WebSphere Developer Technical Journal

Table of Contents

From the Editor

IBM is celebrating the announcement of the WebSphere software platform for e-business. This new initiative expands IBM's commitment to Web development, providing an integrated environment for rapidly developing and deploying flexible e-business applications. Recently this Technical Journal has broadened its focus to include the WebSphere Application Server as a prime target for server-side Java created using VisualAge for Java and related IBM tools. This month we're officially expanding our coverage of solution-level, cross-product information, while continuing our usual fare of how-to information on using VisualAge for Java. So we're updating our name -- to the WebSphere Developer Technical Journal.

This month's issue kicks off with a live online banking demo running under WebSphere Application Server, and developed using VisualAge for Java. Other articles cover an overview of Websphere's commitment to Java standards, and development patterns for data locking. Finally, we return to VisualAge for Java, with an article on developing GUIs that contain variable-length strings.

Using the Enterprise Java Online Banking Demo

If you read last month's Enterprise Java Tutorial, you know that **Tim Biernat** laid out the architecture for an Online Banking System that uses WebSphere Application Server and VisualAge for Java. This month the demo is up and running - read this article to try it out and get an overview of what's going on behind the scenes.

Enterprise Java Tutorial Part 2: Business Object Design and Implementation

After you've played with the Online Banking Demo, dive into implementation. In this article **Tim Biernat** goes into more detailed design of the model layer of his application. He also shows you how to implement the business objects as Container Managed Persistence (CMP) Entity Beans. Come back next month to see how he implements the controller layer using a lightweight servlet framework.

How WebSphere Supports Java Standards

Jim Ramaker provides a quick reference for the various Java APIs and technologies, and details how WebSphere plans to support each of them. The paper also describes how Java 2 Enterprise Edition (J2EE) is integral to the WebSphere strategy, and how IBM will keep WebSphere in sync with the evolving J2EE application platform. IBM will continue to be a leading contributor to Java platform definitions, while providing an ideal framework through WebSphere to develop, deploy, and manage e-business applications.

[Optimistic Locking Pattern for IBM WebSphere Advanced Edition](#)

In this excerpt from his upcoming Redbook, **Martin Weiss** provides a locking pattern for entity beans that ensures data integrity with short commit cycles, and improves total system throughput and performance. Optimistic locking lets different transactions read the same state concurrently, and checks data integrity constraints at update time. This optimistic locking is part of WebSphere Enterprise Edition 3.02, but not the Advanced Edition (or the EJB spec).

You can also download a zip file containing this pattern and others from the Redbook. A draft of this book is currently available as a "[Redpiece](#)."

[String Externalization with VisualAge for Java, Part 2](#)

In his [previous article](#) on string externalization, **Joe Winchester** covered the basics of separating externalized strings from the application code. In Part 2, he tackles the problem of programming a GUI that allows for variable-length strings. The trick is to create the GUI with layout managers, so that it calculates its dimensions at run time, based on the contents of its labels.

Using the Enterprise Java Online Banking Demo

Tim Biernat
SoftwareMentor.com
June 2000

Enterprise Java Tutorial, Part 1: Architecture and Analysis introduced you to an Online Banking System (OBS) application. This series on Enterprise Java will walk you through building this Web application using VisualAge for Java and WebSphere. This article will show you how to access the Online Banking System demo now available on the Web, which will give you a better feel for the end product we're building. Alternatively, you can download the sample code for the demo.

This article walks you through the OBS demo, explaining how to try out the pages, and what's going on behind the scenes. In case you haven't read the previous article, let's do quick a overview.

This application implements several key use cases of a commercial banking application, using a combination of cutting-edge Web technologies, including:

- Enterprise JavaBeans (EJB 1.1)
- JavaServer Pages (JSP 1.0)
- Java Servlets (Servlet API 2.1)
- Extensible Markup Language (XML 1.0)

Development Environment and Tools

We deployed this demo using the following platforms and development tools:

- Websphere Application Server Advanced Version 3.02
- DB2 Version 6.1
- Windows NT
- VisualAge for Java, Enterprise Edition, Version 3.0 for Windows NT
- TogetherSoft's TogetherJ Version 3.1 modeling software

Architecture

The application extends the reach of existing core commercial banking services to thin-client HTML browsers. Banking acount data resides on legacy backend mainframe machines, subject to existing account reconciliation processing. Middle tiers consist of Java application servers using servlet controllers to coordinate access to EJB session and entity beans. The entity beans employ container-managed persistence (CMP) using a relational database. The OBS architecture uses the proven Model-View-Controller (MVC) strategy for separation of responsibilities and reduced coupling.

Use Cases

Requirements for the commercial banking application are captured as Use Cases. A Use Case defines user interaction with the system from the user's perspective. This demo implements View Account Activity, Upload Transactions, and Update Customer Profile. An Authorized Bank User acts on behalf of a customer. View Account Activity lets the Bank User browse activity on customer accounts. Update Customer Profile lets the Bank User update customer profile information, including address and phone numbers. Finally, Upload Transactions lets the Bank User upload banking transactions for posting to customer accounts. The system validates the upload, and presents summary information back to the Bank User for approval. Upon approval, the system posts transactions to the customer's accounts, for subsequent processing by the Check Services system. For a full description of the requirements, use cases, and architecture for this system, see Enterprise Java Tutorial Part I: Architecture and Analysis

OBS Demo

Now you're ready to check out the running OBS demo. To try out the demo:

1. The first thing you see is a login screen. Go with the default login by clicking the **login** button. This takes you to the main menu, with options for all three use cases implemented in the demo. A simple navigation bar at the top of all screens allows navigation to the Login Page, Main Menu or Help Page (not yet implemented). Keep in mind that several people may be working with the demo concurrently, logged in as the same default user! So you may see additional account activities or customer profile updates that you didn't enter.

2. Try Update Customer Profile by changing some of the Customer profile attributes.

3. Next View Account Activity. First you will see a list of accounts.

4. Click on **activity** to see account details (transactions posted to the account).

5. Navigate back to the Main Menu via the Home icon, and select **Upload Transactions**. You can paste the following XML into a text file on your machine, then upload the file to OBS.

```
<!DOCTYPE xactionFile SYSTEM "banking.dtd">

<xactionFile>

    <customer-number>1</customer-number>

    <account>

        <account-number>3</account-number>

        <xaction>

            <xaction-type>CREDIT</xaction-type>

            <xaction-amount>444.44</xaction-amount>

            <xaction-description>Services Rendered</xaction-description>

        </xaction>

        <xaction>

            <xaction-type>DEBIT</xaction-type>

            <xaction-amount>666.66</xaction-amount>

        </xaction>

    </account>

</xactionFile>
```

6. OBS will parse and validate the XML, presenting you with summary results on the Upload Verification page.

7. Click **post** to process these transactions, then View Account Activity for account #3 to see the results.

What's going on behind the scenes:

Logon

The front door to OSB is the login page. This form is generated by login.jsp, and posts to the main controller, BankingServlet. BankingServlet delegates processing of login requests to the LoginHandler class. LoginHandler searches for a matching user with the User CMP EJB. If a user with this login exists, and the password is valid, the LoginHandler authenticates the user and places the user object in the session. It then locates the user's customer with the Customer CMP EJB, and places the customer object in the session. Finally, control passes to the main.jsp, which returns the Main Menu page. If the login fails, control passes back to login.jsp with a suitable error message. (Details of exception handling and the servlet framework will be covered later in the Enterprise Java Tutorial series.)

View Account Activity

This is the first OBS use case, which lets users browse account activity details. The application uses Account CMP EJBs to locate the user's accounts, and to list account summary information. By clicking **activity** on a particular account, the user can browse account transaction details from the Transaction CMP EJB, including transaction type, description, and date posted.

Upload Transactions

Authorized users can post various transactions to customer accounts through an XML file upload mechanism. A DTD (Document Type Definition) would reside on a Web server at the bank. Users can access these rules and grammar to compose valid XML upload files with their preferred XML tool, then use the OBS to upload the XML file. The UploadHandler retrieves the file, validates and parses the document according to DTD rules, then presents a summary for the user to verify. Upon verification, there is a business requirement to process all transactions in a file as a complete *unit of work* -- all transactions are posted, or none are posted. The PostHandler accomplishes this with a UserTransaction, delineating transactional boundaries to include processing of all the transactions.

Update Customer Profile

The last OBS use case allows authorized users to maintain customer profile information. profile.jsp accesses customer profile data from the Customer Session object to build the Customer Profile page. This form has editable profile fields, and is processed by ProfileHandler, which commits the changes to the Customer CMP EJB.

Conclusion

Now that you've seen the demo in action, you're probably itching to see how it was developed. This monthly tutorial series will walk you through each step, beginning with architecture and high-level design.

If you missed last month's article, Enterprise Java Tutorial, Part 1: Architecture and Analysis, you may want to review it before continuing. Otherwise plunge right into development, in Enterprise Java Tutorial, Part 2: Business Object Design and Implementation.

Download

File Description	Type	Size	Method
Enterprise Java demo	zip	.7 MB	FTP or HTTP

Enterprise Java Tutorial
Part 2: Business Object Design and Implementation

Tim Biernat
SoftwareMentor.com
June 2000

This is the second article in a tutorial series on Enterprise Java. Last month's tutorial covered analysis of the Online Banking System (OBS) to develop a conceptual model. This month we begin looking at system design, plus you can check out a running demo of my final application. For information on using the demo, see Using the Enterprise Java Online Banking Demo. You can also download the sample code for the demo. Now that you've seen the demo in action, let's talk about the OBS design.
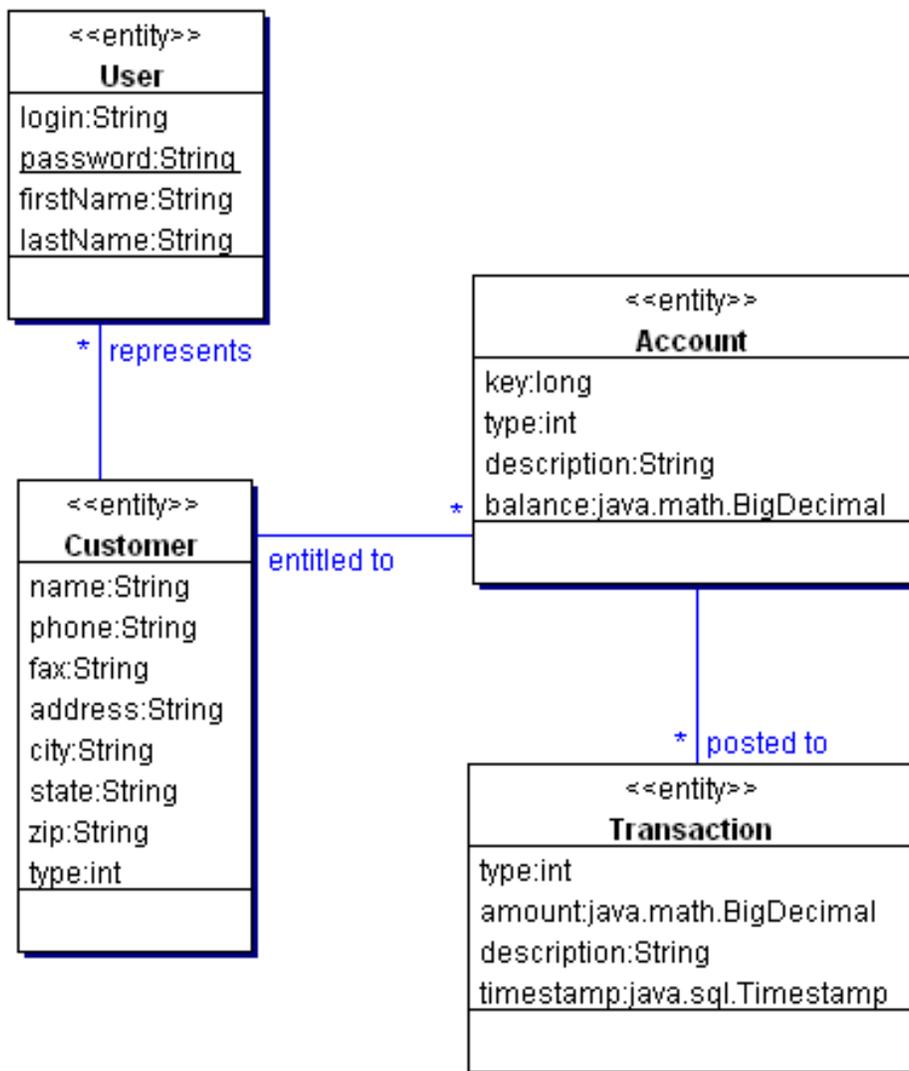
Design

While the conceptual model reflects our understanding of the objects and relationships in the real world, the design model must accommodate constraints and trade-offs encountered in building the actual system. You must maintain persistent objects and their relationships in tables in a database. You also need to cache frequently used data to achieve reasonable performance. Finally, you need to distribute objects to remote clients.

The OBS design leverages EJB and XML technologies and a servlet framework to make our job much easier. Last time I said the OBS design is partitioned into model, view, and controller packages. Let's start with a discussion of the model package design.

Model Package

The model package contains our core business objects, including persistent and distributed objects. Since we have deliberately kept the OBS simple to better focus on the technology, there is a fairly straightforward mapping of business objects from concept to design. As a first pass, we add attributes and review associations. This yields the class diagram in Figure 1:

Figure 1.

<<entity>>
User
login:String
password:String
firstName:String
lastName:String

<<entity>>
Account
key:long
type:int
description:String
balance:java.math.BigDecimal

<<entity>>
Customer
name:String
phone:String
fax:String
address:String
city:String
state:String
zip:String
type:int

<<entity>>
Transaction
type:int
amount:java.math.BigDecimal
description:String
timestamp:java.sql.Timestamp

* represents

entitled to *

* posted to

These four classes make up our core business objects, and will be implemented with Enterprise JavaBean technology as Container Managed Persistence (CMP) Entity Beans. At this point, I generally create entity beans using a tool like Visual Age for Java, working directly from the UML diagram. VisualAge for Java has wizards for EJB creation that speed this process. In a new design like OBS, we employ a "top-down" approach, first creating the EJBs, then letting the tool map the beans into database tables (VisualAge for Java will even generate the tables for you). When working with legacy data, use a "bottom-up" approach, in which you import your existing database schema is imported into VisualAge for Java, and automatically generate EJBs based on that schema. Before we create the EJBs, let's cover a few EJB technology caveats.

EJB Technology Issues

The EJB specification does not define a standard way to implement EJB associations or inheritance. IBM provides mechanisms for implementing EJB associations in VisualAge for Java, Enterprise Edition as a Technology Preview. Obviously this could be subject to changes in future EJB specs. VisualAge for Java also supports an approach to EJB inheritance where you can extend Remote interfaces and Bean classes. Again, the EJB spec does not define a standard way to implement inheritance, so this may also change.

In my experience with VisualAge for Java 3.02 on several projects, IBM's implementation of EJB associations is quite good, and I can recommend this approach when using WebSphere Application Server 3.02. I have experienced some difficulties using the VisualAge for Java EJB inheritance mechanism in combination with EJB associations (GoF Composite Pattern). If you are considering using VisualAge for Java EJB inheritance, I strongly urge you to develop a representative prototype first to ensure that the implementation will address your needs. The OBS uses the VisualAge for Java EJB association support.

When creating CMP EJBs, we must uniquely identify our persistent objects. This is typically accomplished using primary keys in the underlying tables. The primary key is mapped to a CMP field in the EJB. Unfortunately, there is no standard

mechanism in the EJB specification generating these keys. As we will see, the OBS implementation generates new keys in our EJB create() methods by first finding the newest EJB instance in the underlying table, then incrementing that instance's key value. This new key value will be unique, and is used as the primary key in persisting the new instance.

VisualAge for Java EJBs

Now we implement our business objects as EJBs in VisualAge for Java:

1. Create a **Banking Example** project in VisualAge for Java that includes a **banking.model.ejb** package, and be sure to add the EJB Development feature to your Workspace.

2. Moving to the EJB tab view, create an EJB Group named **Banking EJBs**, using the Banking Example project.

3. Create CMP entity EJBs in this group using the banking.model.ejb package, for **Customer**, **User**, **Account**, and **Transaction**, based on the UML in Figure 1.

4. Select **Create Finder Helper** for each. All CMP fields must be public per the EJB spec.

5. Create getters/setters for all CMP fields, and promote them to the Remote interface. For User, make **login** the key field. For all others, define **key** as a key field. No errors should be flagged in these EJBs when you are done, as shown in Figure 2:
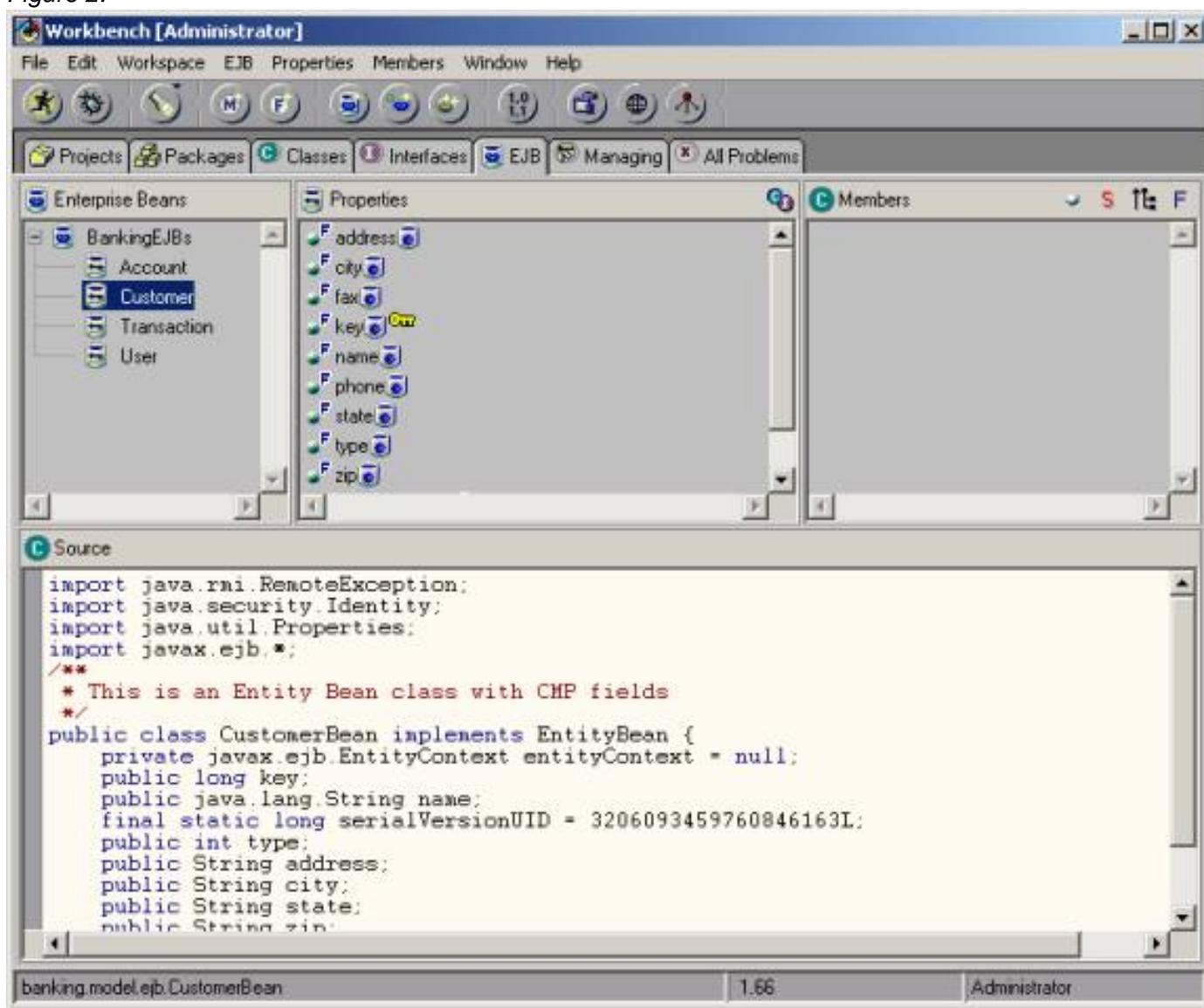
Figure 2.

6. Select Customer, right-click to **Add Association**, and create the one-to-many association with User (Figure 3). This satisfies the OBS business rule that a Customer can be represented by multiple Users. The Navigable, Many, and Required settings apply to the respective *role*. All EJB associations in WebSphere should be navigable, that is reachable from either role in the association. So we specify the **Role of Customer** to be **Navigable** and **Required** (the One side of the One-to-Many association). We specify the **Role of User** to be **Navigable** and **Many** (the Many

side of the One-to-Many association). A Customer does not require a User (**Required** not checked) in the association, meaning we can set up a Customer, then set up the Customer's Users later.

7. Do the same for Customer-Account and Account-Transaction:

Figure 3.

VisualAge for Java EJB code generators created bean and key classes, and home and remote interfaces. So far we have not written a single line of code, but that's about to change. Next we implement the required EJB lifecycle methods. For CMP beans, the container does much of the work, and our primary responsibility is to define appropriate create methods and finders. You usually discover the need for these methods through design interaction diagrams.

We will have to create lifecycle methods to automatically generate keys for new Customer, Account, and Transaction instances. Create methods are implemented in the Bean class as ejbCreate, and have many of the same responsibilities as Java constructors.

Customer instance

Design sequence diagrams identified a need for a CustomerHome create(String name, int type)> method. The implementation in CustomerBean is as follows:

public void ejbCreate(String name, int type)

throws javax.ejb.CreateException, java.rmi.RemoteException {

    try {

        key = getNextKey();

    }

    catch (javax.naming.NamingException fe) {

        throw new CreateException("Naming exception encountered creating Customer.");

    }

```
    catch (FinderException fe) {

        throw new CreateException("Finder exception encountered creating Customer.");

    }

    setName(name);

    setType(type);

}
```

This method uses getNextKey() to set the key value for this instance, then sets name and type attributes. Let's look at getNextKey():

```
protected long getNextKey()

throws RemoteException, FinderException, javax.naming.NamingException {


long nextKey = 0;


try {

Customer newestCustomer = ((CustomerHome)

entityContext.getEJBHome()).findNewest();

nextKey = ((CustomerKey)newestCustomer.getPrimaryKey()).key + 1;

}

// if not found, assume first one

catch (ObjectNotFoundException onfe) {

nextKey = 1;     }

return nextKey;

}
```

This method is protected and only used internally, and it relies on a finder: findNewest(). In VisualAge for Java, finders are implemented with SQL hints in a FinderHelper interface. For the Customer bean this looks like:

```
public interface CustomerBeanFinderHelper {

    public final static String findNewestWhereClause = "KEY = (SELECT MAX(KEY) FROM CUSTOMER)";

}
```

Next we examine the user create method. Every User must have a Customer per our UML model, so we need a Customer reference, because we do not want multiple users with the same login name:

```
public void ejbCreate(String login, String password, String firstName, String lastName,

CustomerKey customerKey)

    throws javax.ejb.CreateException, java.rmi.RemoteException {


    // only create if user login name does NOT exist
```

```
    try {

        ((UserHome)entityContext.getEJBHome()).findByPrimaryKey(new UserKey(login));

        throw new CreateException("login name exists.");

    }

    catch(FinderException fe) {

        setLogin(login);

        setPassword(password);

        setFirstName(firstName);

        setLastName(lastName);

        privateSetCustomerKey(customerKey);

        return;

    }

}
```

The findByPrimaryKey throws a FinderException if the login is unique, and then we initialize user fields and set the reference to customer.

Account instance

Next is the Account create method. The model defines that an account requires a Customer reference. Furthermore, business requirements demand that accounts may only be established on behalf of real, active customers. We use the same key generation scheme as for Customer:

```
public void ejbCreate(int type, String description, java.math.BigDecimal balance,

CustomerKey customerKey)

    throws javax.ejb.CreateException, java.rmi.RemoteException {


    try {

        key = getNextKey();

    }

    catch (javax.naming.NamingException fe) {

        throw new CreateException("Naming exception encountered creating Account.");

    }

    catch (FinderException fe) {

        throw new CreateException("Finder exception encountered creating Account.");

    }

    setType(type);

    setDescription(description);

    setBalance(balance);
```

```
        privateSetCustomerKey(customerKey);


    try {

        // check if customer found

        getCustomer();

    }

    catch (FinderException fe) {

        throw new CreateException(fe.getMessage());

    }

}
```

getCustomer() was generated for us when we established the Customer-Account association.

Transaction instance

Last we have the Transaction create method. Again the model defines a relationship that requires an Account, and we use the same key generation scheme:

```
public void ejbCreate(String type, java.math.BigDecimal amount, String description,

    AccountKey accountKey)

    throws javax.ejb.CreateException, java.rmi.RemoteException {


    try {

        key = getNextKey();

        setTimestamp(new java.sql.Timestamp(System.currentTimeMillis()));

        setType(type);

        setAmount(amount);

        setDescription(description);

        privateSetAccountKey(accountKey);


        if (type.trim().equalsIgnoreCase(Transaction.CREDIT)) {

            setType(Transaction.CREDIT);

            creditAccount(accountKey, amount);

        }

        else if (type.trim().equalsIgnoreCase(Transaction.DEBIT)) {

            setType(Transaction.DEBIT);

            debitAccount(accountKey, amount);

        }
```

```java
        else {
            throw new CreateException("Illegal transaction type: " + type + ".");
        }
    }
    catch (javax.naming.NamingException ne) {
        throw new CreateException(ne.getMessage());
    }
    catch (FinderException fe) {
        throw new CreateException(fe.getMessage());
    }
}
```

If the account does not exist, an attempt to credit or debit throws a FinderException:

```java
protected void creditAccount(AccountKey accountKey, java.math.BigDecimal amount)
    throws RemoteException, FinderException, javax.naming.NamingException {


    AccountHome accountHome = getAccountHome();
    Account account = accountHome.findByPrimaryKey(accountKey);
    java.math.BigDecimal balance = account.getBalance();
    balance = balance.add(amount);
    account.setBalance(balance);
}


protected void debitAccount(AccountKey accountKey, java.math.BigDecimal amount)
    throws RemoteException, FinderException, javax.naming.NamingException {


    AccountHome accountHome = getAccountHome();
    Account account = accountHome.findByPrimaryKey(accountKey);
    java.math.BigDecimal balance = account.getBalance();
    balance = balance.subtract(amount);
    account.setBalance(balance);
}
```

At this point, our core business objects are completely implemented, and should be tested. Perform the following steps:
1. Select the **BankingEJBs** group, right-click and select **Add Schema and Map.**
2. Select the **BankingEJBs** group, right-click and select **Generate Deployed Code**, then **Generate Test Client**.
3. Select the **BankingEJBs** group, right-click and select **Add To Server Configuration**.

4. Select **EJB Server** in the Server Configuration window, right-click and select **Properties**. Configure access to a database where the EJBs can be persisted, as shown in Figure 4:

Figure 4.
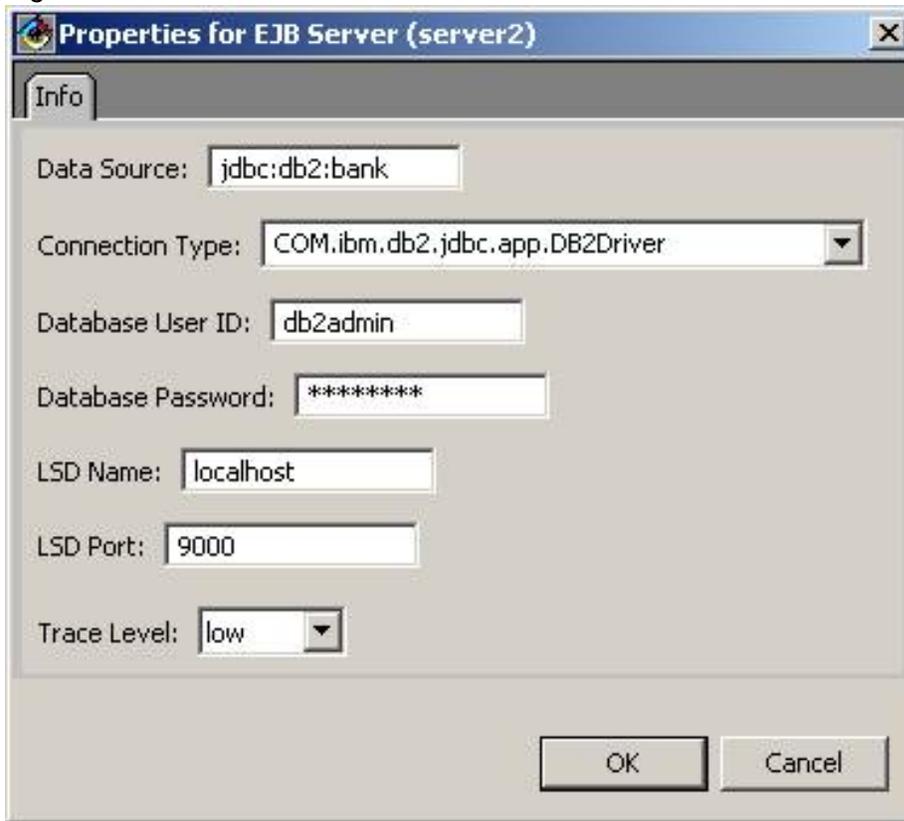
5. Select **EJB Server** in the Server Configuration window, and right-click and select **Create Database Tables**. The previously generated schema is used to create the required EJB tables in the database.

6. Start the **Location Service Daemon** and **Persistent Name Server**, then start the **EJB Server**.

7. Select **EJB Server**, and expand **BankingEJBs**. Select **Customer**, right-click and select **Run Test Client.Connect**, which retrieves a proxy to a CustomerHome. Select the **create(String, int)** method, and provide appropriate parameters; then click **Send**. The container creates a new Customer instance, and provides a proxy to that instance. Exercise the getters/setters to ensure all is well. **Send** getPrimaryKey(), and inspect it to ensure the wrapped value is 1.

8. Select the other EJBs and run their test clients in the same way. You need to provide appropriate keys to existing beans to satisfy the association requirements. Creating a user requires a valid Customer reference. In the User create dialog for CustomerKey select **New**, then the **new Customer(long)** constructor, providing a long value of 1 to reference the first Customer created.

Summary

We've completed the design and implementation of the business objects in the model layer of our MVC-based system. You can download the source code below. Next month we'll look at design and implementation of the controller layer, and use of a servlet framework.

Download

File Description	Type	Size	Method
Enterprise Java demo	zip	.7 MB	FTP or HTTP

Related Information

⇢ Which download Method should I choose?

⇢ Download FAQs

About Tim Biernat

Tim Biernat has held various research and development positions at General Dynamics and Motorola, and has taught internationally for IBM and Learning Tree International. Tim was trained as an electrical engineer at Marquette University and the University of Texas, and his professional experience spans 17 years, including research in fault tolerant real-time avionics software, and development of many distributed object business systems. Currently, Tim is principal of softwareMentor.com, a distributed object technology consulting firm based in Milwaukee WI.

How WebSphere Supports Java Standards

Jim Ramaker
June 2000

Introduction

The WebSphere software platform for e-business strongly supports the full range of Java standards and technologies. In particular, it unequivocally supports the J2EE specification, with full implementation based on a schedule described below. This support for Java helps make the WebSphere software platform an ideal, open framework to develop, deploy, and manage e-business applications. This paper gives you a quick reference to the various Java standards and technologies, and then describes how WebSphere supports and leverages each of them, both in currently available versions of WebSphere Application Server and in versions now being developed.

Java 2 Standards

The table below describes the Java 2 Enterprise Edition (J2EE) platform. J2EE, a Sun brand, is the Java standard for enterprise application development and deployment. The platform specifications in the table below include the Java 2 APIs and technologies required to provide the minimum quality of service, compatibility, portability, and integration defined by J2EE.

In addition to these J2EE platform specifications, J2EE includes three other deliverables:

J2EE compatibility test suite

Verifies that a Java implementation complies with the J2EE platform specs.

J2EE reference implementation

Provides an operational definition of the J2EE platform and demonstrates its capabilities.

J2EE blueprints

Provide guidance on building enterprise applications that can run on a J2EE platform. J2EE blueprints focus on the development of JSPs, Servlets, and EJBs, which use resources such as JDBC and JNDI to run the applications.

Table 1. Java 2 Platform Summary

Technology	Description
EJB (Enterprise JavaBeans)	Server transactional components that are reusable and provide portability across application servers while implementing transaction services.
JAF (JavaBeans Application Framework)	Standard services to determine the type of an arbitrary piece of data and activate an appropriate bean component to manipulate the data.
JavaIDL (Java Interface Definition Language)	Creates remote interfaces to support Java-to-CORBA application communications. JavaIDL includes an IDL-to-Java compiler and an ORB (Object Request Broker) that supports IIOP.

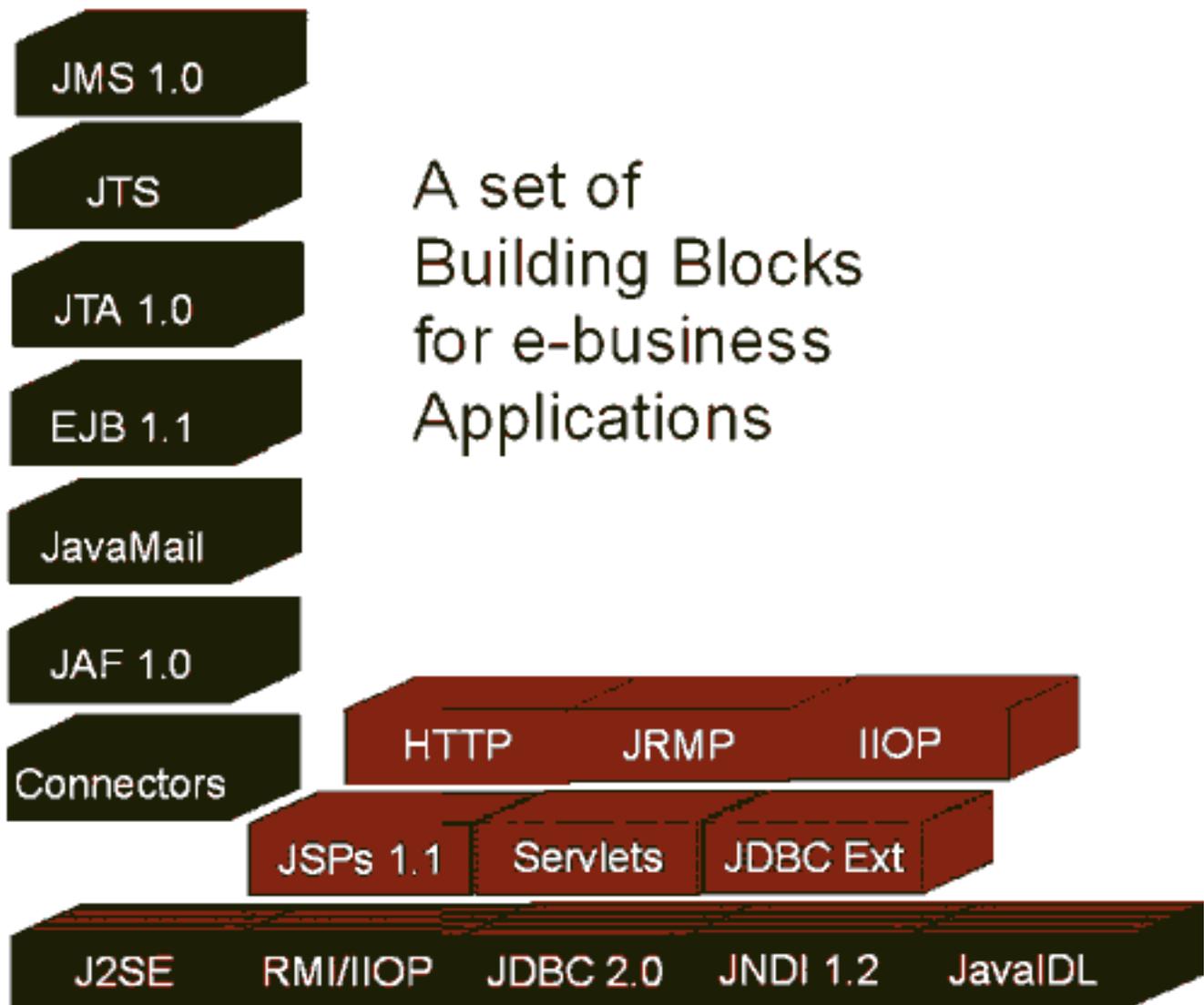|	
JavaMail	Provides a protocol-independent framework to build mail and messaging applications. Requires the JavaBeans Application Framework API.
JDBC (Java Database Connectivity)	The JDBC database access API provides uniform access to relational databases such as DB2, Informix, Oracle, Sybase, SQL Server, etc.
JDK (Java Developer's Kit)	Provides the JVM (Java Virtual Machine) base with Java classes and basic routines required to execute Java applications. The JDK is now referred to as the Java 2 Platform Standard Edition, and it includes the Java 2 SDK Standard Edition and the Java 2 Run-time Environment Standard Edition. For versions prior to 1.2, the Java 2 SDK is called the JDK -- JDK 1.0 and JDK 1.1.
JMS (Java Messaging Service)	Supports asynchronous communications using either a reliable queuing or publish/subscribe programming model.
JNDI (Java Naming and Directory Interface)	Provides access to naming and directory services such as DNS, LDAP, Novell Directory Services, and CORBA COSNaming.
JSP (JavaServer Pages)	A simple, fast, and consistent way to extend Web server functionality and create dynamic Web content. JSP technology enables rapid development of Web applications that are server and platform independent.
JTS/JTA (Java Transaction Service / Java Transaction API)	A distributed transaction management service and associated API based on the CORBA Object Transaction Service.
RMI/IIOP (Remote Method Invocation / Internet Inter-ORB Protocol)	RMI creates remote interfaces for Java-to-Java application communications. The RMI/IIOP extension uses the CORBA-standard IIOP for communications.
Servlet	Servlets are server applications that execute within a Web application server that supports dynamic HTML. The Java Servlet API gives Web developers a simple, consistent way to extend the functionality of a Web server.
XML (Extended Markup Language)	As it relates to Java 2, XML provides an extended format for deployment descriptors within Enterprise JavaBeans for J2EE.

The wide range of application technologies and APIs shown in the table above can communicate with a variety of systems, servers, and applications that might exist within a given network.

The J2EE platform is still evolving, and future enhancements will include EJB security interoperability, security

auditing, transaction interoperability across multiple products, IIOP client access, and legacy connections. Current WebSphere development efforts are already incorporating some of these enhancements.

The graphic below shows the J2EE technologies as a set of building blocks for applications. The technologies in the horizontal bars form the basis of all J2EE client and server e-business applications. Those on the vertical bar may or may not be used, depending on the specific application requirements and network topology. For example, JMS would be needed for message queuing with a product such as MQSeries, while JTS/JTA would be needed for a transactional system.

Java Enterprise Technologies

JMS 1.0
JTS
JTA 1.0
EJB 1.1
JavaMail
JAF 1.0
Connectors

A set of
Building Blocks
for e-business
Applications

HTTP   JRMP   IIOP

JSPs 1.1   Servlets   JDBC Ext

J2SE   RMI/IIOP   JDBC 2.0   JNDI 1.2   JavaIDL

WebSphere and the Java 2 Standards

The table below shows how WebSphere (specifically WebSphere Application Server) stacks up against the full range of Java 2 APIs and technologies. Here are explanations of the four columns:

Java 2 Technologies and Profile Requirements

Lists the Java 2 API or technology. Acronyms not defined above in Table 1 are defined here.

Current Version

Indicates the current version of the Java 2 API or technology.

WebSphere Today and Version 3.5

Describes the level of support in WebSphere Application Server 3.02 (available now) and 3.5 (available 3Q2000)

WebSphere Version 4

Describes the level of support in WebSphere Application Server 4.0 (available early 2001)

Following the table, WebSphere support for Java 2 Standards is described in more detail.

Table 2. WebSphere Server Profile

Java 2 Technologies and Profile Requirements	Current Version	WebSphere Today and Version 3.5	WebSphere Version 4
EJB	1.1	1.0 plus extensions (RMI/IIOP, transactions, CMP, etc.)	1.1 including XML descriptor support
HTTP	1.1	Yes, plus across multiple Web servers	Yes, plus across multiple Web servers
JavaIDL/CORBA		Enterprise Edition supports 10 different CORBA services. WebSphere also supports JavaIDL.	Same as 3.5
JavaMail	1.1	No, but supports third-party	Yes, plus Domino support
JDBC	2.0	Yes, across all databases plus partnership with Merant	Yes, 2PC across heterogeneous databases
JDK	1.2	In 3.5, JDK 1.2.2 across Win NT, Win 2000, AIX, Solaris, HP-UX	Upgrade to 1.3 and include Linux, OS/400, OS/390
JMS	1.0.1	Yes, MQSeries native support	Yes, plus JMS-based EJBs, Message Beans
JNDI	1.2	JNDI 1.1 for EJB lookup and COSNaming	Yes, plus JNDI over LDAP

JRMP (Java Remote Message Platform)	1.0	Fully supported via RMI/IIOP	Fully supported via RMI/IIOP
JSP	1.1	JSP 1.0 and 0.91 API levels with some 1.1 features, investigating earlier 1.1	JSP 1.1
JTS/JTA	1.0	Yes, with distributed transactions	Yes, with distributed transactions
LDAP (Lightweight Directory Access Protocol)		Client/server support for LDAP; Domino, Netscape, Novell, IBM	Same as 3.5
RMI/IIOP	1.0	Yes, fully supported, and not required until 12/00	Yes, fully supported
Servlet	2.2	Servlet 2.1 API with many 2.2 features, investigating earlier 2.2	Servlet 2.2 API
SQLJ		Support via DB2 UDB	Support via DB2 UDB
SSL Security (Secure Sockets Layer) **JCE** (Java Cryptography Extension)	2.0 1.2	Complete end-to-end EJB security, partial Java security APIs	Complete, except for JCE
XML DOM/SAX (Document Object Model / Simple API for XML) **XSL** (Extensible Stylesheet Language)		IBM is industry leader	Perf. in XML Parser, XML in EJBs

EJB

WebSphere Advanced Edition and Enterprise Edition fully support EJB 1.0. They also support container-managed persistence, which is optional in the EJB 1.0 spec. WebSphere Enterprise Edition provides CMP, and EJB persistence into DBMSs, CICS, IMS, and SAP.

WebSphere today supports about 90% of the EJB 1.1 features. For example, WebSphere uses the EJB 1.1 package for distributed transactions, and restricts bean-managed transactions to session beans as required by the EJB 1.1 spec. Full EJB 1.1 support is planned for WebSphere Version 4 in early 2001, by which time the Sun reference implementations will become standardized and more stable.

In the area of EJB development, the WebSphere software platform leads the industry, with our award-winning integrated development environment, VisualAge for Java, Enterprise Edition. It provides a complete EJB-based development and test environment. Entity beans can be mapped to databases, and EJBs can be generated that tie into transaction processing systems and MQSeries.

VisualAge for Java Enterprise Edition supports additional advanced EJB capabilities, enabling EJBs to communicate with one another and share key security and user data through associations and inheritance. The VisualAge for Java Enterprise Access Builder generates EJB components that are in turn deployed into VisualAge for Java's test environment -- WebSphere Application Server.

Some parts of the EJB 1.1 spec have yet to be addressed by WebSphere, such as support for XML deployment descriptors, more advanced use of JNDI within the EJB deployment environment, and additional Java security APIs.

For more details, see the IBM Redbook Design and Implement Servlets/JSPs/EJBs for IBM WebSphere, (SG24-5754).

JavaIDL/CORBA

WebSphere Enterprise Edition supports JavaIDL today, along with support for the complete list of all 10 CORBA services: Concurrency, Event, Externalization, Identity, Lifecycle, Naming, Notification, Query, Security, and Transaction. WebSphere 3.5 Advanced Edition will also add support for JavaIDL.

JavaMail

WebSphere does not currently support JavaMail, although it does support third-party mail tools. WebSphere Version 4 will include JavaMail support.

JDBC

WebSphere 3.5 Advanced Edition and Enterprise Edition include full JDBC 2.0 support for a wide range of JDBC 2.0-compliant relational database systems, both as container-managed persistence databases and with distributed, heterogeneous transactions. The list of databases includes DB2, Informix, Microsoft SQL Server, Oracle, Sybase, Versant, and others. WebSphere distributes DB2 as part of the product for use as persistent storage. It supports distributed database transactions across homogeneous DB2, Oracle, or Sybase servers.

JDK

WebSphere Application Server Version 3.02, available now, fully supports JDK 1.1.8+. WebSphere Application Server Version 3.5, available later this summer, fully supports JDK 1.2.2 (Java 2 Standard Edition) on Windows NT, Windows 2000, AIX, Solaris, and HP-UX. Support for Linux, OS/390, and OS/400 will be added later this year.

While other JDK providers have claimed Java 2 support for several months now, IBM's goal with its JDKs is to provide extensions that significantly improve stability and performance, instead of supporting the latest JDK level but burdening developers with possible stability and performance problems. In addition, IBM has "downported" many JDK 1.2 features into JDK 1.1.X.

The VolanoMark 2.1.2 benchmark of messages per second shows that the IBM JDK 1.1.7 on Windows has the fastest JDK performance now available -- better than the Microsoft SDK 3.1 and the Sun JDK 1.2. Here are some brief evaluations of IBM JDKs from independent users.

For more information on IBM's JDKs and their performance, see the VisualAge Developer Domain Library or the IBM developerWorks Java zone.

JMS

JMS is another area where IBM goes beyond the current Java/J2EE standard. IBM's MQSeries product offers

a superset of the functionality in the JMS 1.0.1 spec. MQSeries is fully JMS compatible and will be shipped with WebSphere Application Server Advanced Edition and Enterprise Edition Version 3.5. MQSeries is the world's leading message queuing product with a 70+% market share, and is used by the majority of Fortune 2000 companies.

WebSphere Version 4.0 will offer tighter integration of MQSeries, and additional MQ/JMS EJB transactional support, including JMS-based EJBs.

JNDI

JNDI 1.2 is a major upgrade that includes support for event notification and LDAP Version 3 extensions and controls. WebSphere supports JNDI 1.1 today for EJB lookup, with plans to support JNDI over LDAP in WebSphere Version 3.5.

JSP

WebSphere fully supports JSP 1.0, and maintains support for JSP 0.91 for the many existing JSPs written at the 0.91 level. WebSphere Version 4.0 will support JSP 1.1. IBM provided the reference implementations for JSPs.

Just as with the JDK levels described above, IBM's goal with the JSP (and Servlet) standards is to not so much to meet the latest version of the standard as to deliver real business value to developers and customers today through productive development environments (such as VisualAge for Java and WebSphere Studio), and stable run times (such as WebSphere Application Server).

JTS/JTA

WebSphere supports JTS/JTA 1.0.

As part of the reference implementation for the J2EE, IBM has implemented a transaction manager that supports JTS/JTA. JTS/JTA lets application servers built on the J2EE platform take the burden of transaction management off the component developer. Developers can define the transactional properties of EJB-based components during design or deployment, using declarative statements in the deployment descriptor. The application server takes over the transaction management responsibilities.

JTA specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.

A JTS Transaction Manager provides transaction services to the parties involved in distributed transactions: the application server, the resource manager, the standalone transactional application, and the Communication Resource Manager (CRM).

WebSphere supports distributed transactions with EJB technology, where EJBs running in WebSphere can participate within the same transaction and utilize two-phase commit. WebSphere Advanced Edition focuses on providing this across databases, whereas WebSphere Enterprise Edition focuses on transaction processing systems and MQSeries. Complete EJB distributed transaction support with EJB 1.1 will be part of WebSphere Application Server Version 4.

RMI/IIOP

RMI/IIOP speeds distributed application development by letting developers work completely in Java. When using RMI/IIOP to produce Java-based distributed applications, there is no separate IDL or mapping to learn. Like RMI, this provides flexibility by allowing developers to pass any serializable Java object (referred to as "objects by value") between application components. Like CORBA, RMI/IIOP is based on open standards defined with the participation of hundreds of vendors and users in the OMG. Like CORBA, RMI/IIOP uses IIOP as its communication protocol. IIOP eases legacy application and platform integration by allowing application components written in C++, Smalltalk, and other CORBA-supported languages to communicate

with components running on the Java platform.

With RMI/IIOP, developers can write remote interfaces in Java and implement them just using Java technology and the Java RMI APIs. These interfaces can be implemented in any other language supported by an OMG mapping and a vendor supplied ORB for that language. Similarly, clients can be written in other languages using IDL derived from the remote Java technology-based interfaces. Using RMI/IIOP, objects can be passed both by reference and by value over IIOP. It combines the usability of Java RMI with the interoperability of the IIOP. It provides you with a powerful environment in which to do distributed programming in Java. WebSphere fully supports RMI/IIOP for communications.

Servlets

WebSphere Version 3.02 supports the Servlet 2.1 API. WebSphere Version 4 will fully support the 2.2 API. IBM provided the reference implementations for Servlets. For more information, see the IBM Redbook Design and Implement Servlets/JSPs/EJBs for IBM WebSphere, (SG24-5754).

Security

WebSphere supports secure integration with the Web server (HTTP/SSL), and also supports SSL directly to the application server for Java clients (RMI/IIOP/SSL). WebSphere also supports third-party authentication (including a lightweight version) for secure delegation. WebSphere also supports distributed registries for Authentication, Method Level Authorization, Single Sign-On, existing pluggable registries, LDAP schema, secure delegation between servers, and distributed authorization (ACLs) for EJBs, Servlets, JSPs and HTML content.

For more information on WebSphere security, see the white paper IBM WebSphere Standard/Advanced 3.02 Security Overview.

XML/XSL

WebSphere supports the latest specs -- DOM 1.0 and SAX 1.0. IBM is an industry leader in XML, and WebSphere includes complete XML support for parsing and understanding information sources that utilize XML tags. WebSphere Version 4 will include XML development and deployment descriptor support within EJBs.

For more information on IBM's XML support, including free downloads of XML parsers and tools, see the IBM developerWorks XML zone.

Related Links

- IBM developerWorks
- VisualAge Developer Domain -- WebSphere Java Zone
- Java 2 Platform, Enterprise Edition -- Overview of Technologies [Sun site]
- Java 2 Platform, Enterprise Edition -- Downloads and Specifications [Sun site]

About Jim Ramaker

Jim Ramaker is the Web site editor for IBM's [VisualAge Developer Domain](#), the product community and download site for VisualAge for Java.

E-mail: ramaker@us.ibm.com.

Optimistic Locking Patterns for IBM WebSphere Advanced Edition

Martin Weiss,
IBM Global Services, Switzerland
June 2000

Introduction

This article is part of the IBM Redbook Design and Implement Servlets/JSPs/EJBs for IBM WebSphere. The completed Redbook will be available in July 2000.

In a multi-user environment, database managers use locking mechanisms to handle resource allocation without compromising data integrity. Because every request in a Web application is a separate entity, applications that need to maintain information between requests must decide how to preserve state. With pessimistic locking, data is locked on the database when read for update. This can lead to high lock contention, because copying the data to the browser (which causes an exclusive lock on the database row) and updating the changed data belong to different user interactions. Optimistic locking lets different transactions read the same state concurrently, and checks data integrity only at update time. Optimistic locking enables data integrity with short lock cycles, reduces overall locking overhead, and improves system throughput and performance.

Obviously, we expect optimistic locking support for EJB Entity Beans as part of the application server infrastructure (transparent container service). IBM WebSphere Enterprise Edition (Component Broker) provides support for optimistic locking through its caching service by mouse-click. This optimistic locking support for entities is not part of the EJB specifications and is not included in WebSphere Advanced Edition 3.02.1. But you can download a sample application that includes the proposed optimistic locking pattern and other patterns introduced in the IBM Redbook Design and Implement Servlets/JSPs/EJBs for IBM WebSphere. This article provides an application pattern for run times without optimistic locking infrastructure support. A user request may start a transaction that should be terminated before returning an answer to the user. As a general rule, transactions (and associated locks) should not span user think-time. This article discusses the proposed locking mechanism based on the following example:

A user needs to update customer data via the following steps:

1. Gets a copy of customer data (first transaction)
2. Modifies this copy
3. Sends the copy to the server to make the update permanent (second transaction)

Another user could concurrently access the same data for update, as described in this scenario:

1. User A requests a copy of Customer A's data
2. User B requests a copy of Customer A's data
3. User A changes Customer A's name from `Kurt Weiss' to `Martin Weiss'
4. User B changes Customer A's birthday from 57/05/01 to 55/08/08
5. User A sends the changed copy for server-side update
6. User B sends the changed copy for server-side update

How can we ensure that updates done by User B don't override those of User A? The answer resides in the implementation of an update service. This additional service checks that the persistent customer data has not been modified since a copy was sent to the user.

Applicability

IBM WebSphere Advanced Edition with Commit option C (see the Sun Specifications for Enterprise JavaBeans).

Components

- OptimisticLockService (package wmt.pattern.optimistic.ejb)
  - Generic, unified copy/update service for EJB Entity Beans
  - Implemented as EJB Stateless Session Bean
- EJB Entity home
  - findByPrimaryKey (for copy services)
  - Custom finder (for update services)
  - For example: CustomerHome
- OptimisticLockable interface (package wmt.pattern.optimistic.etc)
- Remote interface of an Optimistic Lockable EJB Entity Bean
  - Extends OptimisticLockable
  - For example: Customer
- Optimistic Lockable Entity Bean implementation

- ❍ Implements copy / update method (see OptimisticLockable interface)
- ❍ Accessed concurrently from different transactions
- ❍ Timestamp field
- ❍ For example: CustomerBean
- OptimisticLockableEntityCopy (package wmt.pattern.optimistic.etc)
  - ❍ Abstract base class for an EJB Entity Bean copy
  - ❍ Implements java.io.Serializable
- EJB Entity Bean copy
  - ❍ Extends OptimisticLockableEntityCopy
  - ❍ Serialized copy of the EJB Entity Bean state
  - ❍ For example: OptimisticLockableCustomerCopy
- Database manager
  - ❍ Responsible for managing concurrency with Commit Option C
  - ❍ Timestamp setting via insert and update triggers

Implementation

This section describes the Optimistic Locking pattern for WebSphere Advanced Edition. The pattern is based on an OptimisticLockService implemented as a Stateless Session Bean. The OptimisticLockService is a generic, unified session facade for copying and updating all optimistic lockable entities. The following sections describe the classes and interfaces required to implement the OptimisticLockService bean.

Optimistic Lockable Entity Bean

1. The remote interface of an Optimistic Lockable Entity Bean extends from the OptimisticLockable interface (Example 1). The entity bean implements the copy and update methods:

   Example 1. OptimisticLockable interface

   ```
   public interface OptimisticLockable {
       public OptimisticLockableEntityCopy copy() throws java.rmi.RemoteException;
       public void update(OptimisticLockableEntityCopy entityCopy) throws java.rmi.RemoteException;
   }
   ```

2. You also need a java.sql.Timestamp field (lastModified) in the implementation of the bean. The timestamp represents the creation time or last update of the bean's state. As the timestamp is a persistent field, it has to be part of the underlying database table. The database triggers are responsible for setting the timestamp. For each underlying database table, we define two triggers: a before_insert trigger and a before_update trigger. See Database triggers for timestamp setting for a concrete sample (Example 7). This approach makes the application more scalable. Delegating the timestamp setting to the bean implementation would rely on synchronized clocks (application server clones). The approach with the database triggers is application server independent. We do not implement a setter for the timestamp field in the bean.

3. For the update service: a custom finder in the entity's home implementing a read-for-update (with an SQL clause "FOR UPDATE"). The parameter list includes:
   1. Primary key fields of the bean
   2. Timestamp

For the copy service we use the findByPrimaryKey finder method of the bean's home.

Implementation of a copy class for each optimistic lockable entity

The copy class extends from OptimisticLockableEntityCopy class:

Example 2. OptimisticLockableEntityCopy class

```
public abstract class OptimisticLockableEntityCopy implements java.io.Serializable {
    public abstract javax.ejb.EJBObject findEntity() throws javax.ejb.FinderException,
java.rmi.RemoteException;
    public abstract javax.ejb.EJBObject findEntityForUpdate()
            throws javax.ejb.FinderException, java.rmi.RemoteException;
    protected abstract String getHomeName();
    public OptimisticLockableEntityCopy(java.sql.Timestamp argLastModifier) {
        lastModified = argLastModifier;
    }
    public java.sql.Timestamp getLastModified() {
        return lastModified;
    }
    protected java.sql.Timestamp lastModified;
```

```
    protected javax.ejb.EJBHome getHome() throws java.rmi.RemoteException {
        String homeName = getHomeName();
        // HomeFactory class is an implementation of the Factory for EJB Homes pattern published
        // in the IBM Redbook "Design and Implement Servlets/JSPs/EJBs for IBM WebSphere."
        // HomeFactory narrows the remote reference to the appropriate type.
        javax.ejb.EJBHome home = itso.dg.ejb.base.HomeFactory.getSingleton().lookupHome(homeName);
        if (home == null)
            throw new java.rmi.RemoteException("home with name '" + homeName + "' not found or
narrowing error");
        else
            return home;
    }
}
```

The OptimisticLockableEntityCopy base class has lastModified as a protected member. Each concrete copy class has to implement the three methods findEntity(), findEntityForUpdate(), and getHomeName().

The getHome() method of base class delegates the naming service (home lookup and narrowing the remote reference to the appropriate type) to the Home Factory. For a detailed discussion, see section 10.1, "Factory for EJB Homes" in the IBM Redbook Design and Implement Servlets/JSPs/EJBs for IBM WebSphere.

OptimisticLockService Methods

The OptimisticLockService (session facade for copying and updating all optimistic lockable entities) has two methods:

- copy entity
- update entity

For both methods, we set the transaction attribute to TX_REQUIRES_NEW. This transaction setting forces both methods to run in an own transaction context with a short commit cycle and lock time.

Example 3 shows the implementation of the copyEntity method:

Example 3. CopyEntity method of OptimisticLockService Session Bean

```
public OptimisticLockableEntityCopy copyEntity(OptimisticLockableEntityCopy
copyWithEntitySearchArgs)
            throws javax.ejb.FinderException, java.rmi.RemoteException {
    //find entity (read-only)
    EJBObject ejbObject = copyWithEntitySearchArgs.findEntity();
    if (ejbObject instanceof OptimisticLockable) {
        return ((OptimisticLockable) javax.rmi.PortableRemoteObject.narrow(ejbObject,
OptimisticLockable.class)).copy();
    }
    else
        throw new java.rmi.RemoteException("entity bean is not OptimisticLockable");
}
```

The caller of the copyEntity method instantiates a copy of an entity and sets the primary key fields for the copy request. The copy object with the search arguments is passed as a parameter to the copy method. The OptimisticLockService facade delegates the entity search to the findEntity method of the copy class, which invokes the entity's home findByPrimaryKey. A findByPrimaryKey does not exclusively lock the row in the underlying database.

The OptimisticLockService redirects the copy request to the found entity bean. The copy method of the entity implementation instantiates and returns a full-blown copy to the facade, which returns the serialized copy to the caller. The copy transaction commits.

Copy set to read-only: the copy method of the entity has to be set to read-only. To prevent a store at commit (which would update the timestamp via database trigger), we select the **Const method** check box for the copy() method in the Properties panel of the bean (VisualAge for Java). For a detailed discussion, see section 11.1.1, "Read-only methods" in the Redbook.

Example 4 shows the implementation of the updateEntity method:

Example 4. UpdateEntity method of OptimisticLockService Session Bean

```
public void updateEntity(OptimisticLockableEntityCopy entityCopy)
        throws OptimisticLockException, java.rmi.RemoteException {
    EJBObject ejbObject = null;
    try {
        //find entity (read-for-update)
        ejbObject = entityCopy.findEntityForUpdate();
    }
    catch(javax.ejb.FinderException finderExcept) {
```

```
        throw new OptimisticLockException();
    }
    if (ejbObject instanceof OptimisticLockable)
        ((OptimisticLockable) javax.rmi.PortableRemoteObject.narrow(ejbObject,
            OptimisticLockable.class)).update(entityCopy);        }
    else
        throw new java.rmi.RemoteException("entity bean is not OptimisticLockable");
}
```

The caller invokes the updateEntity method to synchronize the (updated) entity copy with the server-side entity state. The OptimisticLockService facade delegates the entity search to the findEntityForUpdate method of the copy class, which invokes the entity's home custom finder, passing the primary key fields and the timestamp from the entity copy as parameters.

If findEntityForUpdate() does not throw a FinderException, it means that no other transaction has changed the entity state in the meantime. The OptimisticLockService redirects the update request to the found entity bean. The update method of the entity implementation synchronizes the entity state with the copy. The OptimisticLockService then implicitly commits the update transaction and the database trigger updates the timestamp

If the findEntityForUpdate() throws a FinderException, it means that the EJB does not exist anymore or that another transaction has changed the entity state reflected by the timestamp. In this case, the OptimisticLockService throws an OptimisticLockException and the transaction rolls back.

Update not set to read-only: The update method of the entity does not have to be set to read-only. To enforce a store at commit (which updates the timestamp via database trigger) do not select the **Const method** check box for the update(OptimisticLockableEntityCopy) method in the Properties panel of the bean in VisualAge for Java. For a detailed discussion, see section 11.1.1, "Read-only methods" in the IBM Redbook Design and Implement Servlets/JSPs/EJBs for IBM WebSphere. To prevent database deadlocks, implement the custom finder with a SELECT FOR UPDATE, which acquires an exclusive lock on the corresponding row. For a detailed discussion, see section 11.1.3, "Database deadlocks" in the Redbook.

Sample code for an optimistic lockable Customer entity bean

This section shows the source for an optimistic lockable Customer entity bean. Customer has three persistent fields: customerId, name, and lastModified (java.sql.Timestamp).

Example 5 shows the copy method implementation of CustomerBean. The copy method creates a copy of the customer state:

Example 5. Copy method implementation of CustomerBean
```
public OptimisticLockableEntityCopy copy() {
    OptimisticLockableCustomerCopy copy = new OptimisticLockableCustomerCopy(customerId, name,
lastModified);
    return copy;
}
```

Example 6 shows the update method implementation of CustomerBean. The update method synchronizes the bean state with the customer copy:

Example 6. Update method implementation of CustomerBean
```
public void update(OptimisticLockableEntityCopy entityCopy) {
    OptimisticLockableCustomerCopy copy = (OptimisticLockableCustomerCopy) entityCopy;
    //update entity fields
    setName(copy.getName());
}
```

Example 7 shows the creation of the two database triggers (before insert / before update):

Example 7. Database triggers for timestamp setting
```
db2 create trigger cust_ins_timestamp no cascade before insert on ITSO.CUSTOMER
        referencing new as CUST for each row mode db2sql
    set CUST.LASTMODIFIED = CURRENT TIMESTAMP
db2 create trigger cust_upd_timestamp no cascade before update on ITSO.CUSTOMER
        referencing new as CUST for each row mode db2sql
    set CUST.LASTMODIFIED = CURRENT TIMESTAMP
```

Example 8 shows the source for the read-for-update custom finder (finder helper interface and class):

Example 8. Read-for-update custom finder implementation
```
public interface CustomerBeanFinderHelper {
    public java.sql.PreparedStatement
    findByPrimaryKeyForUpdate(CustomerKey customerKey, java.sql.Timestamp timestamp) throws
Exception;
```

```
}

public class CustomerBeanFinderObject extends com.ibm.vap.finders.VapEJSJDBCFinderObject
        implements CustomerBeanFinderHelper {
    public CustomerBeanFinderObject() {
        super();
    }
    public java.sql.PreparedStatement findByPrimaryKeyForUpdate(CustomerKey customerKey,
java.sql.Timestamp timestamp)
            throws Exception {
        java.sql.PreparedStatement ps = null;
        ps = getMergedPreparedStatement("T1.customerId = ? and T1.lastmodified = ? FOR UPDATE");
        ps.setString(1, customerKey.customerId);
        ps.setObject(2, timestamp);
        return ps;
    }
}
```

Example 9 shows the implementation of the Customer copy class. The constructor OptimisticLockableCustomerCopy(String argCustomerId) specifies the primary key field (customerId) of the customer to copy (passed as parameter in the copyEntity method of the OptimisticLockService facade):

Example 9. Implementation of Customer copy class

```
public class OptimisticLockableCustomerCopy extends OptimisticLockableEntityCopy {
    private String customerId;
    private String name;
    public OptimisticLockableCustomerCopy(String argCustomerId) {
        super(0);
        customerId = argCustomerId;
    }
    public OptimisticLockableCustomerCopy(String argCustomerId, String argName, java.sql.Timestamp
argTimestamp) {
        super(argTimestamp);
        customerId = argCustomerId;
        name = argName;
    }
    public javax.ejb.EJBObject findEntity() throws java.rmi.RemoteException,
javax.ejb.FinderException {
        //get narrowed home reference from base class
        CustomerHome customerHome = (CustomerHome) getHome();
        return customerHome.findByPrimaryKey(new CustomerKey(customerId));
    }
    public javax.ejb.EJBObject findEntityForUpdate() throws java.rmi.RemoteException,
javax.ejb.FinderException {
        //get narrowed home reference from base class
        CustomerHome customerHome = (CustomerHome) getHome();
        return customerHome.findByPrimaryKeyForUpdate(new CustomerKey(getCustomerId()),
getTimestamp());
    }
    public String getCustomerId() {
        return customerId;
    }
    public String getName() {
        return name;
    }
    protected String getHomeName() {
        return "Customer";
    }
}
```

Conclusion

The proposed Optimistic Locking Pattern is intended for application servers that do not have optimistic locking support for EJB Entity Beans. It ensures data integrity with short commit cycles and improves system throughput and performance. The Optimistic Locking Pattern is part of the IBM Redbook Design and Implement Servlets/JSPs/EJBs for IBM WebSphere.

Related Redbooks

Design and Implement Servlets/JSPs/EJBs for IBM WebSphere, (SG24-5754) by Joaquin Picon, Regis Coqueret, Andreas Hutfless,

Gopal Indurkhya, Martin Weiss

[Enterprise JavaBeans Development Using VisualAge for Java](#), (SG24-5429) by Joaquin Picon, Patrizia Genchi, Maneesh Sahu, Martin Weiss, Alain Dessureault. This Redbook is based on WebSphere Advanced Edition 2.0 and VisualAge for Java, Enterprise Edition 2.0

[WebSphere Studio and VisualAge for Java Servlet and JSP Programming](#) (SG24-5755) by Ueli Wahli, Mitch Fielding, Gareth Mackown, Deborah Shaddon

Download

File Description	Type	Size	Method
Patterns sample code	zip	.2 MB	[FTP](#) or [HTTP](#) or [Download Director](#)

Related Information
⇢ Which download [Method](#) should I choose?
⇢ [Download FAQs](#)

About Martin Weiss

Martin Weiss of IBM Global Services in Switzerland has been with IBM since 1978, and has worked in application development in the OS/390, AS/400, OS/2, and Windows NT environments. Since 1993, he has focused on object technology as a developer and mentor on VisualAge customer projects, including Smalltalk, C++, and Java. Martin has developed Java enterprise applications for WebSphere Application Servers, and is the co-author of two IBM Redbooks:

- Design and Implement Servlets/JSPs/EJBs for IBM WebSphere (SG24-5754)
- Enterprise JavaBeans Development Using VisualAge for Java (SG24-5429)

You can reach Martin at wmt@ch.ibm.com.

String Externalization with VisualAge for Java (Part 2)

Joe Winchester, IBM
May 2000

Introduction

In String Externalization with VisualAge for Java (Part 1), I described how to externalize user-visible strings from application code. The strings are held separately from the class that references them in Java resource bundles that can exist either as classes or as properties files. The bundles are located at run time by combining the resource bundle name with the current locale. By separating the definition of the strings from their use, and having their location indirected via the current locale, you can write an application that can be easily translated. Translators can create different bundles for each locale using the keys defined in the base bundle, and at run time the application will locate the correct set for the user's machine.

Translations of the same word in different languages will most likely have different lengths. For example, in French, "red" is "rouge" and "green" is "vert." If the strings are used as part of a label in a GUI, then the GUI must accommodate the variable lengths without clipping longer strings or having white space around shortened strings. To achieve this, the GUI must calculate dimensions at run time based on the contents of its labels. This article shows how to use layout managers to write a GUI that is sensitive to the size of its externalized strings.

Resource bundles in the Visual Composer Editor

Part 1 of this article showed how to reference a string that came from a resource bundle. A hard-coded string reference such as System.out.println("red"); can be replaced by a resource bundle lookup such as:

```
ResourceBundle colorBundle =
    ResourceBundle.getBundle("com.app.bundles.ColorResourceBundle");
String red = helloWorldBundle.getString("red");
System.out.println(red);
```

The available bundles for ColorResourceBundle are looked up based on the current locale and used to create a hierarchy of bundles. The hierarchy of bundles is searched for the key "red", which is either returned or else a MissingResourceBundle exception is thrown.

In a GUI built with the Visual Composition Editor (VCE), user-visible strings for controls such as buttons are set using the property sheet. Figure 1 shows a frame with two buttons that have labels of "red" and "green." The **red** button is selected and the Property Editor shows its text property set to the literal string "red":

Figure 1. The red button has its label set to the string red
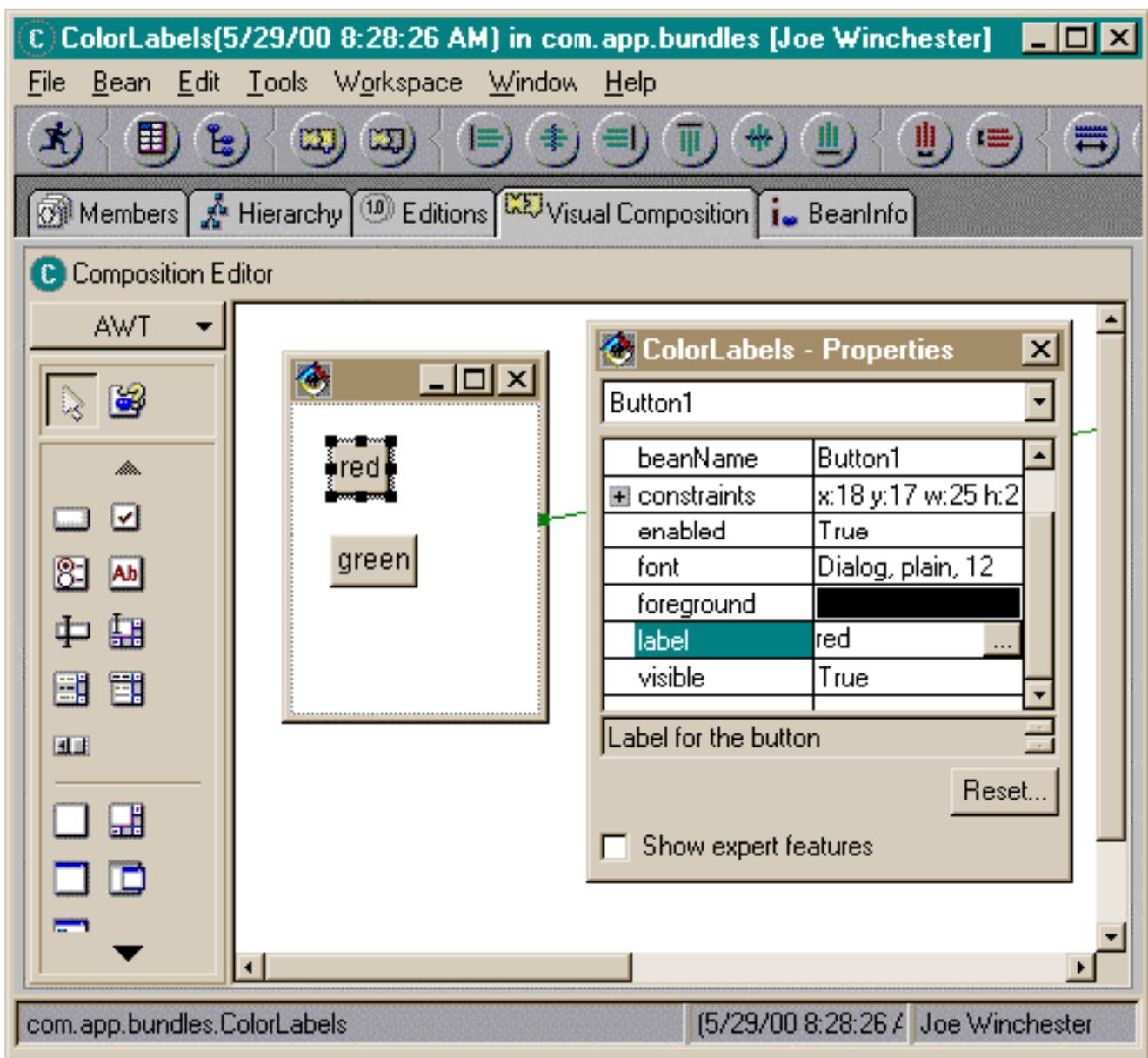
The generated code that initializes the property values for the button is shown below:

```
/**
 * Return the Button1 property value.
 * @return java.awt.Button
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private java.awt.Button getButton1() {
    if (ivjButton1 == null) {
        try {
            ivjButton1 = new java.awt.Button();
            ivjButton1.setName("Button1");
            ivjButton1.setBounds(18, 17, 25, 23);
            ivjButton1.setLabel("red");
            // user code begin {1}
            // user code end
        } catch (java.lang.Throwable ivjExc) {
            // user code begin {2}
            // user code end
            handleException(ivjExc);
        }
```

```
        }
        return ivjButton1;
    }
```

The value "red" is hard coded as the parameter of the setLabel(String) method on the ivjButton1 field.

VisualAge for Java does not let you change generated code outside of user code blocks (as delimited by // user code begin and // user code end comments). If the value "red" were replaced with a lookup to a resource bundle key, it would be overwritten the next time the code was regenerated.

There are two ways to teach the VCE to use a resource bundle lookup for a string:

1. From the Property Editor itself

Figure 1 shows the Property Editor for the label property of the button with the value "red" in the Property Editor text box, but with a button labeled **...** beside it. Selecting this button will open an additional dialog, as shown in Figure 2:

Figure 2. The additional prompter for a String property editor
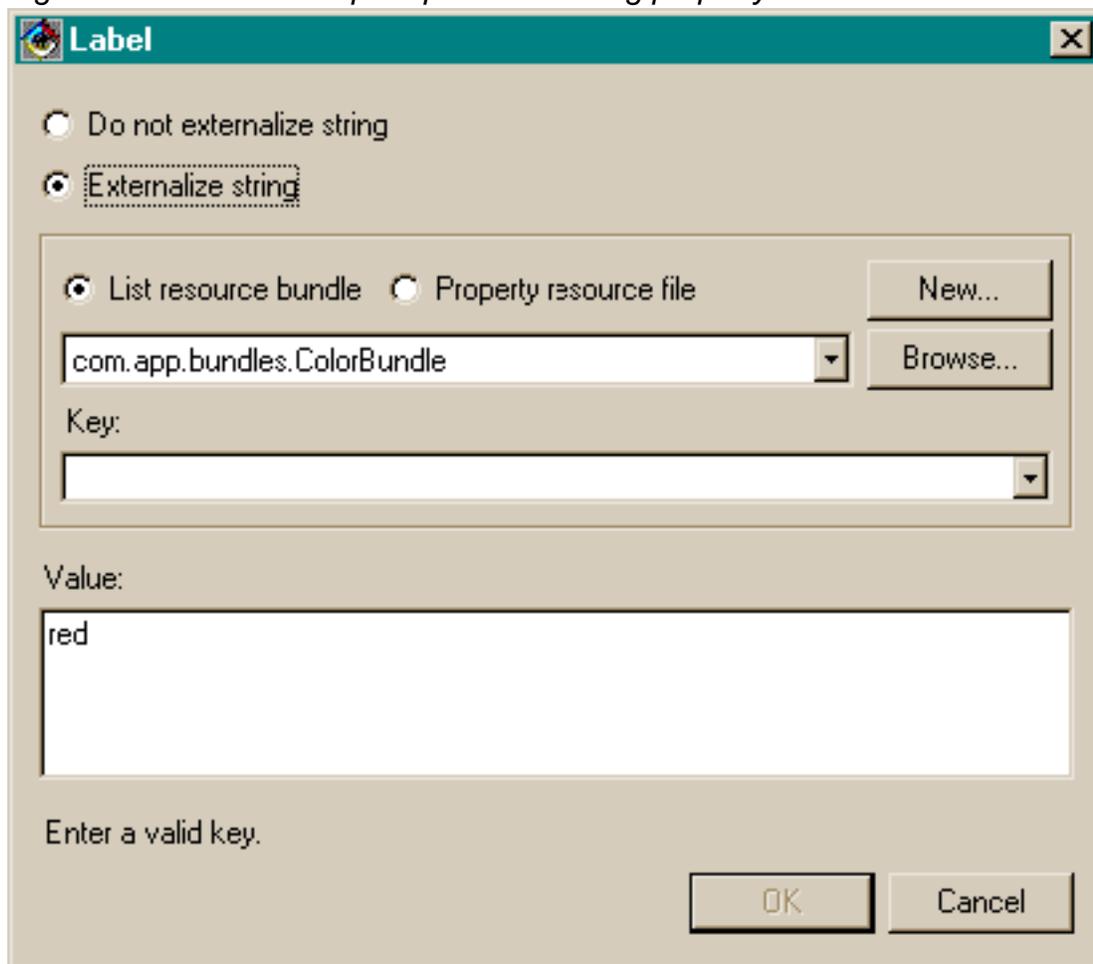
The prompter shown in Figure 2 is similar to the externalize string wizard that was covered in Part 1 of this article. From the prompter, an existing bundle can be selected or a new one created. Figure 2 shows a bundle com.app.bundles.ColorBundle as having been created by pressing the New button.

One of the disadvantages of the Externalize String dialog that lets you externalize strings for an entire class was that you couldn't see the list of available keys inside the bundle. The VCE string externalization prompter, however, lets you see existing keys in its drop-down combo box. Because the bundle is a new one, there are no existing keys and the desired key must be entered. You can enter "red" in the combo box and click OK.

The Property Editor still shows the value "red" as shown in Figure 1. The generated code, however, is different, as shown below:
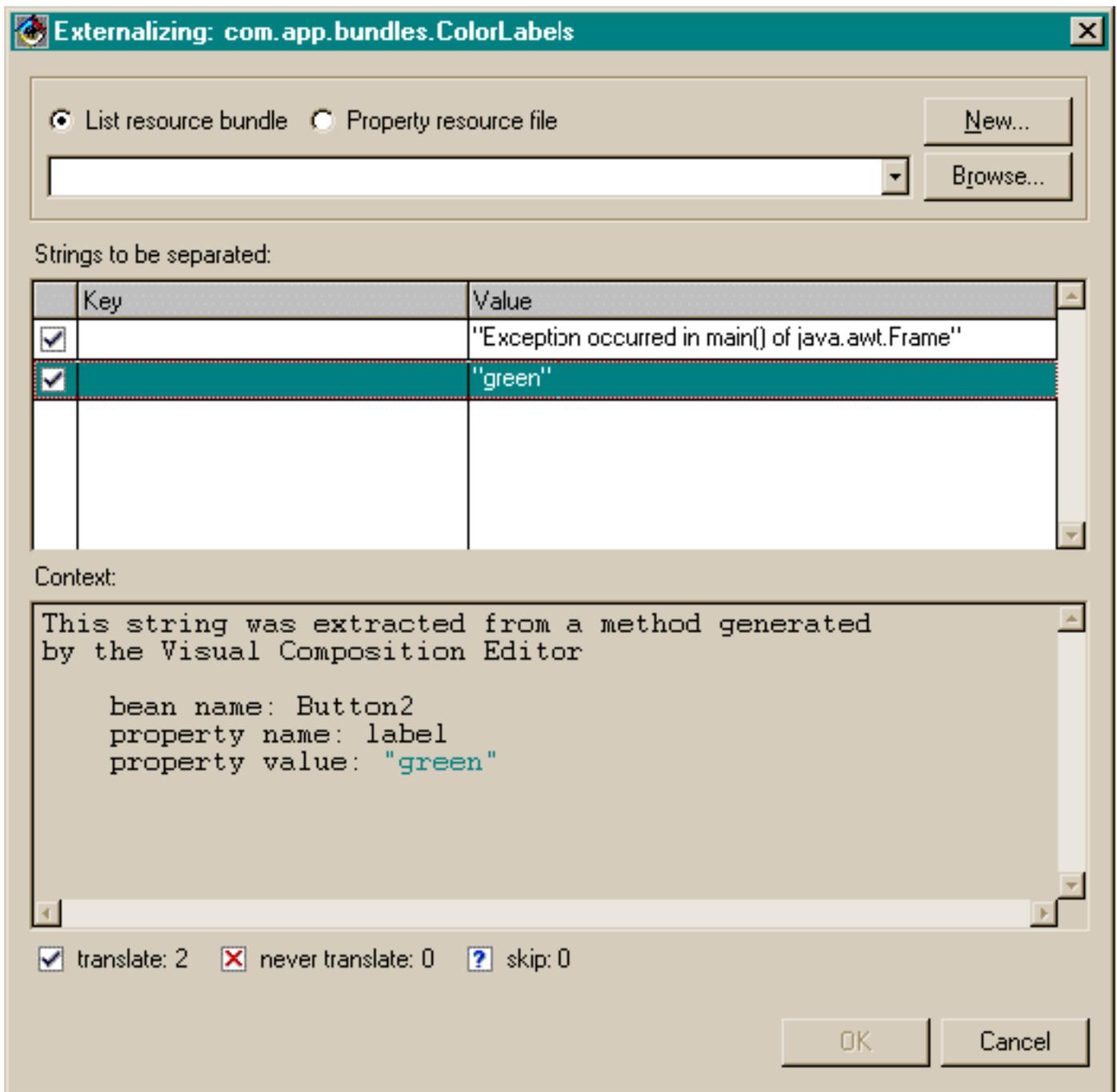
```
/**
 * Return the Button1 property value.
 * @return java.awt.Button
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private java.awt.Button getButton1() {
    if (ivjButton1 == null) {
        try {
            ivjButton1 = new java.awt.Button();
            ivjButton1.setName("Button1");
            ivjButton1.setBounds(18, 17, 25, 23);
            ivjButton1.setLabel(resColorBundle.getString("red"));
            // user code begin {1}
            // user code end
        } catch (java.lang.Throwable ivjExc) {
            // user code begin {2}
            // user code end
            handleException(ivjExc);
        }
    }
    return ivjButton1;
}
```

The value "red" used as the parameter to the setLabel button has been replaced with a lookup to the resource bundle.

2. With the String Externalization dialog

The second way to externalize a string that is a property value of a VCE bean is with the String Externalization dialog available from the type's pop-up menu in one of the browsers. This is the method used to externalize all strings in a class that was covered in Part 1 of this article. VCE generated strings are treated specially, however, as shown in Figure 3:

Figure 3. VCE strings are treated specially by the String Externalization wizard

Externalizing: com.app.bundles.ColorLabels

- ⦿ List resource bundle ○ Property resource file

New...
Browse...

Strings to be separated:

	Key	Value
☑		"Exception occurred in main() of java.awt.Frame"
☑		"green"

Context:

```
This string was extracted from a method generated
by the Visual Composition Editor

     bean name: Button2
     property name: label
     property value: "green"
```

☑ translate: 2   ☒ never translate: 0   ❓ skip: 0

OK   Cancel

The first thing to notice about the String Externalization wizard is that it knew to ignore the string "red" that was previously translated with the Property Editor string externalization prompter. Usually the String Externalization wizard requires that a comment //NON-NLS-$n be present at the end of a Java statement, where n is the number of the string. The value "red" used as the resource bundle lookup key has no such comment, but is not shown in the list of eligible strings. The reason is that the externalization wizard has access to the internal data used by the VCE and knows that the string "red" has been translated by the Property Editor and therefore need not be externalized again.

The value "green," however, was not externalized by the Property Editor and does appear as being eligible for translation. Rather than showing the method in the Context text box, the Externalizer shows the name of the bean, the property name, and the property value. If the value is selected to be translated, it is not only changed in the source but is also changed inside the VCE internal data so that the Property Editor will show the value "green" as having been translated.

Sizing the GUI based on the external strings

The example used for Figure 1 is a simple frame that has two buttons dropped into its content pane. The default layout for the content pane is <none>, which means that the Layout Manager is set to null. This gives absolute position and sizing of components. The size and position of the red and green buttons are therefore hard-coded into the generated code, as you can see in the method above, where the setBounds(Rectangle) method is called with the exact x, y, width, and height.

The advantage of having no layout manager is that you can easily set the position and size of the components and guarantee that these are the same coordinates used at run time. This is also the disadvantage, because in a deployment environment where the strings are different lengths, the application will clip strings that are too long and show white space around strings that have become shorter.

To see this, a resource bundle for the two strings red and green was created for the locale fr. The class definition is shown below:

```
public class ColorBundle_fr extends java.util.ListResourceBundle {
    static final Object[][] contents = {
        {"red", "rouge"},
        {"green", "vert"},
    };
}
```

When the class is run from the scrapbook with the locale set to French, the effect is shown in Figure 4:

Figure 4. The absolute sizing of the components has clipped the label rouge and left too much spacing around the string vert
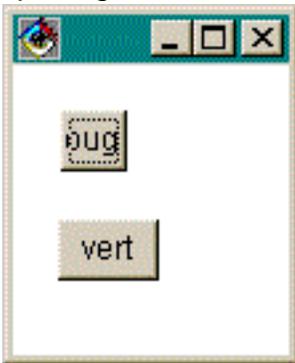
To avoid this effect, a Layout Manager needs to be used that will size the controls based on the size of their initial contents. There are a number of such layout managers, but two of the most commonly used ones are Flow and GridBag.

Flow layout implements the interface LayoutManager but not the interface LayoutManager2. What this means is that components are added to a container with the add(Component) method without an additional constraint. When you select flow layout from the list of available layout managers in the Property Editor for the panel, the constraints property for each of the buttons will be removed. Figure 5 shows the VCE with the layout changed from to FlowLayout:

Figure 5. The panel's layout manager changed from <null> to FlowLayout
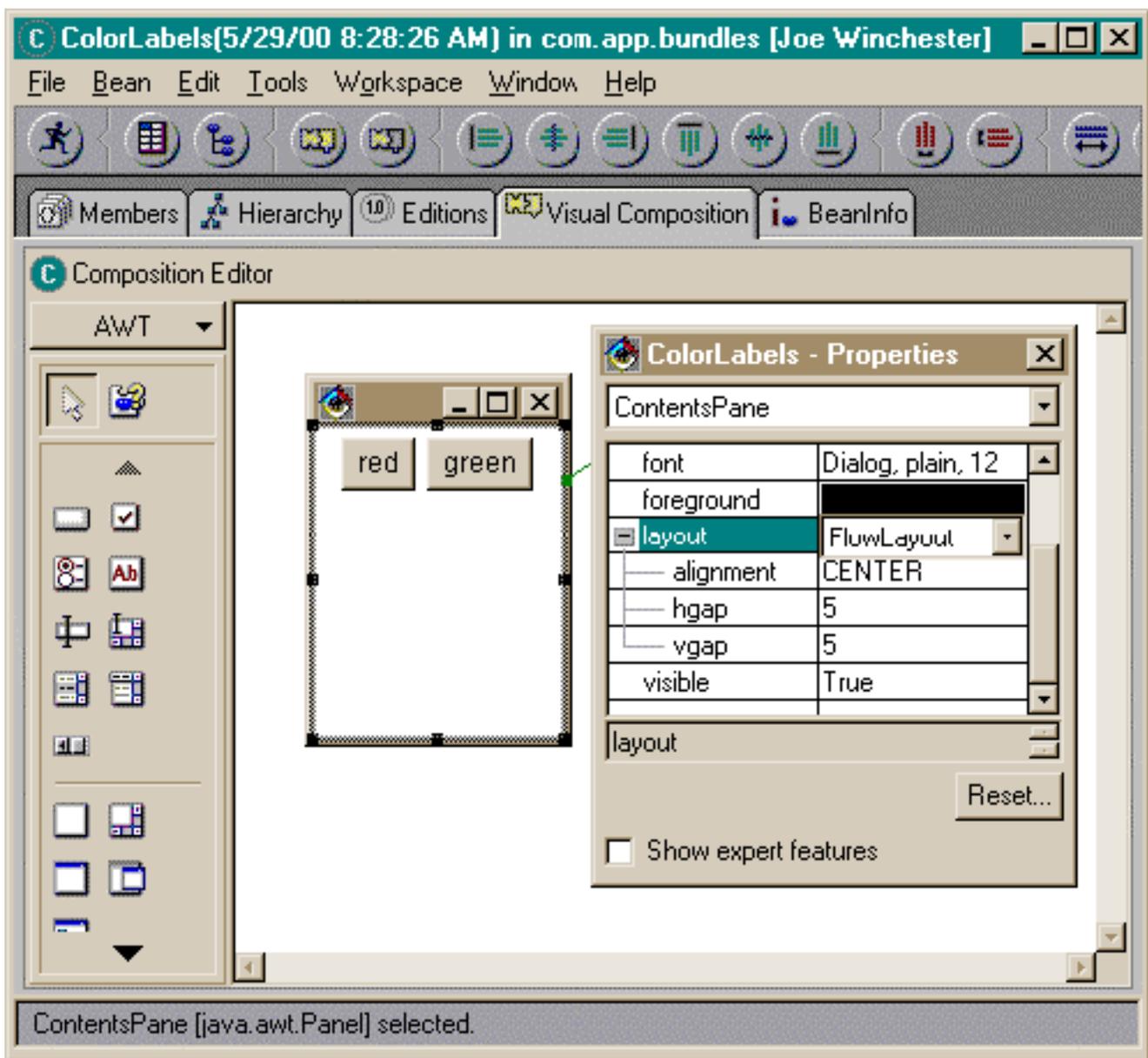
With the flow Layout Manager, you can exert very little control over the size and position of the buttons. Each component has a preferred size that is returned by its getPreferredSize() method, and the layout manager will use this to position each control. Controls can be re-ordered by dragging them in the composition surface (or in the beans list), but their position is always calculated by the Layout Manager. Some fine tuning can be set with the Layout Manager's properties such as the number of pixels to use for the vertical and horizontal gap between components, as well as how to position controls that overflow onto the last line.

The advantage that flow layout has is that because the size of each component is determined at run time by asking it for its getPreferredSize(), the controls will grow or shrink as required by their externalized string values. The effect of this is shown in Figure 6, where the ColorLabels class is run with the locale set to use the French strings:

Figure 6. Flow layout will always use the preferred size for its components

Flow layout ensures that labels grow and shrink as required, but it offers you little control as to how to position the controls. To do this, use the Layout Manager GridBag.

GridBag implements the interface LayoutManger2, which means that when a component is added to its parent container, a constraint is specified. The constraint is an instance of the class GridBagConstraints, and it lets you specify precisely where to position the control and how to calculate its size as its parent container resizes.

The subtleties of GridBag are beyond the scope of this article, but if you want more information on using GridBag Layout, see [Methods for Using the GridBag Layout Manager](#).

The VCE has some nice features that make GridBag easier to use and understand than if you were having to hand write a VCE from scratch. The first of these is that if you switch layout managers, the constraints for the new Layout Manager are calculated from the positions of the existing controls. For example, with the two buttons set as in Figure 5, I can change the layout of the panel from Flow to GridBag. The constraint objects for each of the controls will be calculated as closely as possible to preserve the existing components' positions. This is shown in Figure 7 with the constraints property for the red button selected:

Figure 7. The constraints for the button have been automatically calculated
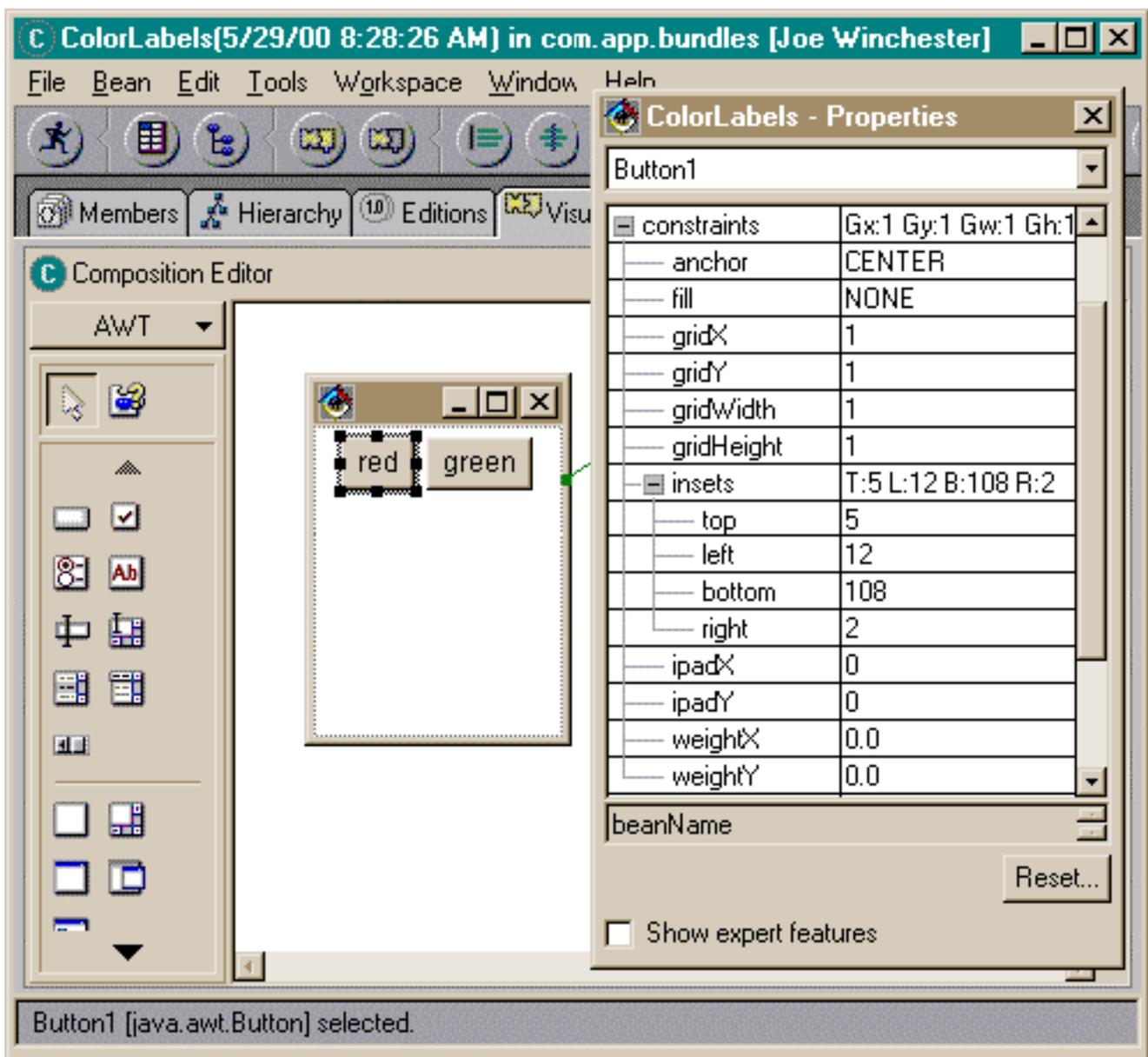
The VCE has created two grid columns, one for the red button and one for the green button. It has created inserts to position the two buttons within the two grid columns.

The second nice feature of the VCE that helps you use GridBag is the ability to see the existing grid columns visually. To do this in the VCE, you cannot statically switch them on, but you can see them when you drop a new control. To see this, select a new control such as a button from the palette and begin to drop it on the panel. Press and hold the left mouse button to begin dropping the control. The drop target emphasis feedback will show you the position of the existing grids as well as the current drop target, as shown in Figure 8:

Figure 8. When dropping into a grid bag layout the target feedback shows the existing columns
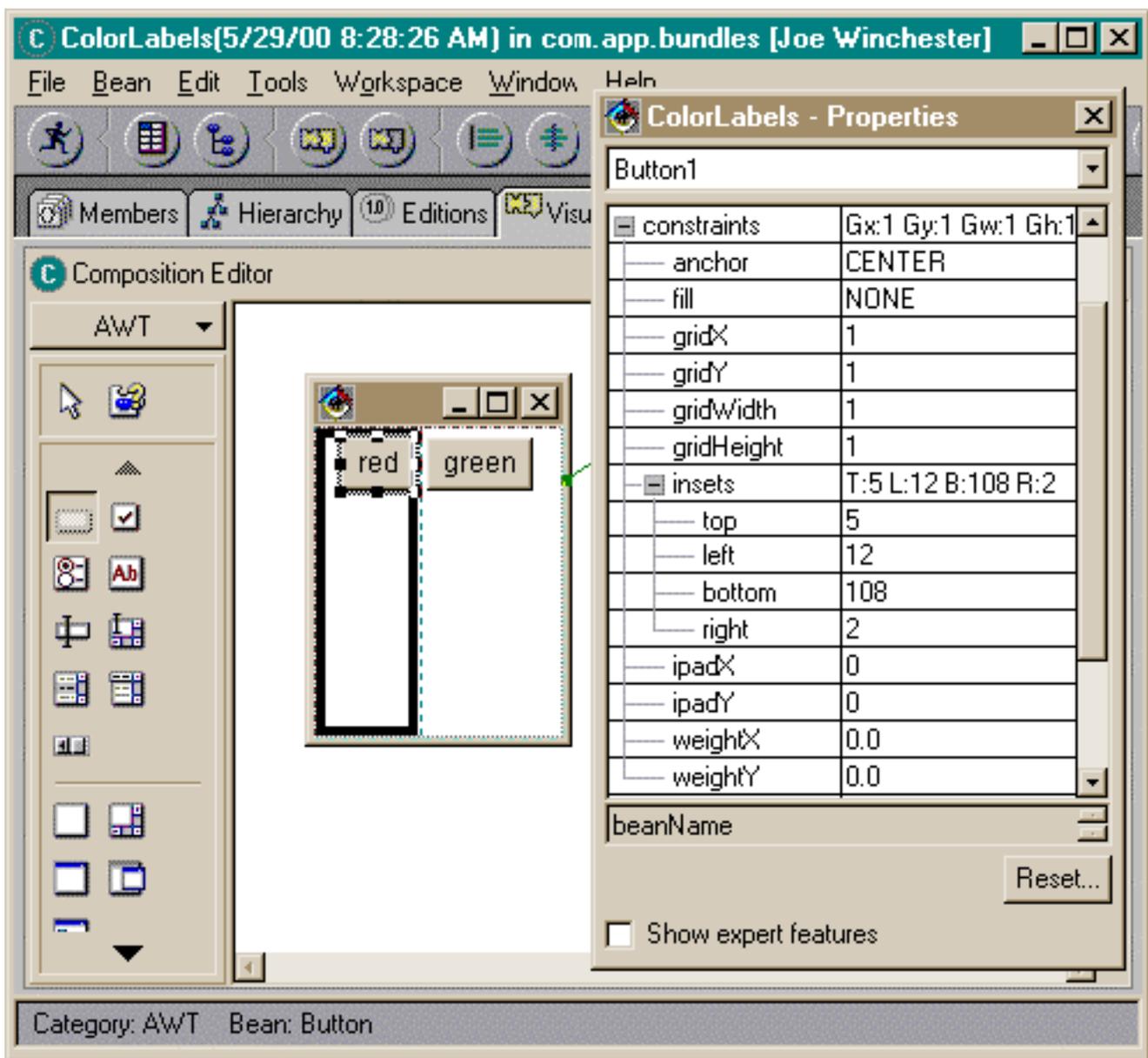
When the drop cursor is over a column, it is shown with a dark rectangle, and the other columns are shown with dotted rectangles. If the cursor is near the border of two columns, the border is emphasized to indicate that the drop operation will create a new grid column (or row) when the component is dropped.

As mentioned before, the full power of GridBag is beyond the scope of this article, but if you want to write a GUI that will respect the preferred size of its components and allow precise design-time positioning of them, then it is the king of layout managers. One effective technique is to lay out the controls with a layout manager of that gives you precise absolute positioning, and then, just before you generate the code, switch the layout manager to GridBag. The VCE will calculate a new set of GridBagConstraints for each component and you have the best of both worlds: a design-time mode with precise developer positioning of controls, and a run-time class with controls that will grow or shrink to fit their external strings.

Conclusion

of this article introduced the need for separating user-visible strings from application code, and it showed you how to achieve this with Java resource bundles. The String Externalization wizard available from a type's menu in Visual Age for Java's browsers will analyze a class's source and lead you through both generating resource bundles as well as consuming existing ones. This article has shown how to generate and consume externalized strings from within the VCE Property Editor for a bean. It has also described how a container's layout manager affects the positioning and sizing of its components. Other issues to consider

when writing an application for multiple locales include different formats for monetary amounts and dates, right-to-left typing, and non-ASCII character sets such as Kanjii. I hope you've found this article useful, and I welcome feedback (joewin@us.ibm.com).

About Joe Winchester

Joe Winchester is a member of the VisualAge Development team working for the Software Solutions group in Research Triangle Park, NC. He has worked on the development of two VisualAge persistence features: Object Extender for Smalltalk and Persistence Builder for Java.

E-mail: joewin@us.ibm.com.