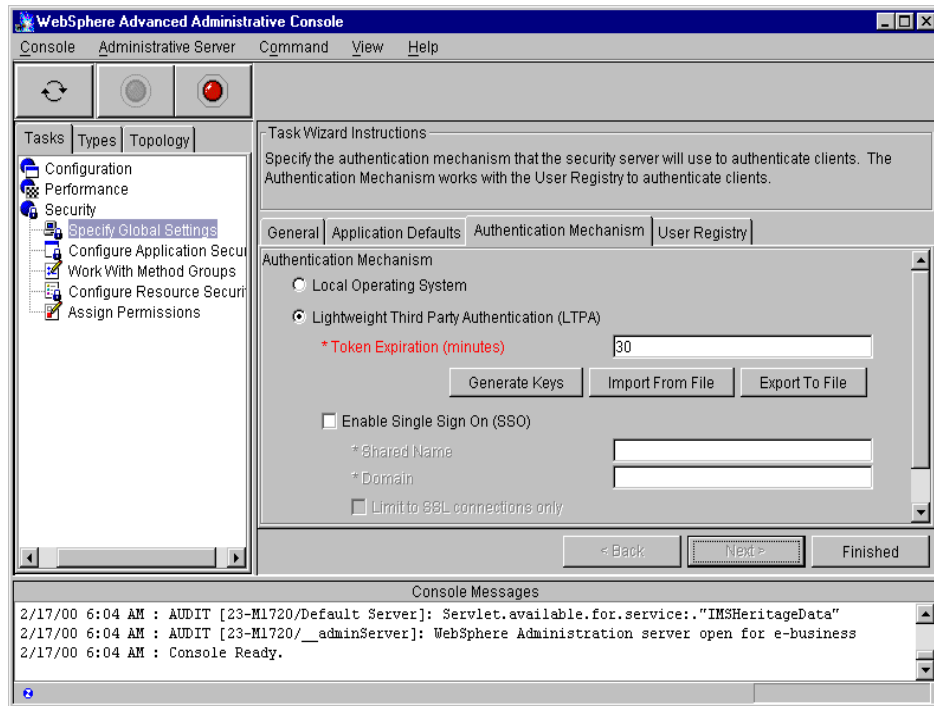


Figure 139. WebSphere security: authentication mechanism

6. Under the User Registry tab in the Global Settings for Security:

- Enter the top level administrator distinguished name (DN) for the security server ID (cn=USERID) and the corresponding password (PASSWORD).
- Choose SecureWay for the directory type.
- Enter the SecureWay Directory host name or IP address.
- Port 389 should be selected. This is the default for the IBM SecureWay Directory. If you changed this during SecureWay installation, you must reflect that here.
- Enter the base distinguished name that you want to use. The base DN identifies the point in the directory that you want to start searching. It could be the root of a tree in the directory (for example, o=ibm, c=us), or you could narrow the search down to a particular organizational unit, as we have done here.
- Enter cn=USERID for the bind DN.
- Enter the password for the bind DN in the bind password field.
- Click **Finished**.



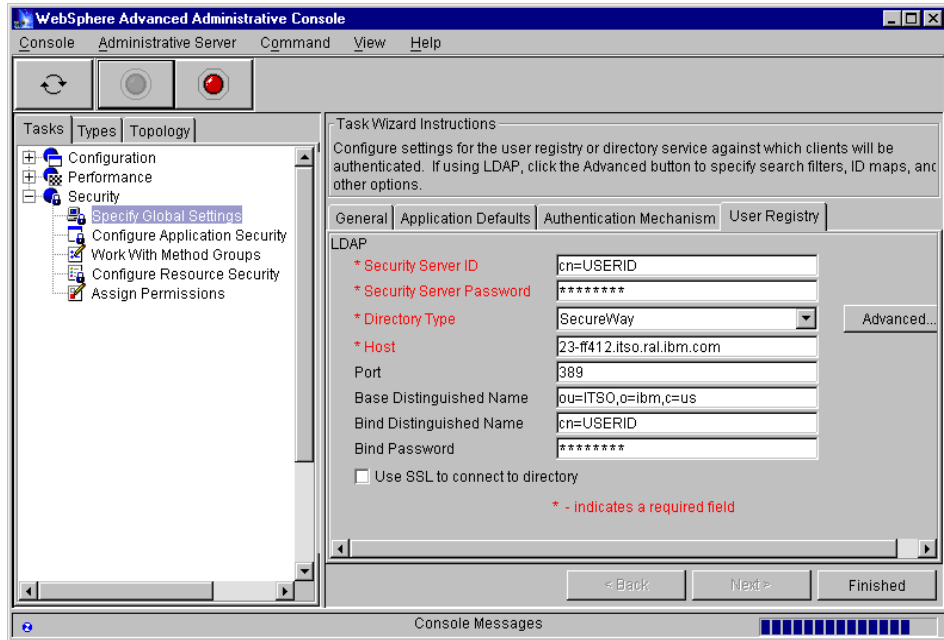


Figure 140. WebSphere security: user registry

You have now enabled security and specified the user registry to use. The administrative server is currently the only resource protected. The next step is to protect the application.

Activating global security requires stopping the administrative server and restarting it.

### 12.3.1 Protecting the application

In order to be protected by WebSphere security, a Web application has to be a part of an enterprise application. The first task will be to create an enterprise application that includes our example application.

1. Bring up the WebSphere administrative console. Under the Tasks tab, double click **Configuration** to expand it. Click **Configure an enterprise application**. Click the green light to start the task.
2. Give your application a name. Click **Next**.

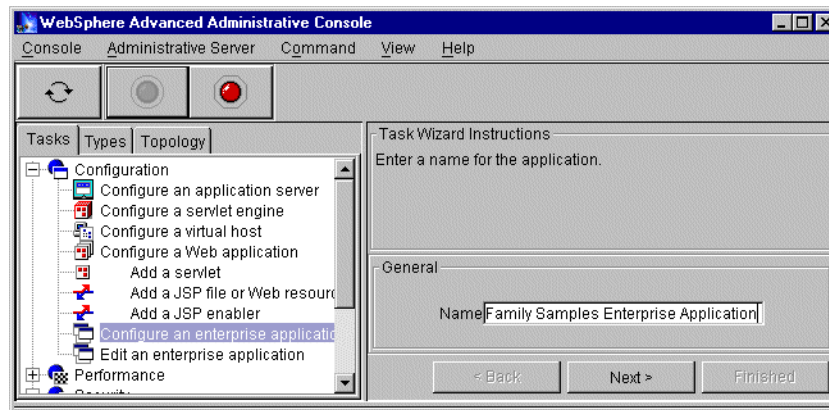


Figure 141. Configuring an enterprise application

3. On the next window add the Web application `FamilySample`. Click **Add**, **Next**, then **Finished**.

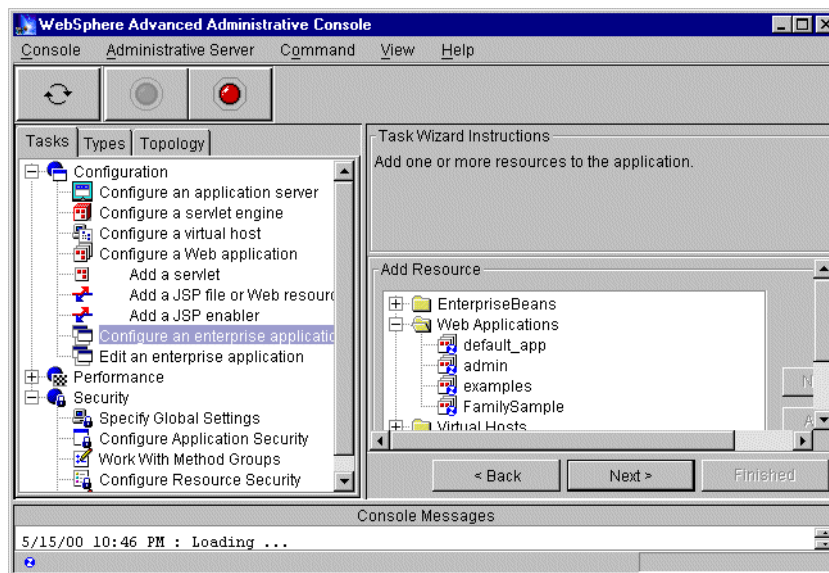


Figure 142. Adding resources to an enterprise application

4. Click **Finished**.

The next step in defining security is to configure application security. This is the next item in the GUI under security.

1. Highlight **Configure Application Security** and click the green start button.
2. Choose the enterprise application you just created and click **Next**.
3. You do not need to change anything on the next window but it is important that you complete this task and click **Finished**.

If you would like to define a new method group, you can do so with the next task, Work With Method Groups. For our example, this is not necessary.

You are now ready to configure the resource security. Without doing so, every application running on the application server will be accessible by anyone.

1. In the Tasks tab, click the next security task, **Configure Resource Security**. Click the green light to start that task.
2. On the right side of the window, expand Virtual Hosts and then default\_host.
3. From the list of resources, select the resource you want to configure. In our example we click **/FamilySample/servlet**. Click **Next**.

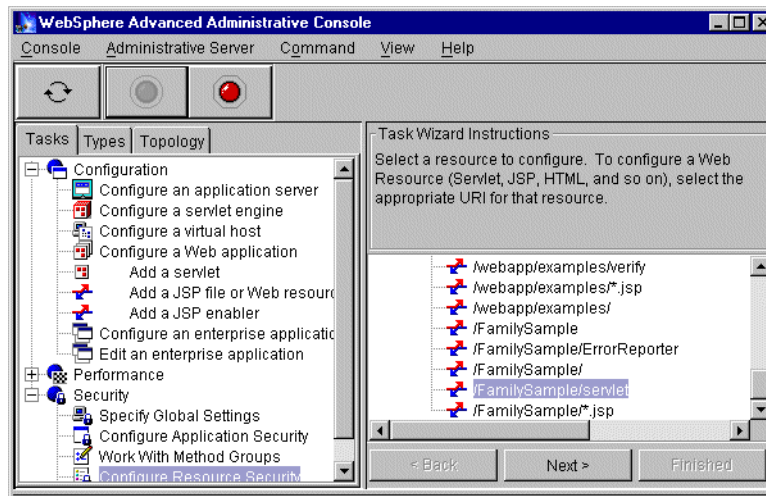


Figure 143. Selecting resources to be configured

4. When asked if you want to use the default method groups, click **Yes**.
5. When the next window comes up, click **Finished**.

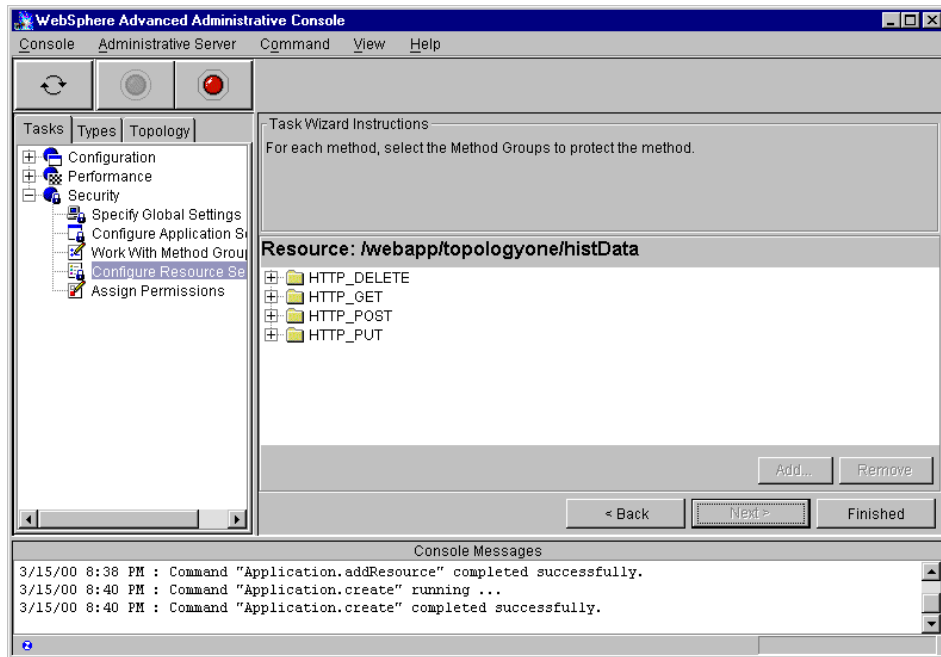


Figure 144. Adding default method groups to resources

You can repeat the steps described above as many times as needed. Each time configuring a different resource.

Now that the resources have been configured for security, you will assign permissions to the enterprise application you created before.

6. In the Tasks tab under Security, highlight **Assign Permissions**.
7. In the next window, select all entries that begin with `FamilySample`. You can do this by clicking the first entry and then, while pressing the Ctrl key, select the other entries one at a time. Then click **Add**.

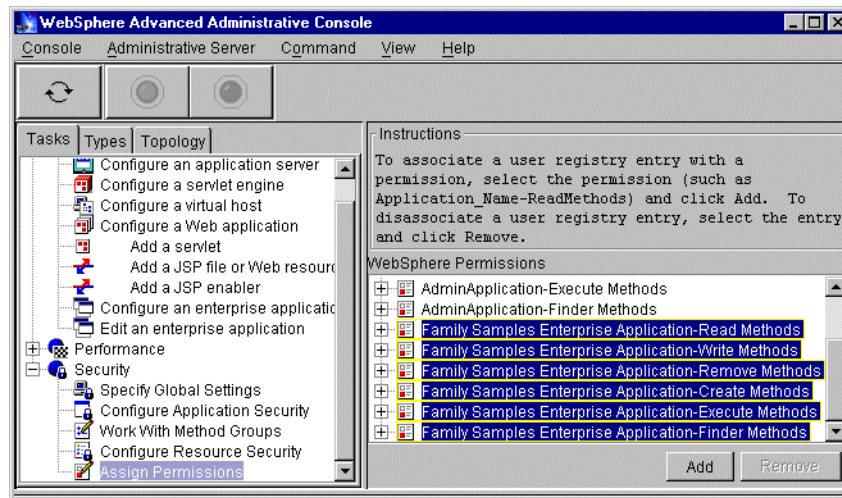


Figure 145. Selecting an application to be secured

8. Select **All Authenticated Users** and click on **OK**.

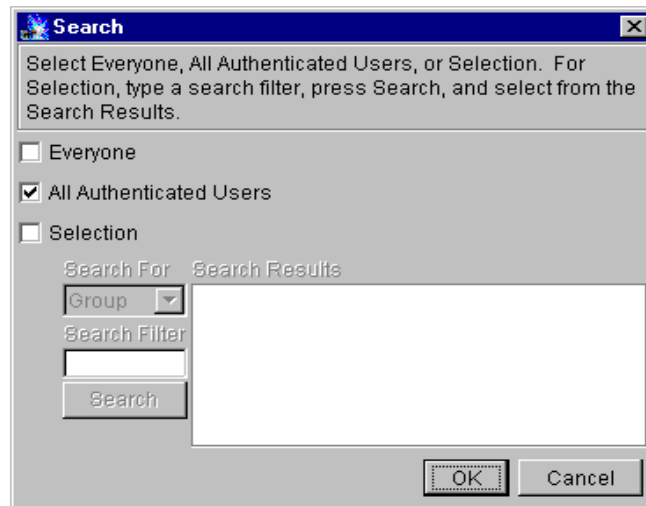


Figure 146. Selecting users who can access an application

The Family Sample Enterprise Application is now secure from unauthorized access.

You will need to stop and restart the administrative server task.

---

## 12.4 Hints and tips

If you turn on security using LDAP, the LDAP server must be available and the definitions must be correct. If you have made any mistakes, you will not be able to start the WebSphere administrative console. The IBM WS AdminServer service will start but the console GUI interface will fail to start. Unfortunately, you use the administrative console to correct the security definitions (if that is the problem). This is known as a catch-22.

**Tip:** If you use the task bar to start the administrative console the messages appear in a window that closes quickly. Open a command prompt window and enter `adminclient` to start the console manually. The advantage of this is that the error messages will not go away.

**Tip:** If you have a problem starting the administrative console you can turn the security feature off manually. Open the DB2 command line processor and enter the following commands:

```
connect to was
update ejssadmin.securitycfg_table set securityenabled = 0
commit
```

Stop and restart the IBM WS AdminServer service.

### Warning

Turning off security this way may not always work. Try this only if everything else fails. You may have to drop all applications from the database or even to reboot the machine.

---

## Chapter 13. Network security

This chapter will show the definitions used to set up the protocol and domain firewalls for our example. The firewall software used was IBM SecureWay Firewall for Windows NT V4.1. This is not an in-depth discussion but is designed to show firewall security experts what is needed specifically for this network environment. For more information on the IBM Firewall see the following two redbooks:

- *A Secure Way to Protect Your Network: IBM SecureWay Firewall for AIX V4.1*, SG24-5855-00
- *Guarding the Gates Using the IBM eNetwork Firewall V3.3 for Windows NT*, SG24-5209

Before you install the firewall software, please use the Planning and Network Configuration Planning Worksheet in *IBM SecureWay Firewall for Windows NT User's Guide V4.1*. When installing the software, follow the instructions in *IBM SecureWay Firewall for Windows NT Setup and Installation V4.1*.

After installation, there are seven steps to configure the firewall. The configuration is done using the SecureWay Firewall Configuration Client. The first time you start the configuration client, a setup wizard will take you through the configuration steps.

---

### 13.1 Determining the firewall ports to open

When first setting up a firewall, it may be helpful to initially allow all traffic to flow between the systems on either side. If there is some question about what ports are used, you can use the `netstat -a` command to list ports currently in use. Obviously, you will want to question what each port is used for before opening up the firewall to that port, but having a list of the ports that are in use is a good way to debug why connections are not being made later when the firewall rules are turned on.

---

### 13.2 Designating the network interfaces

This scenario consists of three network segments:

- The public network (outside world). This is usually the origin of the client requests. This network is considered to be non-secure.
- The Demilitarized Zone (DMZ). The DMZ is the network between the two firewalls. The DMZ is protected from the public network by a protocol

firewall, which limits the type of access that passes through the firewall to the nodes in the DMZ.

- The internal (or enterprise) network. The internal network is where the resources you want to protect reside. It is separated from the DMZ by a domain firewall that further limits traffic that has passed through the DMZ.

The first step is to define which network interfaces the firewall is using and if they are secure or not. There must be at least one secure and one non-secure interface.

### 13.2.1 Domain firewall

The domain firewall is situated between the DMZ, which uses IP addresses in the range of 192.168.10.xx, and the internal network, which uses addresses in the range of 9.24.104.xx. The domain firewall machine has two token-ring network adapters, with one adapter and IP address in each network. In this instance, the internal network is considered to be the secure network.

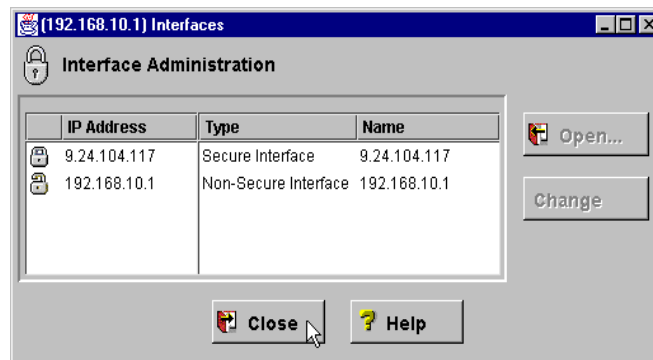


Figure 147. Configure interfaces for domain firewall

### 13.2.2 Protocol firewall

The protocol firewall is situated between the DMZ, which uses IP addresses in the range of 192.168.10.xx, and the external network, which uses addresses in the range of 172.16.0.xx. The protocol firewall machine has two token-ring network adapters, with one adapter and IP address in each network. In this instance, the DMZ is considered to be the secure network.



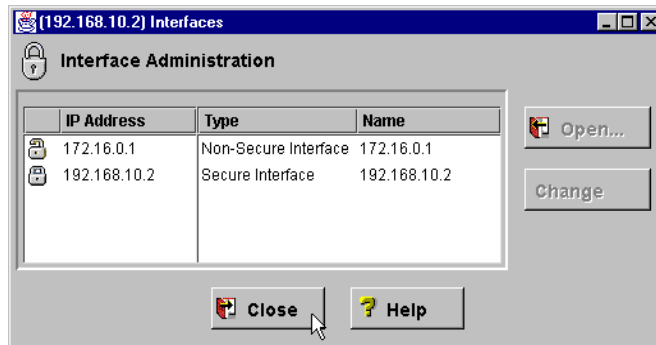


Figure 148. Configure interfaces for protocol firewall

### 13.3 Setting up the general security policy

The next step is to set up the general security policy. We have selected the following options as part of the general security policy on both firewalls.

- Permit DNS queries
- Deny broadcast messages
- Deny SOCKs

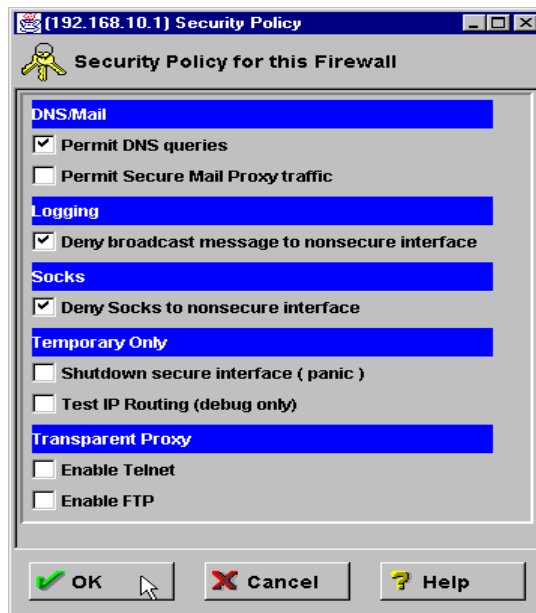


Figure 149. Security Policy configuration for both domain and protocol firewall

---

## 13.4 Creating the network objects

Next, you create the network objects. Network objects will be used as the source objects and/or the destination objects when you build your firewall connections. You can create either single objects (a host, firewall, router) or group objects (a group of single objects).

### 13.4.1 Domain firewall

In the domain firewall, we created a network object for each host that needed to communicate across the firewall.

| Object name       | IP address   | Mask            | Function                                 |
|-------------------|--------------|-----------------|--|
| Intranet-DNS-AIX  | 9.24.104.53  | 255.255.255.255 | Domain Name Server                       |
| Intranet-LDAP-NT  | 9.24.104.186 | 255.255.255.255 | SecureWay Directory                      |
| M23BK67D          | 9.24.104.62  | 255.255.255.255 | WebSphere Advanced in the secure network |
| Topo6-redirect or | 192.168.10.4 | 255.255.255.255 | Remote OSE redirector                    |
| WSA1              | 192.168.10.3 | 255.255.255.255 | WebSphere Advanced in the DMZ            |
| CB1               | 9.24.104.77  | 255.255.255.255 | Component Broker                         |

### 13.4.2 Protocol firewall

Figure 150 shows the network objects created for the protocol firewall.

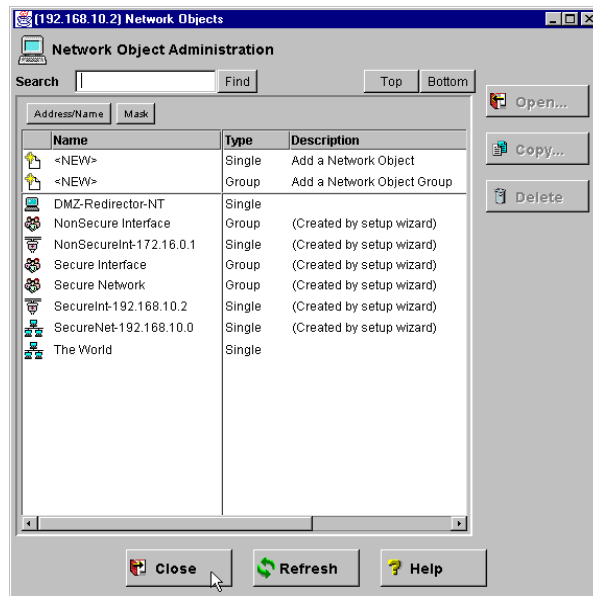


Figure 150. Network objects for the protocol firewall

## 13.5 Configuring the Domain Name Servers

The next step is to configure Domain Name Server information. We used the following configuration:

- Secure Domain Name - itso.ral.ibm.com
- Secure Domain Name Server - 9.24.104.53
- Non-Secure Domain Name Server - 192.168.10.99

## 13.6 Creating the firewall rules and services

The next task is to create the firewall rules and services. The steps to perform this are:

- Define the rules - The rules specify the protocol to be permitted or denied between two ports. They specify the direction of the traffic flow relative to the firewall interface.
- Define the services - A service is used to group common rules together.
- Define the connections - A connection defines the services that are allowed between two network objects.

### 13.6.1 Base topology

In the base topology, the WebSphere Advanced machine resides in the DMZ, while Component Broker is in the internal (secure) network. The Web application on the Advanced server is protected by LDAP, which sits in the secure network.

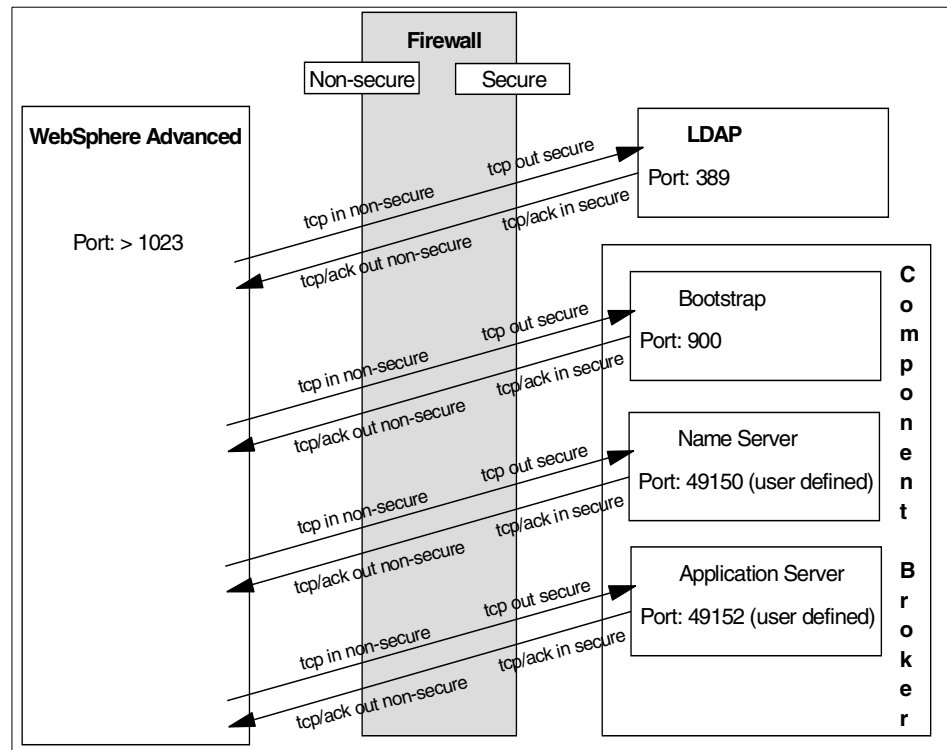


Figure 151. Base topology

Communication must exist between WebSphere and LDAP on port 389. Communication must exist between WebSphere and Component Broker on port 900, a port for the name server, and a port for each application server.

#### 13.6.1.1 Determining the CB ports

The ports used for the name server and application servers are assigned dynamically unless you specify them. To dynamically assign the ports on Windows NT, open the Windows NT Control Panel and choose **System**. Switch to the Environment tab. Add a system variable for each server you want to specify a port for.

The format for the variable name is:

`server_name_LISTEN_PORT`

or

`server_name_SSL_PORT`

where `server_name` is the exact name of the server but substituting an underscore for each “.” and space. The value is any valid unused TCP/IP port.

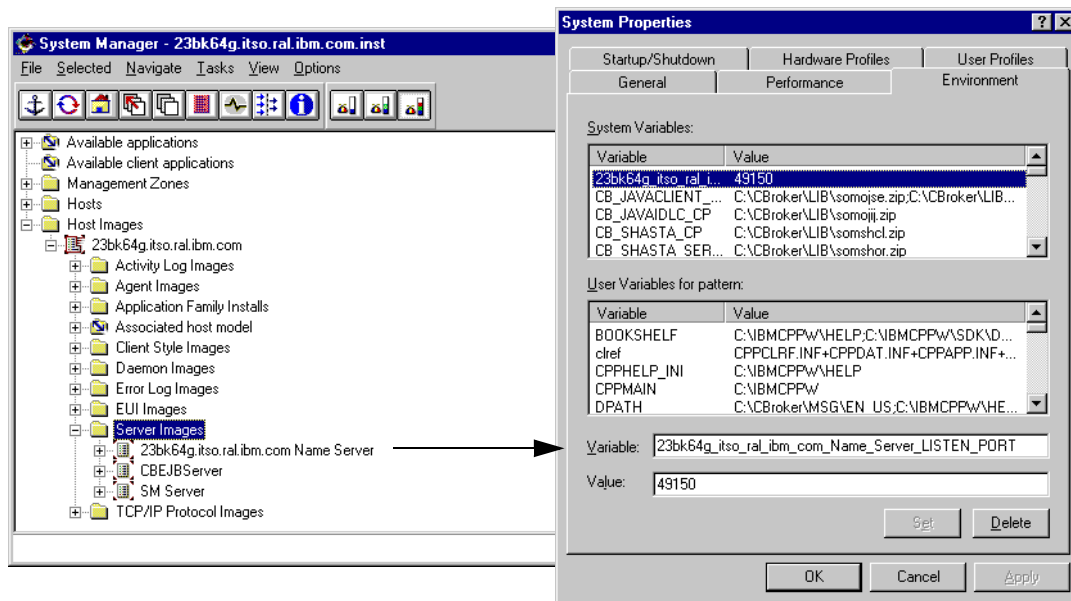


Figure 152. Defining the CB ports

For our environment we set the name server to port 49150 as shown in Figure 152. We also defined port 49152 for the CBEJBServer application server.

In future releases you will be able to change the port number in the server properties (ORB tab) using the CB System Manager.

### 13.6.1.2 Traffic between WebSphere and SecureWay Directory

The default port for SecureWay Directory is port 389. However, this port is configurable. See 12.3, “Enabling WebSphere global security” on page 238 for more information.

### 13.6.1.3 Service name: WAS1 to LDAP

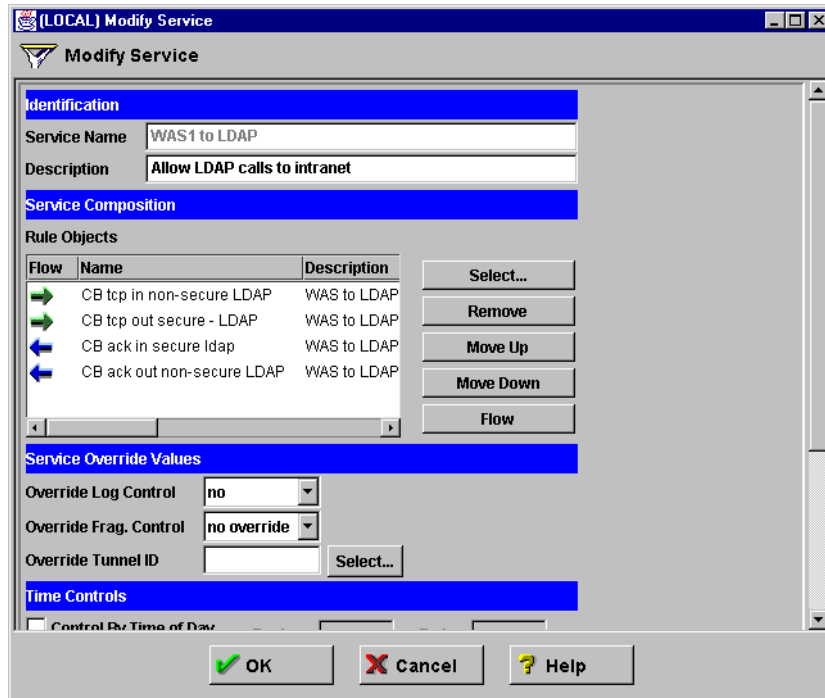


Figure 153. Service for WebSphere Advanced traffic to LDAP in domain firewall

#### Rule name 1: CB tcp in non-secure - LDAP

Table 3. Rule 1

| Action | Protocol | Operation / port at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|----------------------------|--------------------------|------------|---------|-----------|
| permit | tcp      | gt>1023                    | eq 389                   | non-secure | route   | inbound   |

#### Rule name 2: CB tcp out secure - LDAP

Table 4. Rule 2

| Action | Protocol | Operation / port at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|----------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp      | gt>1023                    | eq 389                   | secure    | route   | outbound  |

### Rule name 3: CB ack in secure - ldap

Table 5. Rule 3

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp/ack  | eq 389                       | gt 1023                  | secure    | route   | inbound   |

### Rule name 4: CB ack out non-secure LDAP

Table 6. Rule 4

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp/ack  | eq 389                       | gt 1023                  | non-secure | route   | outbound  |

#### 13.6.1.4 Service name: WAS1 to CB1

The connection between WebSphere Advanced and Component Broker will require a minimum of three ports: 900 for the bootstrap, a port for the name server, and a port for the application server. The ports for the name and application servers, 49150 and 49152 respectively, have been pre-defined in the Component Broker system.

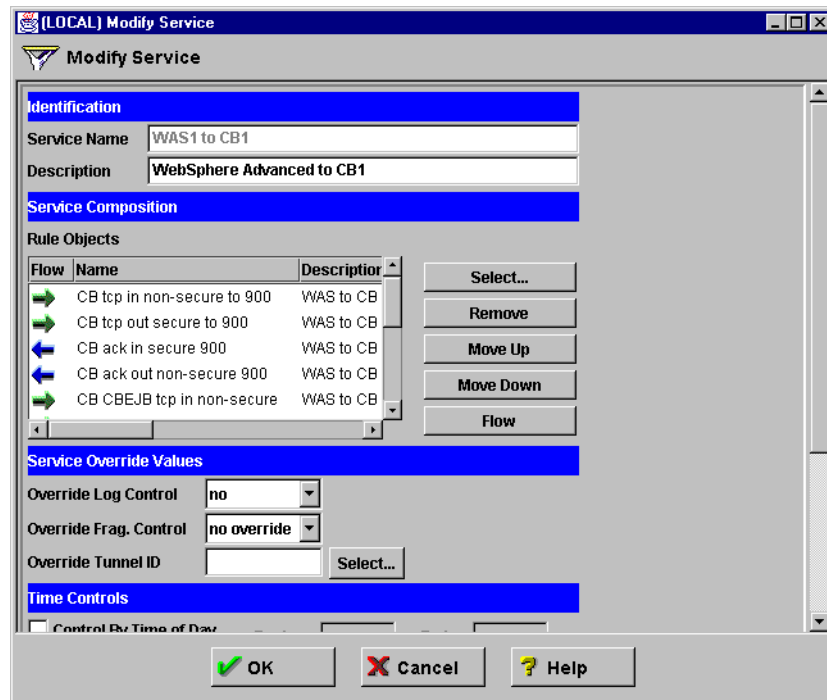


Figure 154. Service for WebSphere Advanced traffic to Component Broker in domain firewall

## CB Bootstrap Rules

### Rule name 1: CB tcp in non-secure 900

Table 7. Rule 1

| Action | Protocol | Operation/<br>port # at<br>source | Operation/<br>port at<br>dest | Interface  | Routing | Direction |
|--------|----------|-----------------------------------|-------------------------------|------------|---------|-----------|
| permit | tcp      | gt 1023                           | eq 900                        | non-secure | route   | inbound   |

### Rule name 2: CB tcp out secure 900

Table 8. Rule 2

| Action | Protocol | Operation/<br>port # at<br>source | Operation/<br>port at<br>dest | Interface | Routing | Direction |
|--------|----------|-----------------------------------|-------------------------------|-----------|---------|-----------|
| permit | tcp      | gt 1023                           | eq 900                        | secure    | route   | outbound  |



### Rule name 3: CB ack in secure 900

Table 9. Rule 3

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp/ack  | eq 900                       | gt 1023                  | secure    | route   | inbound   |

### Rule name 4: CB ack out non-secure 900

Table 10. Rule 4

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp/ack  | eq 900                       | gt 1023                  | non-secure | route   | outbound  |

## CB Application Server Rules

### Rule name 1: CB CBEJB tcp in non-secure

Table 11. Rule 1

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 49152                 | non-secure | route   | inbound   |

### Rule name 2: CB CBEJB tcp out secure

Table 12. Rule 2

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 49152                 | secure    | route   | outbound  |

### Rule name 3: CB CBEJB ack in secure

Table 13. Rule 3

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp/ack  | eq 49152                     | gt 1023                  | secure    | route   | inbound   |

### Rule name 4: CB CBEJB ack out non-secure

Table 14. Rule 4

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp/ack  | eq 49152                     | gt 1023                  | non-secure | route   | outbound  |

## CB Name Server Rules

### Rule name 1: CB NameServer tcp in non-secure

Table 15. Rule 1

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 49150                 | non-secure | route   | inbound   |

### Rule name 2: CB NameServer tcp out secure

Table 16. Rule 2

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 49150                 | secure    | route   | outbound  |

### Rule name 3: CB NameServer ack in secure

Table 17. Rule 3

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp/ack  | eq 49150                     | gt 1023                  | secure    | route   | inbound   |

### Rule name 4: CB NameServer ack out non-secure

Table 18. Rule 4

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp/ack  | eq 49150                     | gt 1023                  | non-secure | route   | outbound  |

#### 13.6.1.5 Connection between WebSphere and Component Broker

The connection identifies two network objects that are bound by one or more services. This example specifies that these rules are valid between two specific machines.

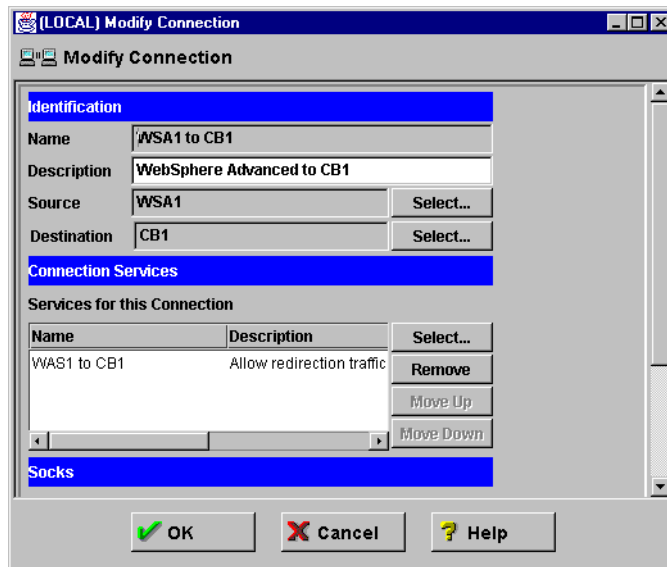


Figure 155. Connection from WebSphere Advanced to Component Broker

### 13.6.2 Variation 1

Variation 1 features a redirector in the DMZ, implemented by the OSE Remote feature of WebSphere Advanced. Ports will need to be opened for each application server that will be used and for the administrative server. The ports assigned for each WebSphere application server can be seen in the WebSphere administration console by clicking on each servlet engine and selecting the **Advanced** tab.

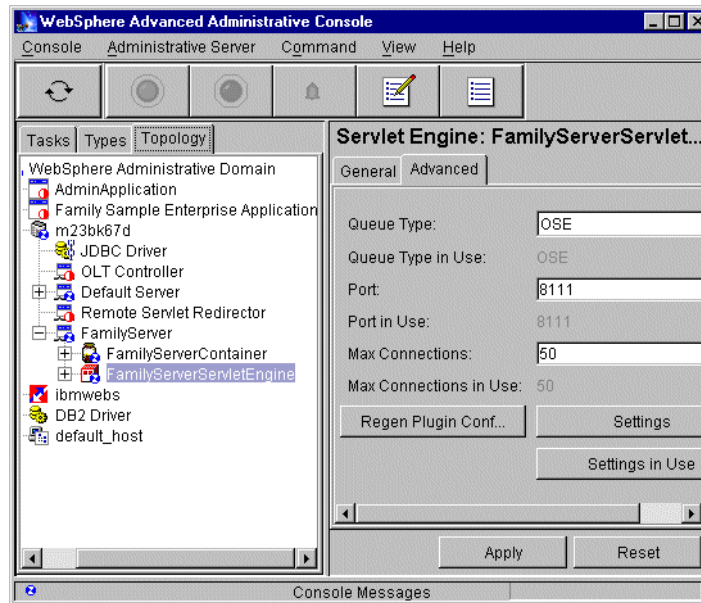


Figure 156. Determining ports for the servlet engines

If you will be using the plug-in script to create the rules files, you will also need to open port 900, port 9000, and a CORBA listener port. Specifying the CORBA listener port is discussed in 11.3.2, “Configure the plug-in using a script” on page 234. For our example, we are using WebSphere security so the plug-in script will not work for us and we don’t need to open these ports.

In this example, we will allow traffic to the default server (port 8110) and the FamilyServer application server installed with the sample (port 8111).

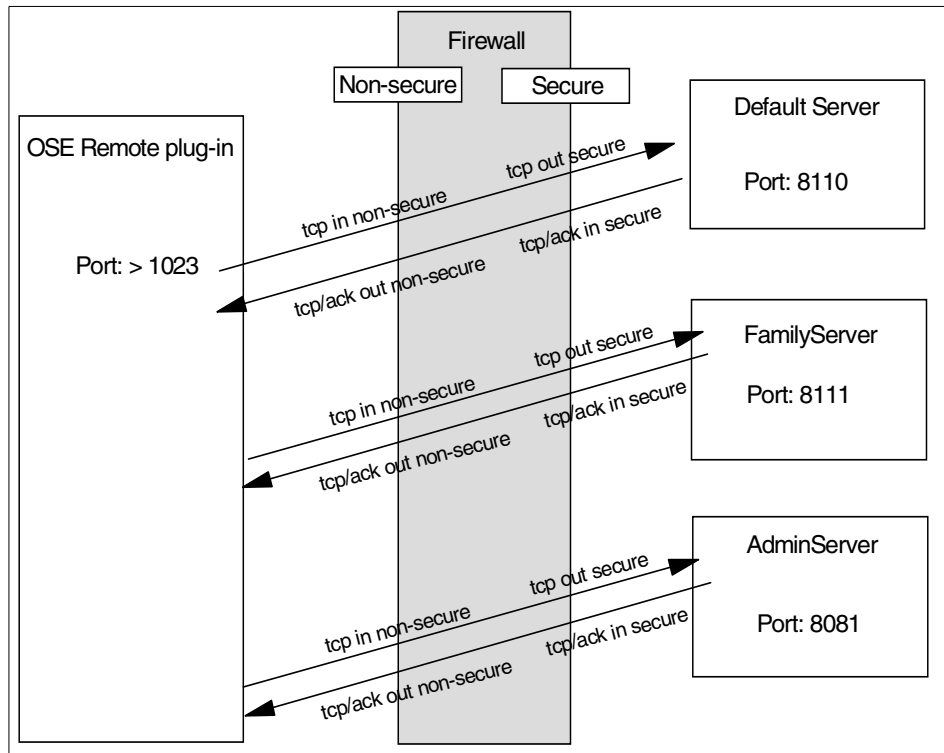


Figure 157. Variation 1 topology

### 13.6.3 Definitions for OSE Remote to WebSphere Advanced

In this example, we defined the services a little differently. Instead of lumping all the rules into one service, we grouped the rules for each port into a separate service. This does not matter; it is simply a matter of convenience. The first service is to open port 8081 to the WebSphere administrative server.

### 13.6.3.1 Service name: OSERemote-WASadmin

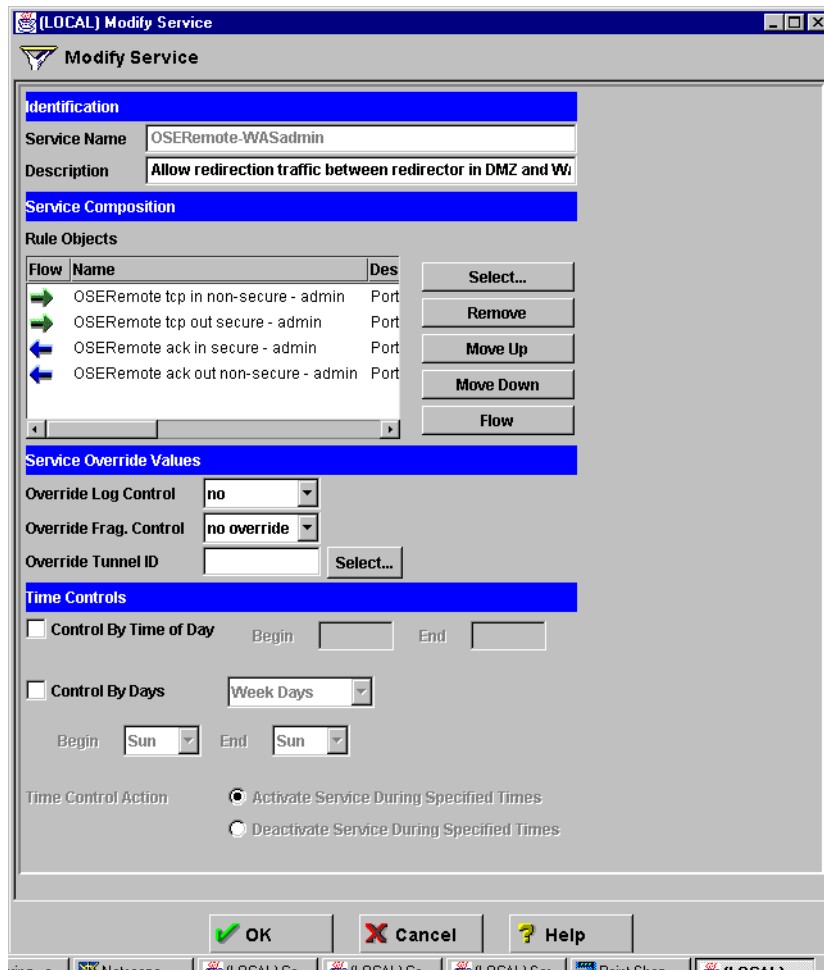


Figure 158. WAS-Bootstrap service in domain firewall

#### Rule name 1: OSERemote tcp in non-secure - admin

Table 19. Rule 1

| Action | Protocol | Operation /<br>port # at<br>source | Operation /<br>port at<br>dest | Interface  | Routing | Direction |
|--------|----------|------------------------------------|--------------------------------|------------|---------|-----------|
| permit | tcp      | gt 1023                            | eq 8081                        | non-secure | route   | inbound   |

### Rule name 2: OSERemote tcp out secure - admin

Table 20. Rule 2

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 8081                  | secure    | route   | outbound  |

### Rule name 3: OSERemote ack in secure - admin

Table 21. Rule 3

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp/ack  | eq 8081                      | gt 1023                  | secure    | route   | inbound   |

### Rule name 4: OSERemote ack out non-secure - admin

Table 22. Rule 4

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp/ack  | eq 8081                      | gt 1023                  | non-secure | route   | outbound  |

#### 13.6.3.2 Service name: OSERemote-WAServlet1

This service and set of rules allows traffic on port 8110, used by the WebSphere Advanced default\_server.

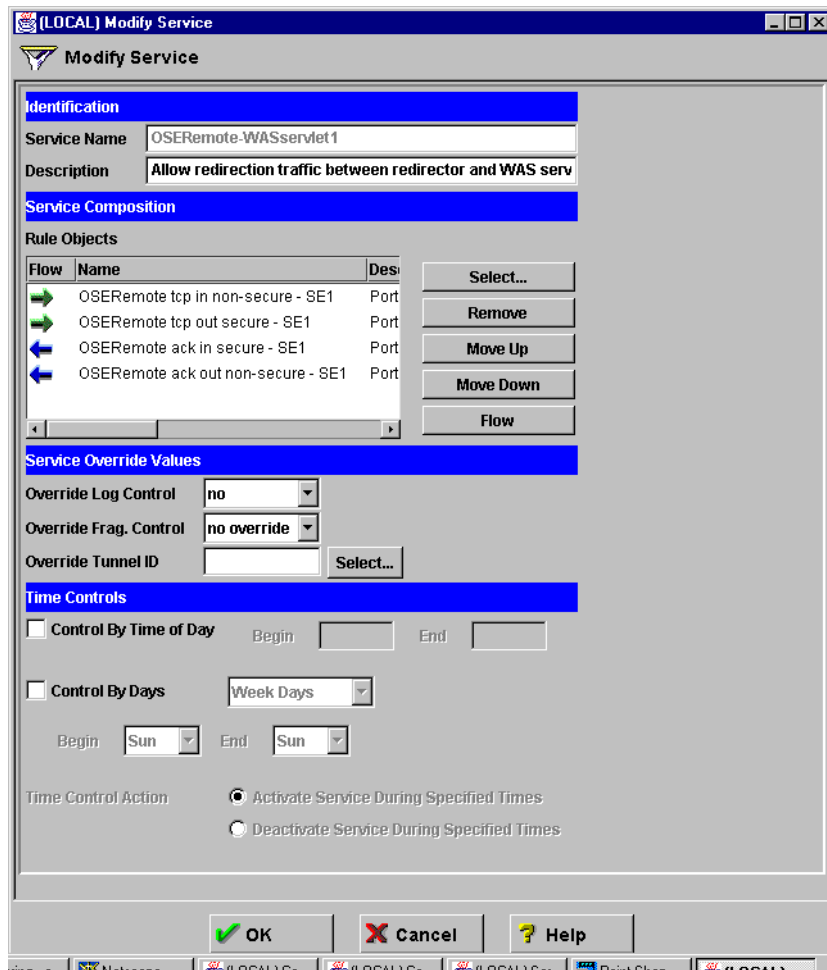


Figure 159. WAS-Bootstrap service in domain firewall

### Rule name 1: OSERemote tcp in non-secure - SE1

Table 23. Rule 1

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 8110                  | non-secure | route   | inbound   |



### Rule name 2: OSERemote tcp out secure - SE1

Table 24. Rule 2

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 8110                  | secure    | route   | outbound  |

### Rule name 3: OSERemote ack in secure - SE1

Table 25. Rule 3

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp/ack  | eq 8110                      | gt 1023                  | secure    | route   | inbound   |

### Rule name 4: OSERemote ack out non-secure - SE1

Table 26. Rule 4

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp/ack  | eq 8110                      | gt 1023                  | non-secure | route   | outbound  |

#### 13.6.3.3 Service name: OSERemote-WAServlet2

This service and set of rules allows traffic over port 8111, used by the FamilySample application server.

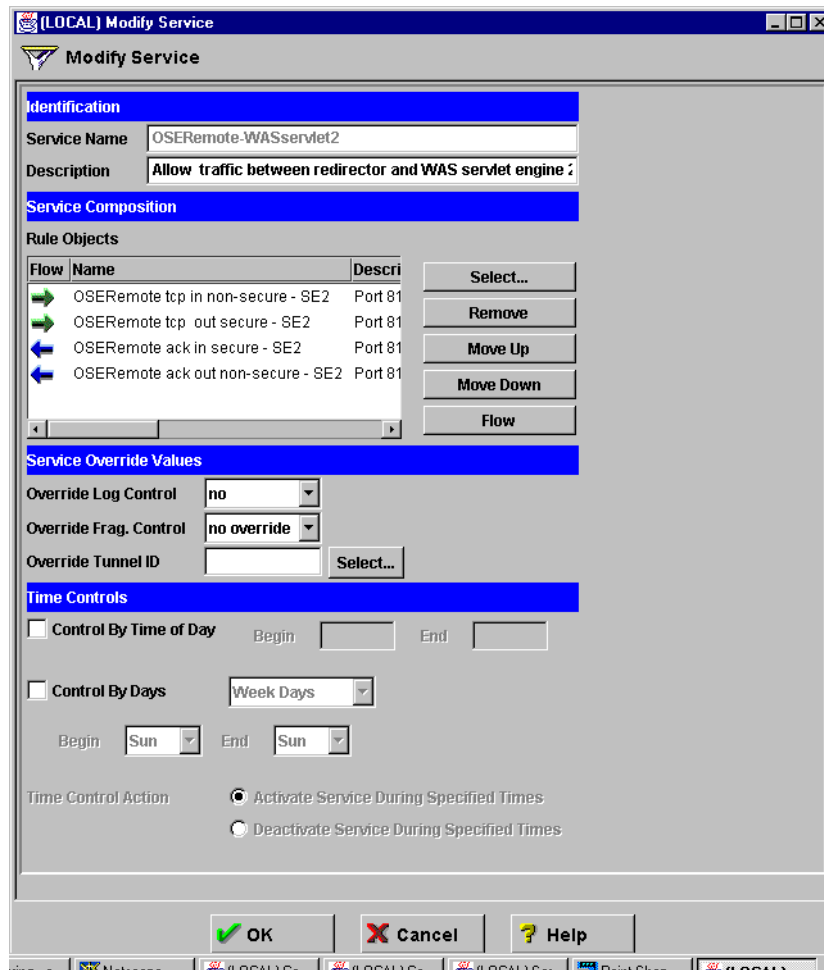


Figure 160. WAS-Bootstrap service in domain firewall

### Rule name 1: OSERemote tcp in non-secure - SE2

Table 27. Rule 1

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 8111                  | non-secure | route   | inbound   |

## Rule name 2: OSERemote tcp out secure - SE2

Table 28. Rule 2

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp      | gt 1023                      | eq 8111                  | secure    | route   | outbound  |

## Rule name 3: OSERemote ack in secure - SE2

Table 29. Rule 3

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface | Routing | Direction |
|--------|----------|------------------------------|--------------------------|-----------|---------|-----------|
| permit | tcp/ack  | eq 8111                      | gt 1023                  | secure    | route   | inbound   |

## Rule name 4: OSERemote ack out non-secure - SE2

Table 30. Rule 4

| Action | Protocol | Operation / port # at source | Operation / port at dest | Interface  | Routing | Direction |
|--------|----------|------------------------------|--------------------------|------------|---------|-----------|
| permit | tcp/ack  | eq 8111                      | gt 1023                  | non-secure | route   | outbound  |

### 13.6.3.4 Connection

A connection is defined between the OSE Remote machine and WebSphere Advanced, specifying the services just created.

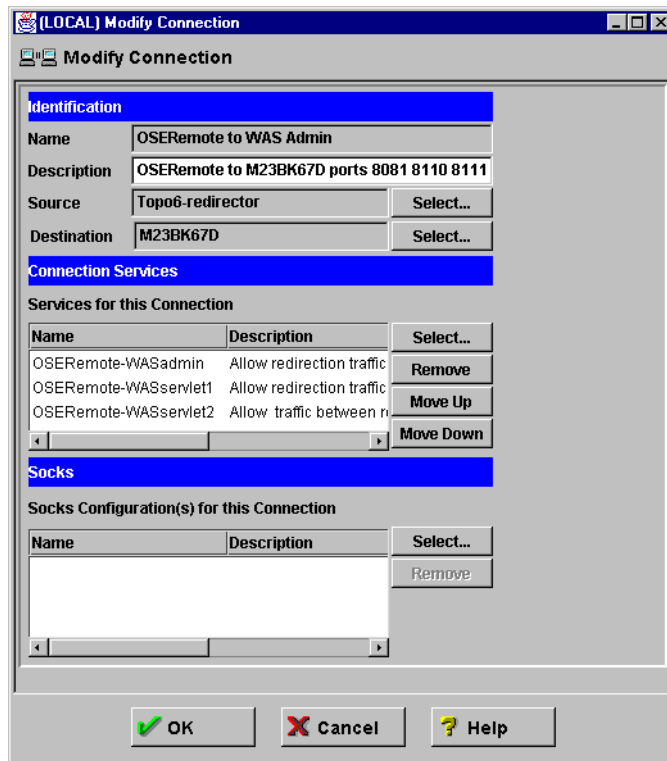


Figure 161. Connection between OSE Remote and WebSphere Advanced

---

## Appendix A. Sample source

This appendix includes text listings of sample source and configuration code used to build and implement the test application.

**Note:** Some lines of code are too long to fit in the given page width and are wrapped to the next line.

---

### A.1 MPSOutbound DO include()

```
::CORBA::Void MPSOutboundDOImpl_Impl::insert()
{
    // Source segment read from file "..\Source\do_insert.tde"
    try {
        cout << "MPSOutbound DOImpl queueName: " << iQueueName << endl;

        //-----
        // Step 1: Build the message data into iData. Store the length in iDataLength.
        //
        // !!!! Please insert the code to build the MQ message here !!!!

        cout << "account: " << iAccountId << endl;
        cout << "paymentAmount: " << iPaymentAmount << endl;

        const int iTextLength = 100;
        char iText[iTextLength];
        sprintf(iText, "%s %s", iAccountId, iPaymentAmount);
        cout << "iText: " << iText << endl;

        MQBYTE *iData = (MQBYTE *) (char *) iText;
        MQLONG iDataLength = iTextLength;

        //-----
        // Step 2: Build the ICBMQPut object.
        ICBMQPut putter( iQueueManagerName
                        , iQueueName
                        , iDataLength
                        , iData
                        );
    }
```

Figure 162. MPSOutbound DO include()

```

putter.format (MQFMT_STRING);

//-----
// Step 3: Copy the correlator to the ICBMPut Object. This is only needed if
//          this message is a reply to an earlier message.
if ( (iCorrelator) &&
      (strlen(iCorrelator) > 0)) {

    /** somehow iCorrelator is set to some unique value already,
    /** but causes an exception when it is read by serializeForm() below
    /** the exception is: 0x494205E0 IMQAAMinor_correlSerialVersionInvalid.
    /** commenting this out, fixes the problem ??

    //cout << "correlator being set to: " << iCorrelator << endl;
    //putter.correlator(iCorrelator);
}

//-----
// Step 4: Simply tell the ICBMPut object to put the message.
putter.put();

//-----
// Step 5: Retrieve the correlator and store it. It will be used by the home.
iCorrelator = putter.correlator().serializedForm();

}
catch (IBOIMException::IDataObjectFailed e)
{
    // One of the steps failed with IDataObjectFailed. Simply re-throw the exception.
    throw e;
}
catch (CORBA::UserException e)
{

```

Figure 163. MPSOutbound DO include()

```

        // A UserException has occurred. Convert it into an IDataObjectFailed exception.
        throw IBOIMEException::IDataObjectFailed("Object Builder"
                                                ,0
                                                ,0

, "::MPSOutboundMessageOutboundDOImpl::MPSOutboundMessageOutboundDOImpl::MPSOutboundMe
ssageOutboundDOImpl_Impl"
                                                , "insert() method caught an unexpected
CORBA::UserException."
                                                );
    }
    catch (CORBA::SystemException e)
    {
        // One of the steps failed with System Exception. Simply re-throw the exception.
        throw e;
    }
    catch (...)
    {
        // An unexpected exception occurred. The request was not successful.
        // Throw an IDataObjectFailed exception.
        throw IBOIMEException::IDataObjectFailed("Object Builder"
                                                ,0
                                                ,0

, "::PrinterDOImpl::PrinterDOImpl::ffic_printerDOImpl::PrintJobDOImpl_Impl"
                                                , "insert() method caught an unexpected
exception."
                                                );
    }
}
}

```

Figure 164. MPSOutbound DO include()

---

## A.2 MPService package (topology 5 implementation)

### A.2.1 PaymentServlet.java

```
package MPService;
/**
 * This file was generated by IBM WebSphere Studio Version 3.0.0
 *
 * e:\WebStudio\BIN\GenerationStyleSheets\AppServerV3\JSP0.91\WebSphere\JavaServlet.xml
 * stylesheet was used to generate this file.
 */
// Imports
import com.ibm.servlet.*;
import com.ibm.webtools.runtime.*;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import java.util.*;

public class PaymentServlet extends PageListServlet implements Serializable {

    private static final String CONFIG_BUNDLE_NAME = "WSFamily.FamilySampleStrings";
    PropertyResourceBundle configBundle = null;
    /* *****
    * Process incoming HTTP GET requests
    * @param request Object that encapsulates the request to the servlet
    * @param response Object that encapsulates the response from the servlet
    */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    {
        performTask(request, response);
    }
}
```

Figure 165. PaymentServlet.java



```

/* *****
 * Process incoming HTTP POST requests
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void doPost(HttpServletRequest request, HttpServletResponse response)
{
    performTask(request, response);
}
/* *****
 * Returns the requested parameter
 * @param request Object that encapsulates the request to the servlet
 * @param parameterName The name of the parameter value to return
 * @param checkRequestParameters when true, the request parameters are searched
 * @param checkInitParameters when true, the servlet init parameters are searched
 * @param isParameterRequired when true, an exception is thrown when the parameter
cannot be found
 * @param defaultValue The default value to return when the parameter is not found
 * @return The parameter value
 * @exception java.lang.Exception Thrown when the parameter is not found
 */
public String getParameter(HttpServletRequest request, String parameterName, boolean
checkRequestParameters, boolean checkInitParameters, boolean isParameterRequired,
String defaultValue) throws Exception
{
    String[] parameterValues = null;
    String paramValue = null;

//    Get the parameter from the request object if necessary.
    if (checkRequestParameters)
    {
        parameterValues = request.getParameterValues(parameterName);
        if (parameterValues != null)
            paramValue = parameterValues[0];
    }

//    Get the parameter from the servlet init parameters if
//    it was not in the request parameter.
    if ( (checkInitParameters) && (paramValue == null) )
        paramValue = getServletConfig().getInitParameter(parameterName);

```

Figure 166. *PaymentServlet.java - continued*

```

//      Throw an exception if the parameter was not found and it was required.
//      The exception will be caught by error processing and can be
//      displayed in the error page.
if ( (isParameterRequired) && (paramValue == null) )
    throw new Exception("Parameter " + parameterName + " was not specified.");

//      Set the return to the default value if the parameter was not found
if (paramValue == null)
    paramValue = defaultValue;

return paramValue;
}
/* *****
 * Process incoming requests for information
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void performTask(HttpServletRequest request, HttpServletResponse response)
{
    System.out.println(getClass().getName() + " performTask");
    String accountId = null;
    String paymentAmount = null;

    Commands.ErrorMsg em = new Commands.ErrorMsg();
    try
    {

        HttpSession session = request.getSession(true);
        System.out.println("MPS Payment: " + session.getId());

        // read in init values from configBundle
        if (configBundle == null) {
            configBundle =
(PropertyResourceBundle) PropertyResourceBundle.getBundle(CONFIG_BUNDLE_NAME);
        }
        session.putValue("nameServiceURLName",
                        "iiop://" +
                        configBundle.getString("WSE.Host") +
                        ":" +
                        configBundle.getString("WSE.Port"));
        session.putValue("nameServiceTypeName", "com.ibm.ejb.cb.runtime.CBCtxFactory");
    }
}

```

Figure 167. *PaymentServlet.java - continued*

```

        String [] names = session.getValueNames();
        if (names != null ) {
            for (int i = 0; i < names.length; i++) {
                System.out.println(session.getId() + " " + names[i] + ": " +
session.getValue(names[i]));
            }
        }

//    Clear for restart
session.removeValue("accountId");
session.removeValue("paymentAmount");
session.removeValue("payment_CMD");

em.setError(false);
session.putValue("errorMsg",em);
Payment_CMD payment_CMD = null;

accountId = getParameter(request, "accountId", true, true, true, null);
System.out.println(session.getId() + " MPS PaymentServlet - accountId: " +
accountId);

paymentAmount = getParameter(request, "paymentAmount", true, true, true, null);
System.out.println(session.getId() + " MPS PaymentServlet - paymentAmount: " +
paymentAmount);

payment_CMD = (Payment_CMD) Util.instance().loadCommand(session,"Payment");

//    Call the perform action on the bean.
payment_CMD.setAccountId(accountId);
payment_CMD.setPaymentAmount(paymentAmount);

System.out.println(getClass().getName() + " perform");
payment_CMD.perform();
//    Make sure the command completes
session.putValue("accountId", accountId);
session.putValue("paymentAmount", paymentAmount);
session.putValue("payment_CMD", payment_CMD);
System.out.println(session.getId() + " MPSPaymentServlet session after
perform");

```

Figure 168. PaymentServlet.java - continued

```

        names = session.getValueNames();
        if (names != null ) {
            for (int i = 0; i < names.length; i++) {
                System.out.println(session.getId() + " " + names[i] + ": " +
session.getValue(names[i]));
            }
        }

//      Call the output page. If the output page is not passed
//      as part of the URL, the default page is called.
callPage(getPageNameFromRequest(request), request, response);
//      Generic error handling for Commands
} catch (Exception cf) {
    try {
        em.setError(true);
        em.setMessage(cf.getMessage());
        callPageNamed(cf.getClass().getName(),request, response);
    } catch (Throwable e) {
        handleError(request, response, e);
    }
}
}
}
}

```

Figure 169. *PaymentServlet.java - continued*

## A.2.2 PaymentServlet.servlet

```

<?xml version="1.0"?>
<servlet>
  <page-list>
    <default-page>
      <uri>/FamilySample/MPService/MPSPaymentSubmitted.jsp</uri>
    </default-page>
    <error-page>
      <uri>/FamilySample/MPService/MPSPaymentError.jsp</uri>
    </error-page>
  </page-list>
  <code>MPService.PaymentServlet</code>
</servlet>

```

Figure 170. *PaymentServlet.servlet*

### A.2.3 Util.Java

```
package MPService;

import javax.servlet.*;
import javax.servlet.http.*;

public class Util {
    private static Util mySingleInstance = null;
    public static Util instance() {
        System.out.println("Util.instance");
        if (mySingleInstance == null) {
            mySingleInstance = new Util();
        }
        return mySingleInstance;
    }
    public CMD loadCommand(HttpSession session,
                           String className) {
        System.out.println("Util.loadCommand");
        try {
            CMD obj = null;
            String fullName = "MPService." + className + "_CMD";
            System.out.println("Util loading class: " + fullName);
            obj = (CMD) java.beans.Beans.instantiate(getClass().getClassLoader(),
fullName);
            obj.setSession(session);
            System.out.println("Util return obj " + obj.getClass().getName());

            return obj;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Figure 171. Util.java

## A.2.4 CMD.java

```
package MPService;

import javax.servlet.*;
import javax.servlet.http.*;

public class CMD extends java.lang.Object implements java.io.Serializable
{
    protected HttpSession session;

    public CMD() {}
    public String hostName() {
        return (String)session.getValue("host");
    }
    public String nameServiceTypeName() {
        return (String)session.getValue("nameServiceTypeName");
    }
    public String nameServiceURLName() {
        return (String)session.getValue("nameServiceURLName");
    }
    public String port() {
        return (String)session.getValue("port");
    }
    public void setSession(HttpSession session) {
        this.session = session;
    }
}
```

Figure 172. CMD.java

## A.2.5 Payment\_CMD.java

```
package MPService;

public class Payment_CMD extends CMD {
    String accountId;
    String paymentAmount;

    private mps.mq.MPSOutboundAccessBean mpsOutbound;
    private transient javax.transaction.UserTransaction currentTransaction;
    private transient javax.naming.InitialContext jndiContext;

    /*
     * begin transaction
     */
    private boolean beginTransaction() {

        if (currentTransaction == null) {
            try {
                currentTransaction = (javax.transaction.UserTransaction)
jndiContext.lookup("jta/usertransaction");

                if (currentTransaction == null) throw new
NullPointerException("currentTransaction");
                System.out.println(">> transaction found");
            }
            catch (javax.naming.NamingException e) {
                ejb.Util.printStackTrace("MPService.Payment_CMD", "beginTransaction", e);
                return false;
            }
            catch (NullPointerException e) {
                ejb.Util.printStackTrace("MPService.Payment_CMD", "beginTransaction", e);
                return false;
            }
        }

        try {
            currentTransaction.begin();
            System.out.println(">> transaction started..");
            return true;
        }
    }
}
```

Figure 173. Payment\_CMD.java

```

catch (java.lang.Throwable e) {
    ejb.Util.printStackTrace("MPService.Payment_CMD", "beginTransaction", e);
    return false;
}
}
/*
 * commit transaction
 */
private boolean commitTransaction() {

    if (currentTransaction == null) {
        System.out.println(">> no trasaction exists to commit");
        return false;
    }

    try {
        currentTransaction.commit();
        System.out.println(">> transaction committed..");
        return true;
    }
    catch (java.lang.Throwable e) {
        ejb.Util.printStackTrace("MPService.Payment_CMD", "commitTransaction", e);
        return false;
    }

}

public String getAccountId() {
    return accountId;
}

public String getPaymentAmount() {
    return paymentAmount;
}
}

```

Figure 174. *Payment\_CMD.java - continued*



```

//Perform Method
public void perform() throws MPSTransactionException {

    System.out.println(getClass().getName() + " perform");
    try {
        if (mpsOutbound == null) {
            System.out.println(getClass().getName() + " initializing mpsOutbound
AccessBean");

            String homeName = "mps/mq/MPSOutboundEJBHome";

            mpsOutbound = new mps.mq.MPSOutboundAccessBean();
            mpsOutbound.setInit_JNDIName(homeName);

            if (nameServiceTypeName() != null) {
                System.out.println(getClass().getName() + " nameServiceTypeName " +
nameServiceTypeName());
                mpsOutbound.setInit_NameServiceTypeName(nameServiceTypeName());
            }
            if (nameServiceURLName() != null) {
                System.out.println(getClass().getName() + " nameServiceURLName " +
nameServiceURLName());
                mpsOutbound.setInit_NameServiceURLName(nameServiceURLName());
            }
        }

        if (jndiContext == null) {
            // the only reason we call getInitialContext() directly is because we need
            // the jndiContext to find and start a UserTransaction,
            // otherwise, the InitialContext would be initialized when we
            // make the first call on the AccessBean - mpsOutbound.put()
            this.jndiContext = mpsOutbound.getInitContext(nameServiceURLName(),
nameServiceTypeName());
        }

        // begin transaction
        beginTransaction();
    }
}

```

Figure 175. *Payment\_CMD.java - continued*

```

// create mpsOutbound message template
String queueName = "MPS.QUEUE";

mps.mq.MPSMsgTemplate template = new mps.mq.MPSMsgTemplate();
template.setQueueName(queueName);
template.setAccountId( getAccountId() );
template.setPaymentAmount( getPaymentAmount() );

// put the message
System.out.println(getClass().getName() + " sending MPS payment - accountId: " +
getAccountId() + " paymentAmount: " + getPaymentAmount());
String correlatorId = mpsOutbound.put(template);
System.out.println(getClass().getName() + "message put successfully");

// commit transaction
commitTransaction();

} catch (org.omg.CORBA.OBJECT_NOT_EXIST one) {
    rollbackTransaction();
    throw (new MPSTransactionException("Failed to connect to server, it may be
down...", getAccountId(), getPaymentAmount()));
} catch (java.lang.Exception e) {
    rollbackTransaction();
    System.out.println(e);
    throw (new MPSTransactionException(e.getMessage(), getAccountId(),
getPaymentAmount()));
}
}
/*
 * rollback transaction
 */
private boolean rollbackTransaction() {

    if (currentTransaction == null) {
        System.out.println(">> no transaction exists to rollback");
        return false;
    }
}

```

Figure 176. *Payment\_CMD.java - continued*

```
        try {
            currentTransaction.rollback();
            System.out.println(">> transaction rolled back..");
            return true;
        }
        catch (java.lang.Throwable e) {
            ejb.Util.printStackTrace("MPService.Payment_CMD", "rollbackTransaction", e);
            return false;
        }
    }
    public void setAccountId(String id) {
        accountId = id;
    }
    public void setPaymentAmount(String amount) {
        paymentAmount = amount;
    }
}
```

*Figure 177. Payment\_CMD.java - continued*

## A.2.6 MPSTransactionException.java

```
package MPService;

public class MPSTransactionException extends Exception {
    public String accountId;
    public String paymentAmount;
    /**
     * MPSTransactionException constructor
     */
    public MPSTransactionException() {
        super();
    }
    /**
     * MPSTransactionException constructor
     * @param s java.lang.String
     */
    public MPSTransactionException(String s) {
        super(s);
    }
    /**
     * MPSTransactionException constructor.
     * @param s java.lang.String
     */
    public MPSTransactionException(String s, String accountId, String paymentAmount) {
        super(s);
        this.accountId = accountId;
        this.paymentAmount = paymentAmount;
    }
    /**
     * Returns a String that represents the value of this object.
     * @return a string representation of the receiver
     */
    public String toString() {
        return (getClass().getName() + "[accountId: " + accountId + " paymentAmount: " +
        paymentAmount + "] " + getMessage());
    }
}
```

Figure 178. MPSTransactionException.java

## A.3 Mortgage Payment System JSPs

### A.3.1 MPSPayment.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"><!-- Sample HTML file -->
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE>MPS Payment</TITLE>
<SCRIPT language="JavaScript" src="/FamilySample/WSFamily/validation.js"></SCRIPT>
<LINK href="/FamilySample/theme/WSFamily.css" rel="stylesheet" type="text/css">
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<CENTER>
<TABLE bgcolor="#FFFFFF" background="/FamilySample/spi_005.gif" border="1">
  <CAPTION></CAPTION></TABLE>
<H2>Mortgage Payment Service</H2>
<TABLE bgcolor="#FFFFFF" background="/FamilySample/spi_005.gif" border="1">
  <TBODY>
    <TR>
      <TD align="left" bgcolor="#ffff80" background="/FamilySample/spi_005.gif">
        <FORM action="/FamilySample/servlet/MPService.PaymentServlet" method="POST">
          <TABLE border="1" bgcolor="#FFFFFF" width="354"
background="/FamilySample/spi_005.gif">
            <TBODY>
              <TR>
                <TD align="right" width="98"><B>Customer AccountId</B></TD>
                <TD><INPUT size="38" type="text" name="accountId"></TD>
              </TR>
              <TR>
                <TD><B>Payment Amount</B></TD>
                <TD><INPUT size="38" type="text" name="paymentAmount"></TD>
              </TR>
              <TR>
                <TD align="right"><INPUT type="submit" value="Submit">
                </TD>
                <TD align="left" width="266"><INPUT type="reset" value="Reset"></TD>
              </TR>
            </TBODY>
          </TABLE>
        </FORM>
      </TD>
    </TR>
  </TBODY>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Figure 179. MPSPayment.jsp

### A.3.2 MPSPaymentSubmitted.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <!-- This file was generated by IBM WebSphere Studio 3.0.0 using
e:\WebStudio\BIN\GenerationStyleSheets\AppServerV3\JSP0.91\WebSphere\Pages.xsl -->

  <HEAD>
    <META HTTP-EQUIV="Content-Type" content="text/html; charset=ISO-8859-1">
    <META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
    <META http-equiv="Content-Style-Type" content="text/css">
    <TITLE> Welcome </TITLE>
    <LINK href="/FamilySample/theme/WSFamily.css" rel="stylesheet" type="text/css">
  </HEAD>
  <BODY>
    <H1 align="center"><BEAN name="payment_CMD" type="MPService.Payment_CMD" create="no"
introspect="no" scope="session"></BEAN>&nbsp; Mortgage Payment Service</H1>
    <CENTER>
      <TABLE border="1">
        <TBODY>
          <TR>
            <TD height="16"><B>Customer AccountId</B></TD>
            <TD><%= payment_CMD.getAccountId() %></TD>
          </TR>
          <TR>
            <TD><B>Payment Amount</B></TD>
            <TD><%= payment_CMD.getPaymentAmount() %></TD>
          </TR>
        </TBODY>
      </TABLE>
    </CENTER>
    <H2 align="center"><B><I>Payment Submitted</I></B></H2>
    <CENTER>
    </CENTER>
  </BODY>
</HTML>
```

Figure 180. MPSPaymentSubmitted.jsp

### A.3.3 MPSPaymentError.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <!-- This file was generated by IBM WebSphere Studio 3.0.0 using
e:\WebStudio\BIN\GenerationStyleSheets\AppServerV3\JSP0.91\WebSphere\Pages.xml -->

  <HEAD>
    <META HTTP-EQUIV="Content-Type" content="text/html; charset=ISO-8859-1">
    <META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
    <META http-equiv="Content-Style-Type" content="text/css">

    <LINK href="/FamilySample/theme/WSFamily.css" rel="stylesheet" type="text/css">

  </HEAD>
  <BODY>
    <P><BR>
    MPS Payment Error has occurred</P>
  </BODY>
</HTML>
```

Figure 181. MPSPaymentError.jsp

---

## A.4 Topology 6 implementation

### A.4.1 Commands.EJB.ShowAccounts\_CMD.java

```
package Commands.EJB;

import java.lang.*;
import java.util.*;
import Commands.*;
import com.ibm.ibmwebs.sample.*;

/**
 */
public class ShowAccounts_CMD extends Commands.ShowAccounts_CMD {
    protected BankTasksAccessBean btab = null;
    protected CustomerAccessBean cab = null;
    protected BankAccountAccessBean baab = null;
    protected BankAccountAccessBeanTable baabt = null;
    // Support for repeat
    protected int index = -1;
    public String getAccountId(int index) throws ArrayIndexOutOfBoundsException {
        String rtn = "Invalid index";
        setIndex(index);
        try {
            rtn = ((BankAccountKey)baab.__getKey()).accountId;
        }catch (java.lang.Exception e) {
        }
        return rtn;
    }
    public String getAccountType(int index) throws ArrayIndexOutOfBoundsException {
        String rtn = "Invalid index";
        setIndex(index);
        try {
            rtn = baab.getAccountType();
        }catch (java.lang.Exception e) {
        }
        return rtn;
    }
}
```

Figure 182. Commands.EJB.ShowAccounts\_CMD.java



```

public boolean getAll(int index) throws ArrayIndexOutOfBoundsException {
    //System.out.println(getClass().getName() + " - getTo( " + index + ")");
    setAllIndex(index);
    return false;
}

public String getBalance(int index) throws ArrayIndexOutOfBoundsException {
    String rtn = "Invalid index";
    setIndex(index);
    try {
        rtn = Double.toString(baab.getBalance());
    } catch (java.lang.Exception e) {
    }

    return rtn;
}

public boolean getFrom(int index) throws ArrayIndexOutOfBoundsException {
    //System.out.println(getClass().getName() + " - getFrom( " + index + ")");
    setIndex(index);
    return false;
}

public boolean getTo(int index) throws ArrayIndexOutOfBoundsException {
    //System.out.println(getClass().getName() + " - getTo( " + index + ")");
    setToIndex(index);
    return false;
}

public void perform() throws CustNotFoundException, java.lang.Exception {
    System.out.println("Commands.EJB.ShowAccounts_CMD perform");
    btab = new BankTasksAccessBean();

    if (nameServiceTypeName() != null)
        btab.setInit_NameServiceTypeName(nameServiceTypeName());
    if (nameServiceURLName() != null)
        btab.setInit_NameServiceURLName(nameServiceURLName());

    System.out.println("Commands.EJB.ShowAccounts_CMD lookupAccounts");
}

```

Figure 183. *Commands.EJB.ShowAccounts\_CMD.java - continued*

```

try {
    baabt = btab.lookupAccounts(customerId);
} catch (javax.ejb.FinderException fe) {
    throw new CustNotFoundException("id " + customerId + " does not exist in our
records...");
} finally {
    btab = null;
}
}
/* *
 *
 */
protected void setAllIndex(int index) throws ArrayIndexOutOfBoundsException {
    try {
        if (index < baabt.numberOfWorks()) {
            if (index >= 0 && this.index != index) {
                baab = (BankAccountAccessBean)baabt.getBankAccountAccessBean(index);
                this.index = index;
            }
        } else {
            throw new ArrayIndexOutOfBoundsException(index);
        }
    } catch (java.lang.Exception e) {
        throw new ArrayIndexOutOfBoundsException(index);
    }
}
/* *
 *
 */
protected void setIndex(int index) throws ArrayIndexOutOfBoundsException {
    try {

        if (index < baabt.numberOfWorks()) {
            if (index >= 0 && this.index != index) {
                baab = (BankAccountAccessBean)baabt.getBankAccountAccessBean(index);

                // added to accomodate for "MORTGAGE" accounts which shouldn't be in
                // the "from" list of Accounts to transfer money out of

```

Figure 184. *Commands.EJB.ShowAccounts\_CMD.java - continued*

```

        if (baab.getAccountType().trim().equalsIgnoreCase("MORTGAGE")) {
            setIndex(++index);
        }
        else
            this.index = index;
    }
} else {
    throw new ArrayIndexOutOfBoundsException(index);
}

} catch (java.lang.Exception e) {
    throw new ArrayIndexOutOfBoundsException(index);
}
}
/* *
 *
 */
protected void setToIndex(int index) throws ArrayIndexOutOfBoundsException {
    try {
        if (index < baabt.numberOfRows()) {
            if (index >= 0 && this.index != index) {
                baab = (BankAccountAccessBean)baabt.getBankAccountAccessBean(index);
                this.index = index;
            }
        }
        else {
            throw new ArrayIndexOutOfBoundsException(index);
        }
    } catch (java.lang.Exception e) {
        throw new ArrayIndexOutOfBoundsException(index);
    }
}
}

```

Figure 185. *Commands.EJB.ShowAccounts\_CMD.java - continued*

### A.4.2 Commands.ShowAccounts\_CMD.java

```
package Commands;
import java.lang.*;
import java.util.*;
/**
 */
public class ShowAccounts_CMD extends CMD
{
    // Input from the servlet getting the session
    protected String customerId = "not set";
    protected String[] balance = {"100","200","300"};
    protected String[] accountId = {"1","2","3"};
    protected String[] accountType= {"C","S","S"};
    public ShowAccounts_CMD() {
    }
    public String getAccountId(int index) throws ArrayIndexOutOfBoundsException {
        return accountId[index];
    }
    public String getAccountType(int index) throws ArrayIndexOutOfBoundsException {
        return accountType[index];
    }
    public boolean getAll(int index) throws ArrayIndexOutOfBoundsException {
        return false;
    }
    public String getBalance(int index) throws ArrayIndexOutOfBoundsException {
        return balance[index];
    }
    public boolean getFrom(int index) throws ArrayIndexOutOfBoundsException {
        return false;
    }
    public boolean getTo(int index) throws ArrayIndexOutOfBoundsException {
        return false;
    }
    public void perform() throws CustNotFoundException, java.lang.Exception {
        System.out.println("ShowAccounts_CMD: lookupAccounts: customerId: " +
        customerId);
    }
    public void setCustomerId(String value) throws java.lang.Exception {
        customerId = value;
    }
}
```

Figure 186. Commands.ShowAccounts\_CMD.java

### A.4.3 ShowAccountsResults.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <!-- This file was generated by IBM WebSphere Studio 3.0.0 using
e:\WebStudio\BIN\GenerationStyleSheets\AppServerV3\JSP0.91\WebSphere\Pages.xml -->
  <!-- The following lines prevent this page from being cached by proxy servers and
browsers. Remove them if you want to allow caching. -->

  <HEAD>
    <META HTTP-EQUIV="Content-Type" content="text/html; charset=ISO-8859-1">
    <META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
    <META http-equiv="Content-Style-Type" content="text/css">
    <LINK href="/FamilySample/theme/WSFamily.css" rel="stylesheet" type="text/css">
    <script language="JavaScript">
load=0;
window.onload=loadem;
hover=((navigator.appName=="Netscape")&&(parseInt(navigator.appVersion)>=3))||
((navigator.appName=="Microsoft Internet
Explorer")&&(parseInt(navigator.appVersion)>=4));

function preload(img){
  var a=new Image(); a.src=img; return a;
}
function mover(tag,row){
  if(load==0)return;
  document[tag+row].src=eval(tag+"_i.src");
}
function mout(tag,row){
  if(load==0)return;
  document[tag+row].src=eval(tag+"_o.src");
}
function loadem(){
  if(hover){
    history_i=preload('/WSFamily/history_i.gif');
    history_o=preload('/WSFamily/history_o.gif');
  }
  load=1;
}
function invoke(name) {
  document.forms[name].submit();
}
</script>
```

Figure 187. ShowAccountsResults.jsp

```

</HEAD>
<BODY>
  <BEAN name="showAccounts_CMD" type="Commands.ShowAccounts_CMD" create="no"
introspect="no" scope="request"></BEAN>
  <CENTER>
    <!--METADATA type="DynamicData" startspan
<TABLE border="1" width="600" dynamicelement
innerloopproperty="showAccounts_CMD.all()" innerloopdirection="vertical"
innerloopstartindex="1" innerloopendindex="1">
  <CAPTION><FONT size="+3"><B><FONT size="+2">Account
Balances</FONT></B></FONT></CAPTION>
  <TR>
<TH>Id</TH>
<TH>Type</TH>
<TH>Balance</TH>
<TH>History</TH>
</TR>
<TR>
  <TD align="center" bgcolor="#00cccc"><WSPX:PROPERTY
property="showAccounts_CMD.accountId()"></TD>
  <TD align="center" bgcolor="#00ffff"><WSPX:PROPERTY
property="showAccounts_CMD.accountType()"></TD>
  <TD align="right" bgcolor="#00cccc"><WSPX:PROPERTY
property="showAccounts_CMD.balance()"></TD>
  <TD class="TBL_EVEN" align="center" bgcolor="#00ffff"><FORM name="history<%= _i0
%>" method="POST" action="/FamilySample/servlet/WSFamily.ShowHistory" target="_self">
<A href="javascript:invoke('history<%= _i0
%>');"onmouseover="if (hover)mover('history',<%= _i0 %>)"
onmouseout="if (hover)mout('history',<%= _i0 %>)">
<IMG name="history<%= _i0 %>" src="/FamilySample/WSFamily/history_o.gif" width="50"
height="50" border="0"><INPUT type="hidden" dynamicelement name="accountId"
valueproperty="showAccounts_CMD.accountId()"></A></FORM></TD>
  </TR>
</TABLE>
--><%
try {
  boolean _p0 = showAccounts_CMD.getAll(0); // throws an exception if empty.
  java.lang.String _p0_0 = showAccounts_CMD.getAccountId(0);
  java.lang.String _p0_1 = showAccounts_CMD.getAccountType(0);
  java.lang.String _p0_2 = showAccounts_CMD.getBalance(0); %>

```

Figure 188. ShowAccountsResults.jsp

```

<TABLE border="1" width="600">
  <CAPTION><FONT size="+3"><B><FONT size="+2">Account
Balances</FONT></B></FONT></CAPTION>
  <TBODY>
    <TR>
      <TH>Id</TH>
      <TH>Type</TH>
      <TH>Balance</TH>
      <TH>History</TH>
    </TR><%
for (int _i0 = 0; ; ) { %>
    <TR>
      <TD align="center" bgcolor="#00cccc"><%= _p0_0 %></TD>
      <TD align="center" bgcolor="#00ffff"><%= _p0_1 %></TD>
      <TD align="right" bgcolor="#00cccc">$<%= _p0_2 %></TD>
      <TD align="center" bgcolor="#00ffff" class="TBL_EVEN">
        <FORM action="/FamilySample/servlet/WSFamily.ShowHistory" method="POST"
target="_self" name="history"<%= _i0 %>">
<A href="javascript:invoke('history<%= _i0 %>');"
onmouseout="if (hover)mout ('history',<%= _i0 %>)"
onmouseover="if (hover)mover ('history',<%= _i0 %>)">
<IMG border="0" height="50" name="history"<%= _i0 %>"
src="/FamilySample/WSFamily/history_o.gif" width="50"><INPUT name="accountId"
type="hidden" value="<%= _p0_0 %>"></A></FORM></TD>
      </TR><%
      _i0++;
      try {
        _p0 = showAccounts_CMD.getAll(_i0);
        _p0_0 = showAccounts_CMD.getAccountId(_i0);
        _p0_1 = showAccounts_CMD.getAccountType(_i0);
        _p0_2 = showAccounts_CMD.getBalance(_i0);
      }
      catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
        break;
      }
    } %>
  </TBODY>
</TABLE><%
}
catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
} %><!--METADATA type="DynamicData" ends-->
</CENTER></BODY> </HTML>

```

Figure 189. ShowAccountsResults.jsp

#### A.4.4 ShowTransferAccountsResults.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <!-- This file was generated by IBM WebSphere Studio 3.0.0 using
e:\WebStudio\BIN\GenerationStyleSheets\AppServerV3\JSP0.91\WebSphere\Pages.xml -->
  <!-- The following lines prevent this page from being cached by proxy servers and
browsers. Remove them if you want to allow caching. -->
  <%
response.setHeader("Pragma", "No-cache");
response.setHeader("Cache-Control", "no-cache");
response.setDateHeader("Expires", 0);
%>

  <HEAD>
    <META HTTP-EQUIV="Content-Type" content="text/html; charset=ISO-8859-1">
    <META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
    <META http-equiv="Content-Style-Type" content="text/css">
    <LINK href="/FamilySample/theme/WSFamily.css" rel="stylesheet" type="text/css">
    <SCRIPT language="JavaScript" src="/FamilySample/WSFamily/validation.js"></SCRIPT>
    <SCRIPT language="JavaScript">function control(myForm) {
      var from= "";
      var to = "";
      for (var i =0; i < myForm.elements.length; i++) {
        var element = myForm.elements[i];
        if ((element.name == "toAccount") &&
            (element.checked == true))
          to = element.value;
        else if ((element.name == "fromAccount") &&
            (element.checked == true))
          from = element.value;
      }//for

      if ((from=="") || (to == "")) {
        alert("Please select both a From and a To account....");
        return false;
      } else {
        if (from==to) {
          alert("Please select 2 different accounts...both are now " + from);
          return false;
        }
      }
    }
  </SCRIPT>
</HEAD>
  <BODY>
    <FORM name="myForm">
      <TABLE border="1">
        <TR>
          <TD>From Account</TD>
          <TD>To Account</TD>
        </TR>
        <TR>
          <TD>
            <INPUT type="text" value="<%=from%>"/>
          </TD>
          <TD>
            <INPUT type="text" value="<%=to%>"/>
          </TD>
        </TR>
        <TR>
          <TD colspan="2">
            <INPUT type="button" value="Transfer"/>
          </TD>
        </TR>
      </TABLE>
    </FORM>
  </BODY>
</HTML>
```

Figure 190. ShowTransferAccountsResults.jsp



```

    }//else
        return true;

    }
</SCRIPT>
<TITLE>Select From Account</TITLE>
</HEAD>
<BODY>
<FORM action="/FamilySample/servlet/WSFamily.TransferAccount"
onsubmit="this.amount.min=0;return (control(this) && verify(this));" method="POST"
target="_self">
<CENTER>
<BEAN name="showAccounts_CMD" type="Commands.ShowAccounts_CMD" create="no"
introspect="no" scope="request"></BEAN>
<!--METADATA type="DynamicData" startspan
<TABLE border="1" width="600" dynamicelement
innerloopproperty="showAccounts_CMD.from()" innerloopdirection="vertical"
innerloopstartindex="1" innerloopendindex="1">
    <CAPTION>Select From Account</CAPTION>
    <TR>
    <TH>From</TH>
    <TH>Account</TH>
    <TH>Type</TH>
    <TH>Balance</TH>
    </TR>
    <TR>
        <TD align="center" class="TBL_ODD" bgcolor="#00cccc">
            <INPUT type="radio" dynamicelement name="fromAccount"
valueproperty="showAccounts_CMD.accountId()"></TD>
        <TD class="TBL_EVEN" bgcolor="#00ffff"><WSPX:PROPERTY
property="showAccounts_CMD.accountId()">
        </TD>
        <TD class="TBL_ODD" bgcolor="#00cccc"><WSPX:PROPERTY
property="showAccounts_CMD.accountType()">
        </TD>
        <TD align="right" class="TBL_EVEN" bgcolor="#00ffff">${<WSPX:PROPERTY
property="showAccounts_CMD.balance()">
        </TD>
    </TR>

```

Figure 191. ShowTransferAccountsResults.jsp - continued

```

</TABLE>
--><%
try {
    boolean _p0 = showAccounts_CMD.getFrom(0); // throws an exception if empty.
    java.lang.String _p0_0 = showAccounts_CMD.getAccountId(0);
    java.lang.String _p0_1 = showAccounts_CMD.getAccountType(0);
    java.lang.String _p0_2 = showAccounts_CMD.getBalance(0); %>
    <TABLE border="1" width="600">
        <CAPTION>Select From Account</CAPTION>
        <TBODY>
            <TR>
                <TH>From</TH>
                <TH>Account</TH>
                <TH>Type</TH>
                <TH>Balance</TH>
            </TR><%
            for (int _i0 = 0; ; ) { %>
                <TR>
                    <TD align="center" bgcolor="#00cccc" class="TBL_ODD">

                        <INPUT name="fromAccount" type="radio" value="<%= _p0_0 %>"></TD>
                    <TD bgcolor="#00ffff" class="TBL_EVEN"><%= _p0_0 %>
                </TD>
                    <TD bgcolor="#00cccc" class="TBL_ODD"><%= _p0_1 %>
                </TD>
                    <TD align="right" bgcolor="#00ffff" class="TBL_EVEN">${<%= _p0_2 %>
                </TD>
            </TR><%
            _i0++;
            try {
                _p0 = showAccounts_CMD.getFrom(_i0);
                _p0_0 = showAccounts_CMD.getAccountId(_i0);
                _p0_1 = showAccounts_CMD.getAccountType(_i0);
                _p0_2 = showAccounts_CMD.getBalance(_i0);
            }
            catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
                break;
            }
        } %>

```

Figure 192. *ShowTransferAccountsResults.jsp - continued*

```

    </TBODY>    </TABLE><%
    }
    catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
    } %><!--METADATA type="DynamicData" ends-->
    <BR>
    <BR>
    <!--METADATA type="DynamicData" startspan
    <TABLE border="1" width="600" dynamicelement innerloopproperty="showAccounts_CMD.to()"
    innerloopdirection="vertical" innerloopstartindex="1" innerloopendindex="1">
        <CAPTION>Select To Account</CAPTION>
        <TR>
        <TH>To &nbsp; &nbsp; &nbsp; </TH>
        <TH>Account</TH>
        <TH>Type</TH>
        <TH>Balance</TH>
        </TR>
        <TR>
            <TD align="center" class="TBL_ODD" bgcolor="#00cccc">
                <INPUT type="radio" dynamicelement name="toAccount"
                valueproperty="showAccounts_CMD.accountId()" "></TD>
                <TD class="TBL_EVEN" bgcolor="#00ffff"><WSPX:PROPERTY
                property="showAccounts_CMD.accountId()" ">
                </TD>
                <TD class="TBL_ODD" bgcolor="#00cccc"><WSPX:PROPERTY
                property="showAccounts_CMD.accountType()" ">
                </TD>
                <TD align="right" class="TBL_EVEN" bgcolor="#00ffff"><WSPX:PROPERTY
                property="showAccounts_CMD.balance()" ">
                </TD>
            </TR>

        </TABLE>
    --><%
    try {
        boolean _p0 = showAccounts_CMD.getTo(0); // throws an exception if empty.
        java.lang.String _p0_0 = showAccounts_CMD.getAccountId(0);
        java.lang.String _p0_1 = showAccounts_CMD.getAccountType(0);
        java.lang.String _p0_2 = showAccounts_CMD.getBalance(0); %>

```

Figure 193. *ShowTransferAccountsResults.jsp* - continued



```

<TABLE border="1">
  <TBODY>
    <TR>
      <TD align="center"><INPUT type="submit" value="Transfer"></TD>
      <TD>Amount</TD>
      <TD><INPUT size="20" type="text" name="amount"></TD>
      <TD><INPUT type="reset" name="Reset"></TD>
    </TR>
  </TBODY>
</TABLE>
</CENTER>
<BR>
</FORM>

<BEAN type="Commands.ErrorMsg" name="errorMsg" introspect="no" create="no"
scope="session"></BEAN>

<P><% if (errorMsg.getError()) {
    out.println("<CENTER><P><B>" + errorMsg.getMessage() + "</B></P></CENTER>");
    errorMsg.setError(false);
}

%></P>

</BODY>
</HTML>

```

Figure 195. ShowTransferAccountsResults.jsp - continued

### A.4.5 com.ibm.ibmwebs.sample.BankTasksBean

```
package com.ibm.ibmwebs.sample;

import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
import javax.naming.*;
import mps.mq.*;

/**
 * This is a Session Bean Class
 */
public class BankTasksBean implements SessionBean {
    private javax.ejb.SessionContext mySessionCtx = null;
    final static long serialVersionUID = 3206093459760846163L;

    protected CustomerHome customerHome = null;
    protected BankAccountHome bankAccountHome = null;
    protected TranRecordHome tranRecordHome = null;
    /**
     * Insert the method's description here.
     * Creation date: (8/5/99 4:09:37 PM)
     * @param accountId java.lang.String
     * @param amount double
     * @exception javax.naming.NamingException The exception description.
     * @exception java.rmi.RemoteException The exception description.
     * @exception com.ibm.ibmwebs.sample.BankTransactionException The exception
    description.
     * @exception javax.ejb.FinderException The exception description.
     */
    public void deposit(String accountId, double amount) throws
    javax.naming.NamingException, java.rmi.RemoteException, FinderException,
    BankTransactionException {
        try {
            BankAccountAccessBean ba = lookupAccount(accountId);
            ba.deposit(amount);

            java.sql.Timestamp ts = new java.sql.Timestamp(System.currentTimeMillis());
            String id = ts.toString();
        }
    }
}
```

Figure 196. BankTasksBean.java

```

getTranRecordHome().create(id,amount,'C',((BankAccountKey)ba.getEJBRef().getPrimaryKey()).accountId, null);

    } catch (CreateException ce) {
        mySessionCtx.setRollbackOnly();
        throw new BankTransactionException(ce.getMessage());
    } catch (BankTransactionException bte) {
        throw new BankTransactionException(bte.getMessage());
    }
}
/**
 * ejbActivate method comment
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbActivate() throws java.rmi.RemoteException {}
/**
 * ejbCreate method comment
 * @exception javax.ejb.CreateException The exception description.
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbCreate() throws javax.ejb.CreateException, java.rmi.RemoteException {}
/**
 * ejbPassivate method comment
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbPassivate() throws java.rmi.RemoteException {}
/**
 * ejbRemove method comment
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbRemove() throws java.rmi.RemoteException {}
/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:07:50 PM)
 * @return com.ibm.ibmwebs.sample.BankAccountHome
 */

```

Figure 197. BankTasksBean.java

```

public BankAccountHome getBankAccountHome() {
    if (bankAccountHome == null) {
        // Get the initial context
        try {
            Context ctx = new InitialContext();
            Object homeObject = ctx.lookup("BankAccount");
            bankAccountHome =
                (BankAccountHome) javax.rmi.PortableRemoteObject.narrow(homeObject,
                BankAccountHome.class);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    return bankAccountHome;
}
/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:07:28 PM)
 * @return com.ibm.ibmwebs.sample.CustomerHome
 */
public CustomerHome getCustomerHome() {
    if (customerHome == null) {
        // Get the initial context
        try {
            Context ctx = new InitialContext();
            Object homeObject = ctx.lookup("Customer");
            customerHome = (CustomerHome)
                javax.rmi.PortableRemoteObject.narrow(homeObject, CustomerHome.class);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    return customerHome;
}

```

Figure 198. *BankTasksBean.java - continued*



```

/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:08:11 PM)
 * @return com.ibm.ibmwebs.sample.TranRecordHome
 */
public TranRecordHome getTranRecordHome() {
    if (tranRecordHome == null) {
        // Get the initial context
        try {
            Context ctx = new InitialContext();
            Object homeObject = ctx.lookup("TranRecord");
            tranRecordHome = (TranRecordHome)
javax.rmi.PortableRemoteObject.narrow(homeObject, TranRecordHome.class);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    return tranRecordHome;
}
/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:14:24 PM)
 * @return com.ibm.ibmwebs.sample.BankAccountAccessBean
 * @param accountId java.lang.String
 * @exception javax.naming.NamingException The exception description.
 * @exception java.rmi.RemoteException The exception description.
 * @exception javax.ejb.FinderException The exception description.
 */
public BankAccountAccessBean lookupAccount(String accountId) throws
javax.naming.NamingException, java.rmi.RemoteException, FinderException {
    try {
        BankAccountAccessBean baab = new BankAccountAccessBean();
        baab.setInitKey_accountId(accountId);
        baab.refreshCopyHelper();
        return baab;
    } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
    }
}

```

Figure 199. BankTasksBean.java - continued

```

/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:15:40 PM)
 * @return com.ibm.ibmwebs.sample.BankAccountAccessBeanTable
 * @param customerId java.lang.String
 * @exception javax.naming.NamingException The exception description.
 * @exception java.rmi.RemoteException The exception description.
 * @exception javax.ejb.FinderException The exception description.
 */
public BankAccountAccessBeanTable lookupAccounts(String customerId) throws
javax.naming.NamingException, java.rmi.RemoteException, FinderException {
    Customer cust = null;
    java.util.Enumeration accts = null;
    System.out.println("##### lookupAccounts with customerId-" + customerId + "
Begin#####");

    try {
        BankAccountAccessBeanTable baabt = new BankAccountAccessBeanTable();
        CustomerKey custKey = new CustomerKey(customerId);
        cust = getCustomerHome().findByPrimaryKey(custKey);
        accts = cust.getBankAccounts();
        System.out.println("#####");
        System.out.println("Accounts for customer " + customerId);
        while (accts.hasMoreElements()) {
            BankAccount ba = null;
            ba = (BankAccount) javax.rmi.PortableRemoteObject.narrow(accts.nextElement(),
BankAccount.class);
            BankAccountAccessBean baab = new BankAccountAccessBean(ba);
            // baab.setInitKey_accountId(((BankAccountKey)
ba.getPrimaryKey()).accountId);
            baab.refreshCopyHelper();
            System.out.println("Account for Customer");
            System.out.println("Account for customer " + ((BankAccountKey)
baab.__getKey()).accountId);
            baabt.addRow(baab);
        }

        return baabt;
    } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
    }
}

```

Figure 200. BankTasksBean.java - continued

```

/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:14:56 PM)
 * @return com.ibm.ibmwebs.sample.CustomerAccessBean
 * @param customerId java.lang.String
 * @exception javax.naming.NamingException The exception description.
 * @exception java.rmi.RemoteException The exception description.
 * @exception javax.ejb.FinderException The exception description.
 */
public CustomerAccessBean lookupCustomer(String customerId) throws
javax.naming.NamingException, java.rmi.RemoteException, FinderException {
    System.out.println("##### lookupCustomer Begin#####");
    try {
        CustomerAccessBean cab = new CustomerAccessBean();
        cab.setInitKey_customerId(customerId);
        cab.refreshCopyHelper();

        System.out.println("##### lookupCustomer End#####");
        return cab;
    } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
    }
}

/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:16:18 PM)
 * @return com.ibm.ibmwebs.sample.TranRecordAccessBeanTable
 * @param accountId java.lang.String
 * @exception javax.naming.NamingException The exception description.
 * @exception java.rmi.RemoteException The exception description.
 * @exception javax.ejb.FinderException The exception description.
 */
public TranRecordAccessBeanTable lookupHistory(String accountId) throws
javax.naming.NamingException, java.rmi.RemoteException, FinderException {
    BankAccount acct = null;

    try {
        BankAccountKey acctKey = new BankAccountKey(accountId);
        acct = getBankAccountHome().findByPrimaryKey(acctKey);
        java.util.Enumeration tre = acct.getTranRecords();

```

Figure 201. BankTasksBean.java - continued

```

        TranRecordAccessBeanTable trabt = new TranRecordAccessBeanTable();
        System.out.println("*****");
        System.out.println("Starting counting of tran records");
        int counter = 0;
        while (tre.hasMoreElements()) {
            System.out.println("Counter of tran records is " + ++counter );
            TranRecord tr = null;
            tr = (TranRecord) javax.rmi.PortableRemoteObject.narrow(tre.nextElement(),
TranRecord.class);
            TranRecordAccessBean trab = new TranRecordAccessBean(tr);
            // ADDED TEMPORARILY BY GLT
            //trab.setInitKey_transId(((TranRecordKey)tr.getPrimaryKey()).transId);
            trab.refreshCopyHelper();
            trabt.addRow(trab);
        }

        return trabt;

    } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
    }
}
/**
 * setSessionContext method comment
 * @param ctx javax.ejb.SessionContext
 * @exception java.rmi.RemoteException The exception description.
 */
public void setSessionContext(javax.ejb.SessionContext ctx) throws
java.rmi.RemoteException {
    mySessionCtx = ctx;
}
/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:12:30 PM)
 * @param fromAcctId java.lang.String
 * @param toAcctId java.lang.String
 * @param amount double
 * @exception javax.naming.NamingException The exception description.
 * @exception java.rmi.RemoteException The exception description.
 * @exception com.ibm.ibmwebs.sample.BankTransactionException The exception
description.
 * @exception javax.ejb.FinderException The exception description.
 */

```

Figure 202. BankTasksBean.java - continued

```

public void transfer(String fromAcctId, String toAcctId, double amount) throws
javax.naming.NamingException, java.rmi.RemoteException, BankTransactionException,
FinderException {

    System.out.println(getClass().getName() + " transferring $" + amount + " from
account: " + fromAcctId + " to account: " + toAcctId);
    try {
        BankAccountAccessBean fromba = lookupAccount(fromAcctId);
        BankAccountAccessBean toba = lookupAccount(toAcctId);

        fromba.withdraw(amount);

        java.sql.Timestamp ts = new java.sql.Timestamp(System.currentTimeMillis());
        String id = ts.toString();

        getTranRecordHome().create(id, amount, 'D', ((BankAccountKey) fromba.getEJBRef().getPrimary
Key()).accountId, ((BankAccountKey) toba.getEJBRef().getPrimaryKey()).accountId);

        // handle transfers to Mortgage accounts by sending a message to the MQ-backed
MPSOutbound EJB
        if (toba.getAccountType().trim().equalsIgnoreCase("MORTGAGE")) {

            String homeName = "mps/mq/MPSOutboundEJBHome";

            mps.mq.MPSOutboundAccessBean mpsOutbound = new
mps.mq.MPSOutboundAccessBean();
            mpsOutbound.setInit_JNDIName(homeName);

            // create mpsOutbound message template
            String queueName = "MPS.QUEUE";
            String paymentAmount = String.valueOf(amount);

            mps.mq.MPSMsgTemplate template = new mps.mq.MPSMsgTemplate();
            template.setQueueName(queueName);
            template.setAccountId( toAcctId );
            template.setPaymentAmount(paymentAmount );

```

Figure 203. BankTasksBean.java - continued

```

        // put the message
        System.out.println(getClass().getName() + " sending MPS payment - accountId:
" + toAcctId + " paymentAmount: " + paymentAmount);
        String correlatorId = mpsOutbound.put(template);
        System.out.println(getClass().getName() + "message put successfully");
    }
    toba.deposit(amount);

    ts = new java.sql.Timestamp(System.currentTimeMillis());
    id = ts.toString();

    getTranRecordHome().create(id, amount, 'C', ((BankAccountKey) toba.getEJBRef().getPrimary
Key()).accountId, ((BankAccountKey) fromba.getEJBRef().getPrimaryKey()).accountId);

    } catch (BankTransactionException bte) {
        throw new BankTransactionException(bte.getMessage());
    } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
    }
    // handle CB-related exceptions from the MPSOutbound component
    catch (com.ibm.ILogonHome.ILogonPutFailed e) {
        throw new BankTransactionException(e.getMessage());
    } catch (com.ibm.ILogonClient.IInvalidKey e) {
        throw new BankTransactionException(e.getMessage());
    } catch (com.ibm.ILogonClient.IInvalidCopy e) {
        throw new BankTransactionException(e.getMessage());
    }
}
/**
 * Insert the method's description here.
 * Creation date: (8/5/99 4:10:31 PM)
 * @param accountId java.lang.String
 * @param amount double
 * @exception javax.naming.NamingException The exception description.
 * @exception java.rmi.RemoteException The exception description.
 * @exception com.ibm.ibmwebs.sample.BankTransactionException The exception
description.
 * @exception javax.ejb.FinderException The exception description.
 */

```

Figure 204. BankTasksBean.java - continued

```

public void withdraw(String accountId, double amount) throws
javax.naming.NamingException, java.rmi.RemoteException, BankTransactionException,
FinderException {
    try {
        BankAccountAccessBean ba = lookupAccount(accountId);
        ba.withdraw(amount);

        java.sql.Timestamp ts = new java.sql.Timestamp(System.currentTimeMillis());
        String id = ts.toString();

        getTranRecordHome().create(id, amount, 'D', ((BankAccountKey)ba.getEJBRef().getPrimaryKe
y()).accountId, (String) null);

    } catch (BankTransactionException bte) {
        throw new BankTransactionException(bte.getMessage());
    } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
    }
}
}

```

Figure 205. BankTasksBean.java - continued

## A.5 MPSOutboundTest.java

```
package mps.mq;

public class MPSOutboundTest {
    private transient javax.transaction.UserTransaction currentTransaction;
    private javax.naming.InitialContext jndiContext;

    private static final String PROVIDER_URL_PROP =
"javax.naming.Context.PROVIDER_URL";
    private static final String INITIAL_NAMING_FACTORY_PROP =
"java.naming.factory.initial";
    private static String providerURL = "iiop://was1.sl.dfw.ibm.com:910";
    private static final String WAS_NAMING_FACTORY =
"com.ibm.ejs.ns.jndi.CNInitialContextFactory"; // in WebSphere AE
    private static final String CB_NAMING_FACTORY =
"com.ibm.ejb.cb.runtime.CBCtxFactory"; // in WebSphere EE (CB)
    private static String namingFactory = CB_NAMING_FACTORY;

    public MPSOutboundTest() {
        super();
    }
    /*
    * begin transaction
    */
    private boolean beginTransaction() {

        if (currentTransaction == null) {
            try {
                currentTransaction = (javax.transaction.UserTransaction)
jndiContext.lookup("jta/usertransaction");

                if (currentTransaction == null) throw new
NullPointerException("currentTransaction");
                System.out.println(">> transaction found");
            }
            catch (javax.naming.NamingException e) {
                ejb.Util.printStackTrace("ffic.ejb.printer.PrinterBean", "beginTransaction", e);
                return false;
            }
        }
    }
}
```

Figure 206. MPSOutboundTest.java



```

catch (NullPointerException e) {
    ejb.Util.printException("ffic.ejb.printer.PrinterBean", "beginTransaction",
e);
    return false;
}

try {
    currentTransaction.begin();
    System.out.println(">> transaction started..");
    return true;
}
catch (java.lang.Throwable e) {
    ejb.Util.printException("mps.mq.MPSOutboundTest", "beginTransaction", e);
    return false;
}
}
/*
 * commit transaction
 */
private boolean commitTransaction() {

    if (currentTransaction == null) {
        System.out.println(">> no trasaction exists to commit");
        return false;
    }

    try {
        currentTransaction.commit();
        System.out.println(">> transaction committed..");
        return true;
    }
    catch (java.lang.Throwable e) {
        ejb.Util.printException("mps.mq.MPSOutboundTest", "commitTransaction", e);
        return false;
    }
}
}

```

Figure 207. *MPSOutboundTest.java - continued*

```

/**
 * main
 */
public static void main(java.lang.String[] args) {

    MPSOutboundTest test = new MPSOutboundTest();

    if (args.length < 2) {
        System.out.println("Usage: java mps.mq.MPSOutboundTest <hostname> <port>
[\"ACCESSBEAN\"]");
        System.exit(1);
    }

    String hostName = args[0];
    String port = args[1];

    test.providerURL = "iiop://" + hostName + ":" + port;

    if ( (args.length > 2) && (args[2].equalsIgnoreCase("ACCESSBEAN")) )
        test.runWithAccessBean();
    else
        test.run();
}
/*
 * rollback transaction
 */
private boolean rollbackTransaction() {

    if (currentTransaction == null) {
        System.out.println(">> no transaction exists to rollback");
        return false;
    }

    try {
        currentTransaction.rollback();
        System.out.println(">> transaction rolled back..");
        return true;
    }
}

```

Figure 208. *MPSOutboundTest.java - continued*

```

        catch (java.lang.Throwable e) {
            ejb.Util.printStackTrace("mps.mq.MPSOutboundTest", "rollbackTransaction", e);
            return false;
        }
    }
    /**
     * run
     */
    public void run() {

        try {
            System.out.println("MPSOutboundTest using normal EJB client programming");

            String homeName = "mps/mq/MPSOutboundEJBHome";
            mps.mq.MPSOutboundEJBHome home = null;
            mps.mq.MPSOutboundEJBObject mpsOutbound = null;

            // initialize JNDI Context
            System.out.println("Initializing JNDI Context for providerURL: " +
providerURL);
            java.util.Properties env = new java.util.Properties();
            env.put(javax.naming.Context.PROVIDER_URL, providerURL);
            env.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, namingFactory);

            jndiContext = new javax.naming.InitialContext( env );

            if (jndiContext == null) throw new NullPointerException("jndiContext");
            else System.out.println(">> JNDI context initialized");

            // find home
            java.lang.Object o = jndiContext.lookup (homeName);
            if (o instanceof org.omg.CORBA.Object) {
                home = (mps.mq.MPSOutboundEJBHome) javax.rmi.PortableRemoteObject.narrow(
(org.omg.CORBA.Object) o,
                mps.mq.MPSOutboundEJBHome.class);
            }
        }
    }

```

Figure 209. *MPSOutboundTest.java* - continued

```

        if (home == null) throw new NullPointerException("home");
        else System.out.println(">> home found");

        // begin transaction
        beginTransaction();

        // create mpsOutbound
        mpsOutbound = home.create();
        if (mpsOutbound == null) throw new NullPointerException("mpsOutbound");
        else System.out.println(">> mpsOutbound created");

        // create mpsOutbound message template
        String queueName = "MPS.QUEUE";
        String accountId = "11114000";
        String paymentAmount = "1000.00";

        mps.mq.MPSMsgTemplate template = new mps.mq.MPSMsgTemplate();
        template.setQueueName(queueName);
        template.setAccountId(accountId);
        template.setPaymentAmount(paymentAmount);

        // put the message
        String correlatorId = mpsOutbound.put(template);
        System.out.println("message put successfully");

        // commit transaction
        commitTransaction();
    }
    catch (Throwable e) {
        rollbackTransaction();
        ejb.Util.printStackTrace("mps.mq.MPSOutboundTest", "run", e);
    } }
/**
 * runWithAccessBean
 */
public void runWithAccessBean() {
    try {
        System.out.println("MPSOutboundTest using associated AccessBean");
        String homeName = "mps/mq/MPSOutboundEJBHome";
        mps.mq.MPSOutboundAccessBean mpsOutbound = new mps.mq.MPSOutboundAccessBean();
    }
}

```

Figure 210. MPSOutboundTest.java - continued

```

mpsOutbound.setInit_JNDIName(homeName);
mpsOutbound.setInit_NameServiceTypeName(namingFactory);
mpsOutbound.setInit_NameServiceURLName(providerURL);

// the only reason we call getInitialContext() directly is because we need
// the jndiContext to find and start a UserTransaction,
// otherwise, the InitialContext would be initialized when we
// make the first call on the AccessBean - mpsOutbound.put()
this.jndiContext = mpsOutbound.getInitContext(providerURL, namingFactory);

// begin transaction
if (!beginTransaction()) System.exit(0);

// create mpsOutbound message template
String queueName = "MPS.QUEUE";
String accountId = "11114000";
String paymentAmount = "1000.00";

mps.mq.MPSMsgTemplate template = new mps.mq.MPSMsgTemplate();
template.setQueueName(queueName);
template.setAccountId(accountId);
template.setPaymentAmount(paymentAmount);

// put the message
String correlatorId = mpsOutbound.put(template);
System.out.println("message put successfully");
// commit transaction
commitTransaction();
}
catch (Throwable e) {
    rollbackTransaction();
    ejb.Util.printException("mps.mq.MPSOutboundTest", "runWithAccessBean", e);
}
}
}

```

Figure 211. *MPSOutboundTest.java - continued*

---

## A.6 Loading the MQ application in CB

The following file was used to define the new MQ application to Component Broker.

```
@echo OFF
rem This file loads applications in the MPSejbFamily.
rem The variable RET is used to capture return codes and help control
rem program flow.
rem If the program catches an error, check the command window where you
rem are running
rem it for additional information regarding the error.

set DEBUG=0
@if not "%ECHO%"==" " echo %ECHO%
@if not "%OS%"=="Windows_NT" goto DOSEXIT

rem Set local scope and call MAIN procedure
setlocal & pushd & set RET=
if {%1}=={} (call :Help %2) & (goto :HELPEXIT)
set HOST="%1"
if {%2}=={} (call :Help %2) & (goto :HELPEXIT)
set MPS_DIR="%2"
set MZONE=Sample Application Zone
set CONFIG=Sample Configuration
set SRVGRP=CBEJBSG
set SERVER=CBEJBServer
set APPATH=%MPS_DIR%\Working\NT\PRODUCTION\MPSejbFamily\
if "%DEBUG%"=="1" (set TRACE=echo) else (set TRACE=rem)
if /i {%1}=={/help} (call :HELP %1) & (goto :HELPEXIT)
if /i {%1}=={/?} (call :HELP %1) & (goto :HELPEXIT)
call :MAIN %*
if %RET% NEQ 0 (call :ERRORMSG %RET%)

:HELPEXIT
popd & endlocal & set RET=%RET%
goto :EOF

rem
```

Figure 212. loadApplication.bat

```

/////////////////////////////////////////////////////////////////
rem HELP procedure
rem Display brief on-line help message
rem
:HELP
if defined TRACE %TRACE% [proc %0 %*]
    echo LoadApplication will load and activate the MPSejbFamily.
    echo Usage: loadApplication Host MPS_DIR
    echo   Host      - Fully-qualified host name to configure the server
on.
    echo   MPS_DIR   - Full path name of the directory where the
family MPS samples reside.
    echo Example, loadApplication myhost.austin.ibm.com
d:\family\mps

goto :EOF

rem
/////////////////////////////////////////////////////////////////
//
rem MAIN procedure
rem
:MAIN

call wscmd "Host image.%HOST%" blockdo loadApplication
"%APPATH%MPSejbFamily.ddl"
set RET=%ERRORLEVEL%
if %RET% NEQ 0 goto :EOF

call wscmd "Management zone.%MZONE%/Configuration.%CONFIG%" do "Add
application" "HostI.%HOST%/Application family
install.MPSejbFamily/Application install.MPSOutboundApp"
set RET=%ERRORLEVEL%
if %RET% NEQ 0 goto :EOF

call wscmd "Management zone.%MZONE%/Configuration.%CONFIG%" do "Add
application" "HostI.%HOST%/Application family
install.MPSejbFamily/Application install.MPSOutboundEJBObjectApp"
set RET=%ERRORLEVEL%
if %RET% NEQ 0 goto :EOF

```

Figure 213. loadApplication.bat - continued

```

call wscmd "Management
zone.%MZONE%/Configuration.%CONFIG%/Application.MPSOutboundApp" relate
"Management zone.%MZONE%/Configuration.%CONFIG%/Server group.%SRVGRP%"
by ApplicationConfiguredOnModel
set RET=%ERRORLEVEL%
if %RET% NEQ 0 goto :EOF

call wscmd "Management
zone.%MZONE%/Configuration.%CONFIG%/Application.MPSOutboundEJObjectApp
" relate "Management zone.%MZONE%/Configuration.%CONFIG%/Server
group.%SRVGRP%" by ApplicationConfiguredOnModel
set RET=%ERRORLEVEL%
if %RET% NEQ 0 goto :EOF

call wscmd "Management
zone.%MZONE%/Configuration.%CONFIG%/Application.iMQAAServices" relate
"Management zone.%MZONE%/Configuration.%CONFIG%/Server group.%SRVGRP%"
by ApplicationConfiguredOnModel
set RET=%ERRORLEVEL%
if %RET% NEQ 0 goto :EOF

call wscmd "Management
zone.%MZONE%/Configuration.%CONFIG%/Application.iDXAAAServices" relate
"Management zone.%MZONE%/Configuration.%CONFIG%/Server group.%SRVGRP%"
by ApplicationConfiguredOnModel
set RET=%ERRORLEVEL%
if %RET% NEQ 0 goto :EOF

echo Activate the configuration
call wscmd "Management zone.%MZONE%/Configuration.%CONFIG%" blockdo
activate
set RET=%ERRORLEVEL%
if %RET% NEQ 0 goto :EOF

goto :EOF

```

Figure 214. *loadApplication.bat - continued*



```

rem
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
rem Additional procedures go here...
rem ERRORMSG procedure
rem
:ERRORMSG
if defined TRACE %TRACE% [proc %0 %*]

if %RET% EQU 1 echo Return Code 1 - An incorrect action was specified
if %RET% EQU 2 echo Return Code 2 - Not enough parameters were specified
for the action
if %RET% EQU 3 echo Return Code 3 - Named object not located
if %RET% EQU 4 echo Return Code 4 - The server returned a non-zero
return code
if %RET% EQU 8 echo Return Code 8 - An internal error occurred
if %RET% EQU 16 echo Return Code 16 - An asynchronous command invoked by
a blockdo failed

goto :EOF
rem These must be the FINAL LINES in the script...
:DOSEXIT
echo This script requires Windows NT

rem
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//

```

Figure 215. *loadApplication.bat* - continued

To run the bat file, enter the following:

```
loadApplication was1.sl.dfw.ibm.com c:\family\mps
```

---

## A.7 MPSPostingService application

The application, MPSPostingService, is written in C++. The source is shown in Figure 216.

```

extern "C" {
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
}

#include <imqi.hpp> // MQSeries MQI

int main ( int argc, char * * argv ) {

    ImqQueueManager mgr;           /* Queue manager          */
    ImqQueue queue;               /* Queue                  */
    ImqMessage msg;               /* Data message           */
    ImqGetMessageOptions gmo;      /* Get message options    */
    char buffer[ 10001 ];         /* Message buffer         */

    /*****
    /*
    /*  Create object descriptor for subject queue
    /*
    /*****
queue.setName( "MPS.QUEUE" );
mgr.setName( "MPS.QUEUE.MANAGER" );

    /*****
    /*
    /*  Connect to queue manager
    /*
    /*****
if ( ! mgr.connect( ) ) {

    /* stop if it failed */
    printf( "ImqQueueManager::connect failed with reason code %ld\n",
            (long)mgr.reasonCode( ) );
    exit( (int)mgr.reasonCode( ) );
}

```

Figure 216. MPSPostingService.cpp

```

/*****
/*
/*   Associate queue with queue manager.
/*
/*
/*****
queue.setConnectionReference( mgr );

/*****
/*
/*   Open the named message queue for input; exclusive or shared
/*   use of the queue is controlled by the queue definition here
/*
/*
/*****
queue.setOpenOptions(
    MQOO_INPUT_AS_Q_DEF /* open queue for input
    + MQOO_FAIL_IF_QUIESCING
    ); /* but not if MQM stopping
queue.open( );

/* report reason, if any; stop if failed
if ( queue.reasonCode( ) ) {
    printf( "ImqQueue::open ended with reason code %ld\n",
        (long)queue.reasonCode( ) );
}

if ( queue.completionCode( ) == MQCC_FAILED ) {
    printf( "unable to open queue for input\n" );
}

```

Figure 217. MPSPostingService.cpp - continued

```

/*****
/*
/*  Get messages from the message queue
/*  Loop until there is a failure
/*
/*****
msg.useEmptyBuffer( buffer, sizeof( buffer ) - 1 );
/* buffer size available for GET
gmo.setOptions( MQGMO_WAIT | /* wait for new messages
MQGMO_FAIL_IF_QUIESCING );
gmo.setWaitInterval( 3000 ); /* wait three seconds */

// while ( queue.completionCode( ) != MQCC_FAILED ) {
while ( 1 ) {

/*****
/*
/*  In order to read the messages in sequence, MsgId and
/*  CorrelID must have the default value. MQGET sets them
/*  to the values in for message it returns, so re-initialise
/*  them before every call
/*
/*****
msg.setMessageId( );
msg.setCorrelationId( );
if ( queue.get( msg, gmo ) ) {

/*****
/*  Display each message received
/*****
if ( msg.formatIs( MQFMT_STRING ) ) {

    buffer[ msg.dataLength( ) ] = 0 ; /* add terminator
    char *acctId = &(buffer[0]);
    char *payment = &(buffer[9]);

```

Figure 218. MPSPostingService.cpp - continued

```

buffer[8] = '\\0';

    char report[200] = "";
    sprintf (report, "%s's payment is %s\\n", acctId, payment);

    FILE * fp = fopen("MPSReport.txt", "w");
    fprintf( fp, report);
    fclose( fp );

    system ("start /B notepad.exe MPSReport.txt");
}
}

/* Code we never expect to reach... */

/*****
/*
/*   Close the source queue (if it was opened)
/*
/*
/*****
if ( ! queue.close( ) ) {

    /* report reason, if any */
    printf( "ImqQueue::close failed with reason code %ld\\n",
        (long)queue.reasonCode( ) );
}

/*****
/*
/*   Disconnect from MQM if not already connected (the
/*   ImqQueueManager object handles this situation automatically)
/*
/*
/*****
if ( ! mgr.disconnect( ) ) {

    /* report reason, if any */
    printf( "ImqQueueManager::disconnect failed with reason code %ld\\n",
        (long)mgr.reasonCode( ) );
}
return( 0 );
}

```

Figure 219. MPSPostingService.cpp - continued



---

## Appendix B. Special notices

This publication is intended to help IT architects and IT specialists in the design and deployment of e-business applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM WebSphere Application Server 3.0, Enterprise Edition. See the PUBLICATIONS section of the IBM Programming Announcement for IBM WebSphere Application Server 3.0, Enterprise Edition for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee

that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

|            |               |
|------------|---------------|
| AIX        | AS/400        |
| CICS       | CICS/ESA      |
| DB2        | DB2 Connect   |
| Domino     | IBM           |
| Lotus      | MQSeries      |
| Netfinity  | Notes         |
| OS/390     | RS/6000       |
| S/390      | San Francisco |
| SecureWay  | SP            |
| System/390 | Tivoli        |
| TME        | TXSeries      |
| VisualAge  | WebSphere     |
| Wizard     |               |

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.



SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.



---

## Appendix C. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

---

### C.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 337.

- *IBM Component Broker Connector Overview*, SG24-2022
- *WebSphere Application Server Enterprise Edition Component Broker 3.0: First Steps*, SG24-2033
- *CCF Connectors and Database Connections using WebSphere Advanced Edition Connecting Enterprise Information Systems to the Web*, SG24-5514
- *Application Server Solution Guide Enterprise Edition: Getting Started*, SG24-5320
- *Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server*, SG24-5754
- *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864
- *WebSphere Scalability - WLM and Clustering*, SG24-6153
- *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755
- *A Secure Way to Protect Your Network: IBM SecureWay Firewall for AIX V4.1*, SG24-5855
- *Guarding the Gates Using the IBM eNetwork Firewall V3.3 for Windows NT*, SG24-5209

---

### C.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at [ibm.com/redbooks](http://ibm.com/redbooks) for information about all the CD-ROMs offered, updates and formats.

| CD-ROM Title                       | Collection Kit Number |
|------------------------------------|-----------------------|
| IBM System/390 Redbooks Collection | SK2T-2177             |
| IBM Networking Redbooks Collection | SK2T-6022             |

| CD-ROM Title   | Collection Kit Number |
|--|-----------------------|
| IBM Transaction Processing and Data Management Redbooks Collection | SK2T-8038             |
| IBM Lotus Redbooks Collection                                      | SK2T-8039             |
| Tivoli Redbooks Collection   | SK2T-8044             |
| IBM AS/400 Redbooks Collection                                     | SK2T-2849             |
| IBM Netfinity Hardware and Software Redbooks Collection            | SK2T-8046             |
| IBM RS/6000 Redbooks Collection                                    | SK2T-8043             |
| IBM Application Development Redbooks Collection                    | SK2T-8037             |
| IBM Enterprise Storage and Systems Management Solutions            | SK3T-3694             |

---

### C.3 Other resources

These publications are also relevant as further information sources:

#### **Product pubs**

- *WebSphere Application Server Enterprise Edition Component Broker for Windows NT: Planning, Performance, and Installation Guide*, SC09-4436
- *Component Broker Advanced Programming Guide*, SC09-4443
- *WebSphere Application Server Enterprise Edition Component Broker System Administration Guide*, SC09-4445
- *WebSphere Application Server Enterprise Edition Component Broker for Windows NT: Getting Started with Component Broker*, SC09-4433
- WebSphere product documentation found at  
<http://www.ibm.com/software/webservers/appserv/library.html>

#### **External books**

- E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995, ISBN 0201633612
- F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stahl, *Pattern-Oriented Software Architecture - A System of Patterns*, Wiley, 1996, ISBN 0471958697
- J. Vlissides, *Pattern Hatching - Design Patterns Applied*, Addison Wesley, 1998, ISBN 0201432935
- C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel, *A Pattern Language*. Oxford University Press, 1977, ISBN 0195019199
- Flanagan, David, *JavaScript: The Definitive Guide*, Third Edition, O'Reilly & Associates, Inc., 1998, ISBN 1565923928

- Flanagan, David, Jim Farley, William Crawford and Kris Magnusson, *Java Enterprise in a Nutshell*, O'Reilly & Associates, Inc. 1999, ISBN 1565924835
- Maruyama, Hiroshi, Kent Tamura and Naohiko Uramoto, *XML and Java: Developing Web Applications*, Addison-Wesley, 1999, ISBN 0201485435

#### **White papers**

- Nagaratnam, Nataraj et al. 2000. *Security Overview of IBM WebSphere Standard/Advance 3.02*, IBM white paper, available at:  
<http://www.ibm.com/software/webservers/appserv/whitepapers.html>
- *IBM Application Framework for e-business*: white papers available at:  
<http://www.ibm.com/software/ebusiness/>

---

## **C.4 Referenced Web sites**

These Web sites are also relevant as further information sources:

- <http://www.redbooks.ibm.com>
- <http://www.research.ibm.com/journal/sj38-1.html>
- <http://www.ibm.com/software/developer/web/patterns>
- <http://www.ibm.com/software/webservers/samples>
- [http://www.ibm.com/software/webservers/appserv/dmz\\_v3.html](http://www.ibm.com/software/webservers/appserv/dmz_v3.html)
- <http://www.ibm.com/software/webservers/appserv/whitepapers.html>
- <http://www.ecma.ch/stand/ECMA-262.htm>
- <http://www.javasoft.com/products/servlet>
- <http://www.javasoft.com/products/jsp>
- <http://www.javasoft.com/products/ejb/index.html>
- <http://www.ibm.com/software/ebusiness>
- [http://www.ibm.com/software/ebusiness/arch\\_overview.html](http://www.ibm.com/software/ebusiness/arch_overview.html)
- <http://www.ibm.com/software/ebusiness/buildapps/understand.html>
- <http://www.java.sun.com/products/jsp>
- <http://www.ibm.com/developer/features/framework/framework.html>
- <http://www.w3.org/MarkUp>
- <http://java.sun.com/products/servlet/2.1>
- <http://www.ibm.com/software/webservers/appserv/library.html>
- <http://java.sun.com/beans/index.htm>
- <http://java.sun.com/products/ejb/index.html>
- <http://as400.rochester.ibm.com/products/websphere/docs/doc/apidocs/com.ibm>

- <http://www.ibm.com/software/webservers/appserv/doc/v30ee/cbpdf>
- <http://www.ibm.com/software/webservers/appserv/doc/v35/ee/cbpdf.html>
- <http://www.software.ibm.com/websphere/samples>
- <http://www.ibm.com/software/webservers/appserv/cb/support>

---

## How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** [ibm.com/redbooks](http://ibm.com/redbooks)

Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the IBM Redbooks fax order form to:

|                            |   |
|----------------------------|---|
|                            | <b>e-mail address</b>   |
| In United States or Canada | <a href="mailto:pubscan@us.ibm.com">pubscan@us.ibm.com</a>  |
| Outside North America      | Contact information is in the "How to Order" section at this site:<br><a href="http://www.elink.ibm.link.ibm.com/pbl/pbl">http://www.elink.ibm.link.ibm.com/pbl/pbl</a> |

- **Telephone Orders**

|                           |  |
|---------------------------|--|
| United States (toll free) | 1-800-879-2755   |
| Canada (toll free)        | 1-800-IBM-4YOU   |
| Outside North America     | Country coordinator phone number is in the "How to Order" section at this site:<br><a href="http://www.elink.ibm.link.ibm.com/pbl/pbl">http://www.elink.ibm.link.ibm.com/pbl/pbl</a> |

- **Fax Orders**

|                           |  |
|---------------------------|--|
| United States (toll free) | 1-800-445-9269   |
| Canada                    | 1-403-267-4455   |
| Outside North America     | Fax phone number is in the "How to Order" section at this site:<br><a href="http://www.elink.ibm.link.ibm.com/pbl/pbl">http://www.elink.ibm.link.ibm.com/pbl/pbl</a> |

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

### IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

---

## IBM Redbooks fax order form

Please send me the following:

| Title | Order Number | Quantity |
|-------|--------------|----------|
|       |              |          |
|       |              |          |
|       |              |          |
|       |              |          |
|       |              |          |
|       |              |          |
|       |              |          |
|       |              |          |

---

|            |           |
|------------|-----------|
| First name | Last name |
|------------|-----------|

---

|         |
|---------|
| Company |
|---------|

---

|         |
|---------|
| Address |
|---------|

---

|      |             |         |
|------|-------------|---------|
| City | Postal code | Country |
|------|-------------|---------|

---

|                  |                |            |
|------------------|----------------|------------|
| Telephone number | Telefax number | VAT number |
|------------------|----------------|------------|

---

|   |  |
|---|--|
| <input type="checkbox"/> Invoice to customer number |  |
|---|--|

---

|   |  |
|---|--|
| <input type="checkbox"/> Credit card number |  |
|---|--|

---

|                             |                |           |
|-----------------------------|----------------|-----------|
| Credit card expiration date | Card issued to | Signature |
|-----------------------------|----------------|-----------|

**We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.**



---

## Index

### A

Access bean 127, 128, 129, 182, 217, 218  
-adminNodeName 234  
Application Framework 1, 49, 50, 56  
application server 24, 30, 33, 35, 36, 41, 163  
authentication 38, 39

### B

base redirector 42  
bootstrap 231  
business logic (model) 89, 92

### C

cache 202  
CBCConnector 154, 155  
cbejb 186, 188, 189, 205, 208, 214, 224  
CBToolkit 154  
CCF 70  
CICS connectors 79  
cloning 45  
Command bean 99, 100, 101, 102, 103, 114, 119, 120, 124, 125, 129, 130, 136, 138, 139, 140, 146, 150, 217, 222, 223, 225  
Common Connector Framework (CCF) 68  
Concurrency Service 65  
connector 49, 68, 70, 78  
container 62, 64, 66, 67  
CORBA 8, 9, 10, 11, 38, 66, 70, 156  
CORBA listener port 236, 262  
credential mapping 38  
customer relationship management (CRM) 18

### D

Data Definition Language (DDL) 154, 189  
data object 194, 196  
DataSource 182, 183  
db2cntcs.bnd 202, 203  
db2cntrr.bnd 202, 203  
DCE 33, 36, 38, 99, 156, 164, 165, 166, 167, 171, 175, 177  
DCE client 170  
DDL 189, 190, 214  
decomposition 28  
decomposition node 21, 28

delegation 38, 40  
deployment descriptor 205, 207  
DHTML 25, 50, 51, 53, 77  
Directory and security services node 25  
DMZ 25, 26, 27, 28, 30, 33, 40, 41, 42, 43, 158, 161, 227, 231, 249, 250, 261  
Document Object Model (DOM) 53  
Document Type Definition (DTD) 53  
domain firewall 25, 27, 31, 33, 35, 250, 252  
Domain Name Server (DNS) node 24  
dynamic HTML (DHTML) 51, 53

### E

ejbbind 203, 215, 216  
EJBs 79  
eMarketPlace 4  
Enterprise JavaBeans (EJBs) 76  
Enterprise Solution Structure (ESS) 1  
Enterprise-Out 16  
Entity bean 111, 113, 120, 126, 128, 142, 148  
ERWin 189  
ESS 5  
Externalization Service 65

### F

FinderHelpers 186  
Formatter beans 92, 98, 99

### H

HTML 77, 78, 79  
HTTP 76  
HttpServlet 80  
HttpServletRequest 81  
HttpServletResponse 82  
HttpSession 82

### I

IBM Global Services methodology 1  
IBM Global Sign-On 39  
IBM SecureWay Policy Director 34, 36, 38  
Identity Service 65  
iDXAAServices 214  
IIOP 70, 76, 99, 231  
iMQAAServices 214  
IMS connectors 79

INET Sockets 229  
integration server 23, 26, 27, 28, 29, 30, 33, 35, 36, 164  
interaction controller 89, 93  
Interface Definition Language (IDL) 154  
IPSEC 24  
ivjeb302.jar 182, 187, 188, 219

## J

Java applet 54  
Java Message Service (JMS) 71  
Java Naming and Directory Interface (JNDI) 70, 203  
Java Server Pages 76, 83, 90  
Java servlets 79  
Java Transaction API (JTA) 71  
JavaBeans 76, 79, 86  
JavaScript 50, 52, 54, 79  
jcbAccountC.jar 185  
JDBC 59, 60, 79, 99  
JDBC driver 182, 183  
jetace 205, 206, 207  
JNDI 70, 79, 99, 203, 215  
JSP 78, 79, 80, 81, 82, 83, 84, 85, 86, 90, 91, 92, 95, 97, 98, 99, 112, 114, 118, 119, 125, 129, 130, 132, 133, 136, 137, 139, 140, 146, 223, 224, 225  
JTA driver 10

## L

Language Interoperability 66  
LDAP 33, 39, 40, 70, 79, 239, 248, 254  
LifeCycle Service 64  
Lightweight Third Party Authentication (LTPA) 39, 241  
Location Service Daemon 199  
LSD 231  
LTPA 40, 241

## M

make 198  
mapping helper 196  
model 45  
Model-View-Controller (MVC) 88, 89, 93  
MQ 77  
MQ Application Adaptor 214  
MQ connectors 79  
MQ EJB component 140, 146, 148, 149

mqaajb 205, 206, 207  
MQ-EJB component 214, 219  
MQSeries 158

## N

-nameServiceNodeName 234  
NAT 42  
NetSEAT 38  
Network Address Translation (NAT) 41  
nmake 198, 214

## O

Object Builder 104, 188, 189, 197, 198, 208, 214  
Object Management Group (OMG) 9  
Object Services  
    Externalization Service 64  
Open Servlet Engine (OSE) 41  
-ORBInitialHost 215  
-ORBInitialPort 215  
OSE queue 229  
OSE Remote 35, 41, 227, 230, 232, 234, 261

## P

Page Construction services 78  
page constructor 89, 92  
Pattern Development Kit 2, 6  
persistent data 212  
persistent object 192, 194, 196  
personal digital assistant (PDA) 22, 25, 73, 77  
port 389 242, 254, 255  
port 8081 232  
port 8110 232, 262  
port 900 215, 236, 254, 262  
port 9000 236, 262  
port=8081 233  
port=8110 232  
presentation node 28  
primitive 195  
protocol firewall 25, 27, 33, 35, 249, 250, 252  
proxy server 43  
Public Key Infrastructure (PKI) 24

## Q

Query Service 65  
queues.properties 230

## R

Rational Rose 104, 113, 189  
RDB 201  
RDBConnection 200  
redirector 161  
Remote Method Invocation (RMI) 70  
Remote OSE 40, 41, 42, 161  
Result bean 93, 94, 95, 96, 97, 98, 100, 101, 103  
reverse proxy 41, 43  
RMI 70, 99  
RMI/IIOP 42  
router node 26  
rules.properties 230

## S

schema 189, 190, 191, 192  
schema group 190  
Security Service 64  
Server Group Control Point (SGCP) 43  
Server Group Gateway (SGGW) 44  
servlet redirector 41, 161  
ServletContext 83  
servlets 76, 80, 90  
Session bean 111, 113, 120, 126, 127, 129, 140, 142, 146, 149, 225  
session bean 205, 217  
session management services 79  
SGCP server 44  
SGGW server 44  
single sign-on (SSO) 40  
Smalltalk 156  
somojor.zip 185  
somorbd 199  
somsrsm 204  
SSL 24, 25, 38, 39, 41, 42, 79  
SSO 38, 40  
stand-alone redirector 42  
State beans 100, 102  
stdio.h 211, 212  
system management 154, 156

## T

thin client 17, 51  
Tivoli Management Environment 156  
TME 156  
Transaction Series Family 155  
Transaction Service 65  
two-phase commit 22, 65

## U

UML 104, 108, 113  
Unified Modelling Language (UML) 103  
User node 25

## V

vhosts.properties 230  
View bean 93, 94, 95, 96, 97, 98  
VoiceXML 54

## W

WAP 54  
Web application server 23, 24, 27, 28, 29, 36, 56, 78  
Web application server node 30  
Web server redirector 35  
Web server redirector node 25, 30  
WebTV 77  
Web-Up 15  
Wireless Application Protocol (WAP) 52  
Wireless Markup Language (WML) 52  
WLM 43, 44  
workload management 43, 66

## X

XML 2, 51, 53, 54, 59, 82, 182, 184, 205, 223, 224  
XMLConfig tool 181, 182  
XSL (eXtensible Stylesheet Language) 53, 59



## IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at [ibm.com/redbooks](http://ibm.com/redbooks)
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)

|   |  |
|---|--|
| <b>Document Number</b>  | SG24-6151-00   |
| <b>Redbook Title</b>  | User-to-Business Patterns Using WebSphere Enterprise Edition<br>Patterns for e-business Series   |
| <b>Review</b>   | <div></div> <div></div> <div></div> <div></div> <div></div> <div></div>  |
| <b>What other subjects would you like to see IBM Redbooks address?</b>  | <div></div> <div></div> <div></div>  |
| <b>Please rate your overall satisfaction:</b>   | <input type="radio"/> Very Good <input type="radio"/> Good <input type="radio"/> Average <input type="radio"/> Poor  |
| <b>Please identify yourself as belonging to one of the following groups:</b>  | <input type="radio"/> Customer <input type="radio"/> Business Partner <input type="radio"/> Solution Developer<br><input type="radio"/> IBM, Lotus or Tivoli Employee<br><input type="radio"/> None of the above |
| <b>Your email address:</b><br>The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities. | <input type="radio"/> Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.                                |
| <b>Questions about IBM's privacy policy?</b>  | The following link explains how we protect your personal information.<br><a href="http://ibm.com/privacy/yourprivacy/">ibm.com/privacy/yourprivacy/</a>  |





User-to-Business Patterns Using WebSphere Enterprise Edition Patterns for e-business Series

(0.5" spine)

0.475" <-> 0.875"

250 <-> 459 pages







# User-to-Business Patterns Using WebSphere Enterprise Edition

## Patterns for e-business Series



**Redbooks**

**Select topologies  
and mappings to  
build U2B e-business  
solutions**

**Gain insight into the  
latest technologies,  
design guidelines**

**Learn how to  
implement the  
solution from  
examples**

Patterns for e-business are a group of proven, reusable assets that can help speed the process of developing applications. The pattern discussed in this book, the User-to-Business pattern, is the general case of users interacting with enterprise transactions and data. In particular it is relevant to those enterprises that deal with goods and services which cannot be listed and sold from a catalog.

This redbook discusses two application topologies of the User-to-Business patterns. Application topology 5 links multiple presentation tiers to any back-end client, but the back-end is not hidden to the user. Topology 6 extends topology 5 to describe the situation where multiple presentation tiers are linked to multiple back-end clients. It makes third-tier applications seamless by integrating the business logic at the intermediate tier.

The topologies are illustrated using WebSphere Application Server Advanced Edition V3.021 and Component Broker 3.02. This redbook takes you through the process of choosing an application topology, and gives you possible product mappings for implementation of the chosen runtime topology. It provides a set of guidelines for building your e-business application. It includes information on technology options, application design and application development.

### **INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

#### **BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)