
Oracle Business Components for Java

An Oracle Technical White Paper

Contents

- [Introduction](#)
 - [Overview of Oracle Business Components for Java](#)
 - [Ten Key Benefits of Oracle Business Components for Application Developers](#)
 1. [Improves Time-to-Market by Simplifying Business Logic Coding](#)
 2. [Increases Flexibility and Reuse with Updateable, SQL-Based Views](#)
 3. [Reduces Complexity by Coordinating Multiple Views of Data](#)
 4. [Simplifies Deployment by Making JSP- and Swing-based Clients Easy](#)
 5. [Saves Time and Avoids Bugs by Eliminating Hand-Written Database Code](#)
 6. [Enables Dynamic Applications through Active Metadata](#)
 7. [Improves Productivity through Tight Integration with Oracle JDeveloper](#)
 8. [Boosts Performance with an Efficient, Coarse-Grained Service Architecture](#)
 9. [Permits Logical 3-Tier Applications to Target Multiple Physical Deployments](#)
 10. [Supports Robust, Layered Customizations of Component Code and Metadata](#)
 - [Summary](#)
-

Introduction

The Enterprise JavaBeans™ (EJB) technology defines a server-side model for development and deployment of distributed Java components. The EJB specification defines a standard set of APIs and the runtime services that are required to be supported by an EJB Server. This model enables an application developer to focus on application logic and lets the EJB Server focus on providing the services such as transaction, security, and resource management. However, the task of writing the robust functionality needed for real-world business applications is still the responsibility of the enterprise bean developer. In particular, the EJB specification does not specify how to do the following tasks:

write the business application logic,

reuse business logic in multiple applications,

access views of data efficiently from HTML and Java user interfaces,

customize the business functionality once the application is delivered.

In practice, *these* are the activities where application developers and solution providers spend the bulk of their time and effort. **Oracle Business Components for Java** was designed to address the application

developer's needs for these challenging tasks.

Supporting standard EJB and CORBA deployment architectures, **Oracle Business Components for Java** dramatically simplifies the development, delivery, and customization of Java business applications for the enterprise. Oracle Business Components is an application component framework providing developers a set of intelligent software building-blocks that cooperate "out-of-the-box" to manage all of the common facilities required to:

- Productively author and test business logic in components which automatically integrate with databases,

- Flexibly reuse business logic through multiple SQL-based views of data, supporting different application tasks,

- Efficiently access and update the views from Servlets, JavaServer Pages (JSPs), and Thin-Java Swing clients

- Easily customize application functionality in layers without requiring modification of the delivered application.

By eliminating the substantial coding and testing work related to common "application plumbing" facilities, Oracle Business Components lets application developers focus full-time on implementing business solutions. The benefits of using this framework are immediate and obvious: reduced development cost, lower project risk, and shorter time-to-market.

It's important to note that while the framework handles all the common cases with its sophisticated built-in behavior, taking advantage of these benefits does **not** compromise your ability to **have it your way** when necessary. Since any automatic behavior provided by the framework can be easily overridden in your domain-specific components with a few strategic lines of code, you're never locked in to a certain way of doing things for all cases.

Oracle Business Components for Java is the culmination of years of joint design and development work between the Oracle Applications, Oracle Tools, and Oracle Server Technologies development organizations to pragmatically implement Oracle's vision for how scalable, multi-tier enterprise Java applications will be built in the next millennium.

Overview of Oracle Business Components for Java

Oracle Business Components for Java is a 100%-Java, XML-powered framework that enables productive development, portable deployment, and flexible customization of multi-tier, database-savvy applications from reusable business components.

Application developers use the Oracle Business Components framework and Oracle JDeveloper's integrated design-time wizards, component editors, and productive Java coding environment to assemble and test application services from reusable business components. These application services can then be deployed as *either* CORBA Server Objects or EJB Session Beans on enterprise-scale server platforms supporting Java technology.

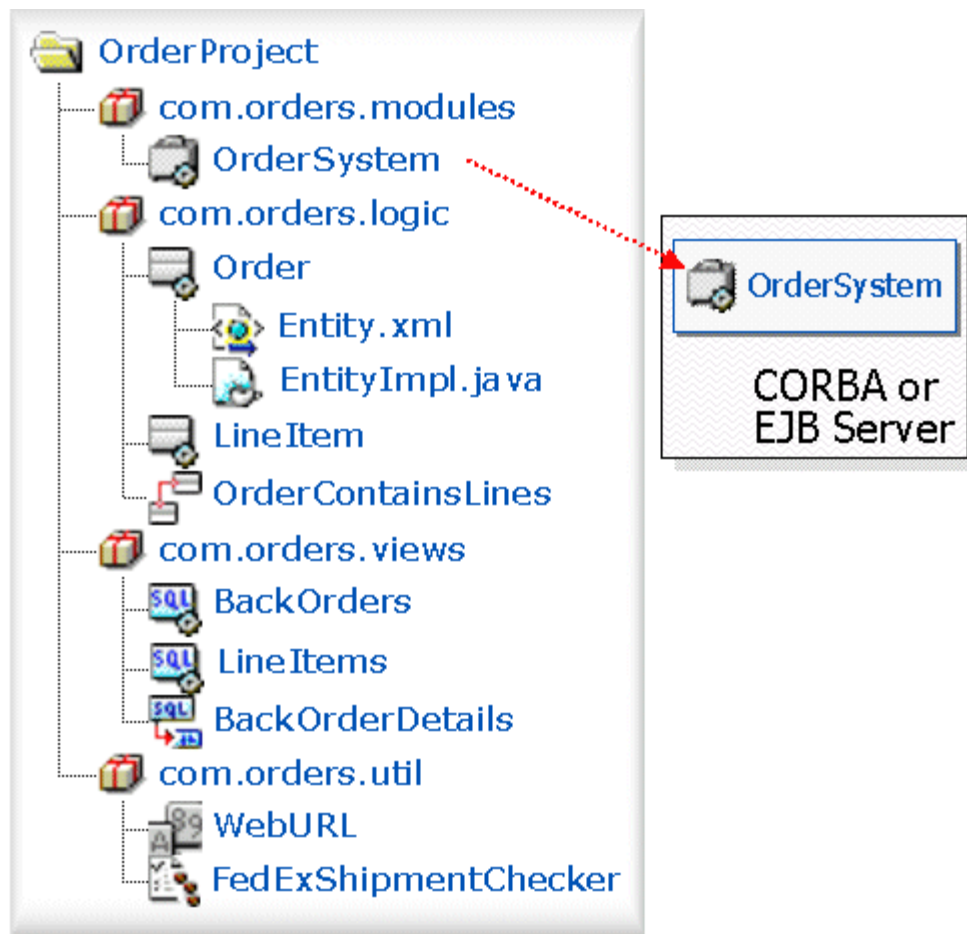


Figure 1: Design-time View of a Sample Project for an OrderSystem Application

Figure 1 shows a design-time view of a basic project: Application Module, Entity Object, Association, View Object, and View Links. Each of these components will be explained in this paper. The components are organized into packages, and each component (like the **Order** Entity Object expanded above) comprises an XML Component Definition and a Java Implementation Class, which cleanly separates metadata from domain-specific code to facilitate application customization and support dynamic application functionality.

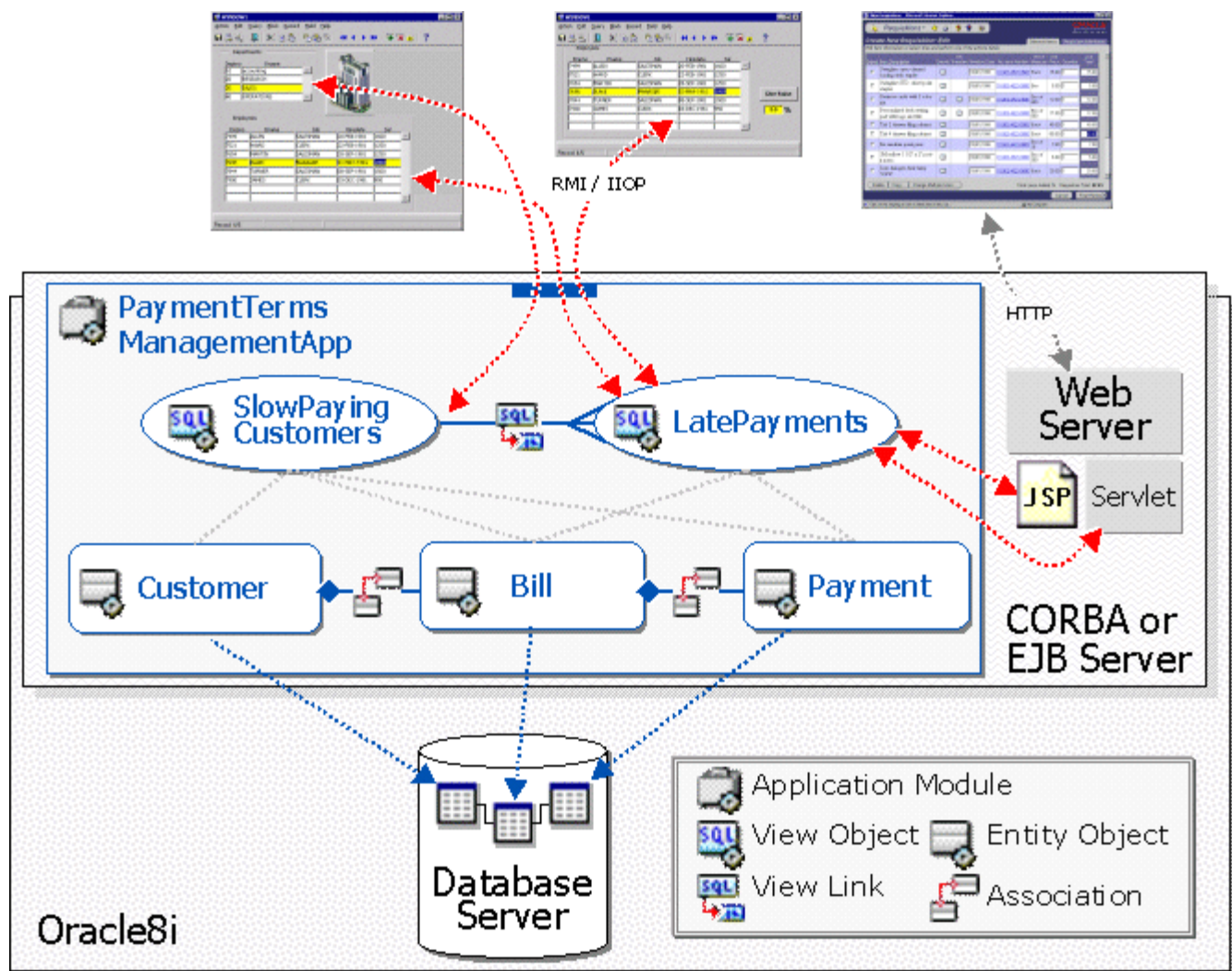


Figure 2: Runtime View of a Sample Application for Managing Customer Payment Terms

Figure 2 shows a runtime view of a simple Oracle Business Components application:

- **Entity Objects** like **Customer**, **Bill**, and **Payment** for encapsulating the business logic and database storage details of your business entities,
- **Associations** for relating and composing your business entities to capture all their relationships,
- **View Objects** like **SlowPayingCustomers** and **LatePayments** for joining, filtering, projecting, and sorting your business data for the specific needs of a given application scenario or task,
- **View Links** for declaratively coordinating hierarchies of master-detail queries to support sophisticated user-interfaces, and
- **Application Modules** like **PaymentTermsManagementApp** for assembling, packaging and deploying all the View Objects and business services required for a complete application.

Once an Application Module is deployed, the Oracle Business Components framework provides facilities for easily coupling HTML and Java user interfaces to server-side business components. In addition, new Application Modules can aggregate the functionality of existing Application Modules for true component assembly.

As *Figure 2* shows, **Oracle Business Components for Java** is not *itself* a CORBA server or an EJB server. Instead, Oracle Business Components is a framework which provides a development infrastructure that enables developers to productively *build* application components out of smart, reusable parts. These application components are then *deployed* as CORBA Server Objects or EJB Session Beans to a CORBA or EJB server that provides their execution environment.

Since *your* domain-specific Entity Objects, View Objects, and Application Modules extend the base building-blocks provided by the Oracle Business Components framework, they automatically inherit a comprehensive set of default runtime behavior. The built-in behavior, which you get by using the framework, makes you dramatically more productive at delivering enterprise Java applications. This paper describes in detail the ten key benefits that **Oracle Business Components for Java** provides developers to achieve this result.

Ten Key Benefits of Oracle Business Components for Application Developers

Oracle Business Components for Java delivers the following key benefits to developers building, deploying, and customizing enterprise CORBA/EJB applications:

1. Improves Time-to-Market by Simplifying Business Logic Coding
2. Increases Flexibility and Reuse with Updateable, SQL-Based Views
3. Reduces Complexity by Coordinating Multiple Views of Data
4. Simplifies Deployment by Making JSP- and Swing-based Clients Easy
5. Saves Time and Avoids Bugs by Eliminating Hand-Written Database Code
6. Enables Dynamic Applications through Active Metadata
7. Improves Productivity through Tight Integration with Oracle JDeveloper
8. Boosts Performance with an Efficient, Coarse-Grained Service Architecture
9. Permits Logical 3-Tier Applications to Target Multiple Physical Deployments
10. Supports Robust, Layered Customizations of Component Code and Metadata

The following sections provide technical highlights and a brief explanation of each of these major benefits above.

1. Improves Time-to-Market by Simplifying Business Logic Coding

Technical Highlights

- ▶ Implement sophisticated business rules easily with the Entity Object's multi-level validation support.
- ▶ Build "active" Associations to related or composed components.
- ▶ Define Domains and Validation Rules to reuse across components.

1.1 Encapsulate Business Logic with Entity Objects



Entity Object

Figure 3 illustrates a UML Model for the business entities in a simple E-Commerce Order System. To immediately put the drawing in action, Oracle Business Components provides a basic framework component called the Entity Object which handles all the details of encapsulating business logic and interacting with the database, for each UML entity in your business model.

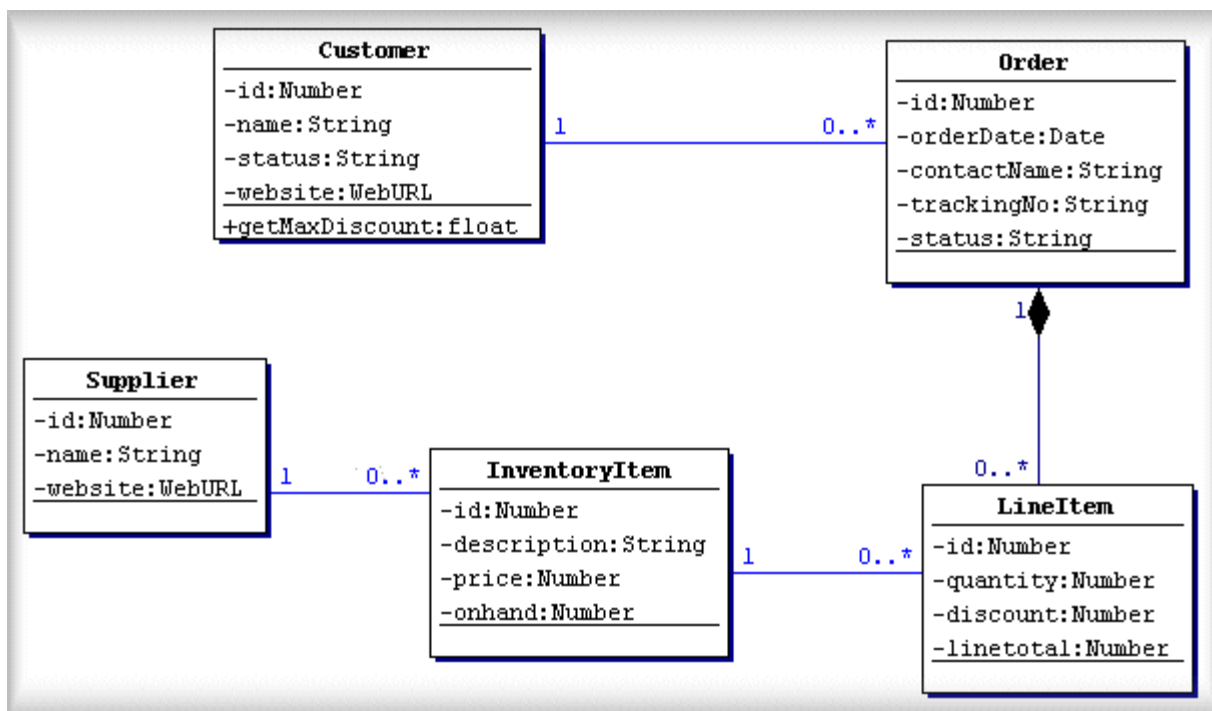


Figure 3: UML Entity Model for an E-Commerce Order System

With **Oracle Business Components for Java**, rather than coding business logic into each client application or servlet you write, you write it once in a reusable, domain-specific business component which extends the basic **Entity Object** in the framework. For example, an **Order** Entity Object implementing the **Order** Entity from *Figure 3* can encapsulate:

- The attribute definitions for Order, including datatype and database column mapping information for each.
- The attribute-level and Entity-level validation logic implementing business logic for an Order.
- Default value logic for attributes on a new Order such as a credit allowances, order date, and account representative.
- Any custom business methods or overridden framework methods required to correctly enforce business policies on Orders.

You write validation logic in your Entity Object's Java Implementation Class and the framework worries about coordinating when that validation logic is enforced. If an Entity Object is composed with other entities (as Order is with LineItem in *Figure 3*) the framework coordinates the validation of composed entities for you too.

1.2 Relate and Combine Components with Associations



Association

By creating an Association between two Entity Objects, you establish an active link between them. You get direct access to "walk" the links in your code through Association getter methods and direct access to assign related objects with Association setter methods. Since getter and setter methods are the coding pattern in Oracle Business Components for attribute access, in effect, the presence of Associations makes virtual attributes appear on your entities, allowing seamless access to related information.

```
j:\myprojects\demo\LineItemImpl.java

*/

public void setDiscount(Number value) {
    // =====
    // Business Rule
    // =====
    //
    // Discount cannot be greater than the Maximum Discount
    // allowed for the customer based on their customer status
    // (Note: Uses Association getXXX() methods to traverse)

    if ( value.floatValue() > getOrder().getCustomer().getMaxDiscount() )
        throw new oracle.jbo.JboException("Discount is too high!");

    setAttributeInternal(DISCOUNT, value);
}
```

Figure 4: Attribute-level Validation Code in LineItem Leveraging Association

Accessors to Traverse to the Customer Who Placed This Order and Call a Method on It

Figure 4 shows how easy it is to write Java code using Oracle Business Components to enforce a business rule whose implementation requires looking up the Order for the current LineItem, then looking up the Customer who placed that Order, and then invoking a business method on **Customer** to check a maximum allowed discount. By leveraging Oracle Business Components Association accessors, business logic authoring is greatly simplified.

Associations indicate that one Entity references another. In addition, identical to UML, Associations can be flagged as being "Compositions", which indicate a stronger ownership or containership relationship between two components. For example in Figure 3, **Order** and **Customer** are related by an Association, as are **LineItem** and **InventoryItem**. This simply means that an **Order** is placed by a *particular* customer and that a **LineItem** places an order for a certain quantity of a *particular* item in the inventory. If an **Order** is deleted from the system, there is no implied impact on the existing **Customer** who placed that order or to the **InventoryItems** which had been requested on the deleted order.

On the other hand, **Order** and **LineItem** above are related by a composition (indicated in UML with the solid diamond on the Association line). Using Oracle Business Components, at runtime this means that:

- An **Order** with valid **LineItems** cannot be deleted until all the contained **LineItems** are deleted,
- When an attempt to change a **LineItem** is made, it will be disallowed unless the owning parent **Order** can be successfully locked, and
- If a **LineItem** gets modified, the framework automatically coordinates the parent **Order**'s revalidation of its children. For example, an **Order** might have a business rule which checks to make sure the sum of all the **LineItems** on the order don't add up to more than the **Customer**'s net credit allowance.

So Associations and Compositions capture the web of relationships among your business entities and make the system smarter by knowing about them. As we've seen, using Association and Compositions in Oracle Business Components automates many common validation tasks for you.

1.3 Share Common Validations with Validation Rules and Domains



Validation Rule

Validation Rules are JavaBeans which encapsulate reusable validation code for use across any Entity Objects where they might be useful. In contrast to the Java code-based validation you write in your Entity Object's Java Implementation Class, you attach Validation Rules to your Entity Object declaratively through its XML Component Definition.

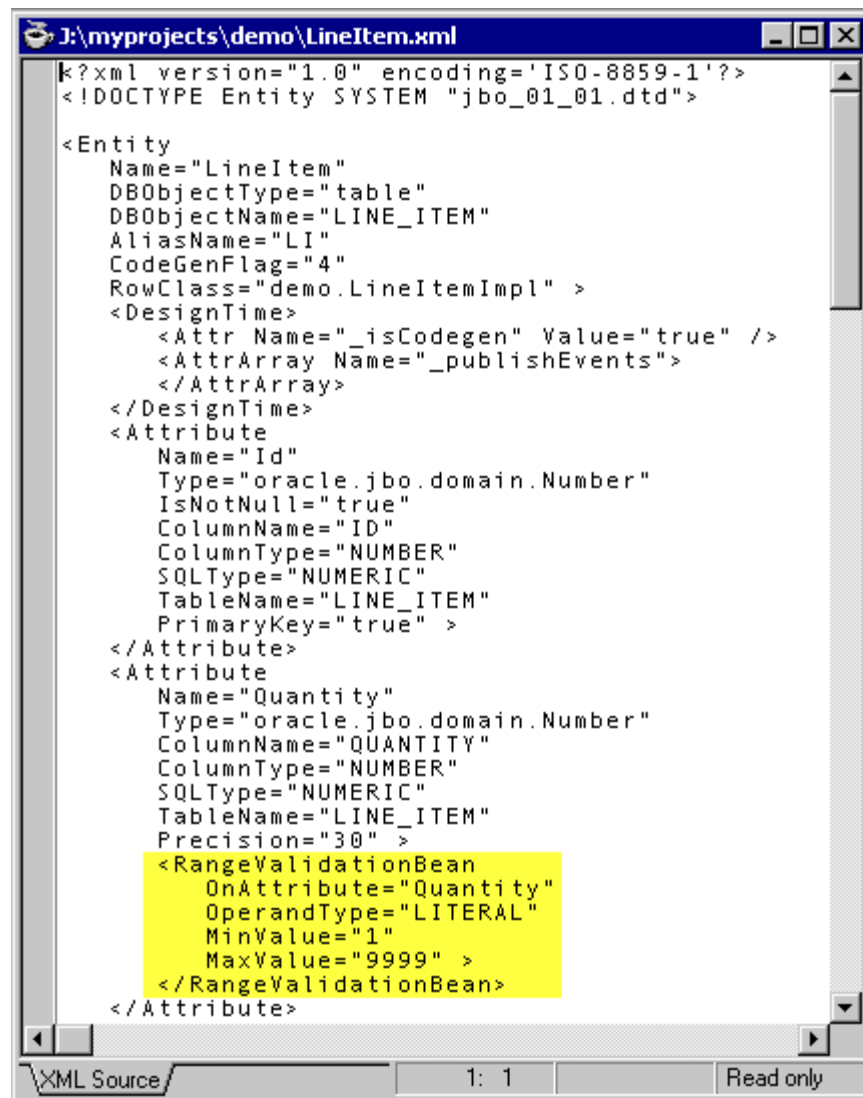


Figure 6: XML Metadata Indicating Use of a Validation Rule Bean for LineItem's Quantity Attribute

Figure 6 shows the result of using the "Rules" tab of the Entity Object Editor in Oracle JDeveloper to attach a **RangeValidationBean** to the **Quantity** attribute of the **LineItem** Entity Object. Since Validation Rules are like little data-driven validation engines housed inside a JavaBean, they typically require a set of parameters to be supplied when they are used. These parameters are captured and represented in the XML Component Definition as XML attributes on the element representing the Validation Rule usage. This makes the values easy to set and easy to customize.

Validation Rules can be attached both to individual Entity Object attributes as well as to the Entity Object as a whole, and the framework supports a collection of zero or more Validation Rule at each place. Validation Rules work in concert with Java code-based validation so you can mix and match the approaches for your own validation.

While Oracle Business Components comes with a set of example Validation Rules, their real power lies in the fact that you can easily implement Validation Rules of your own. After all, they're just JavaBeans that implement a supplied Oracle Business Components interface! This lets you provide libraries of reusable Validation Rules to your development teams or to your end-customer for easy, declarative customization of

your application's behavior.



Domain

When building real applications, you frequently find yourself writing little routines to check:

- the format of a telephone number,
- the checksum of the digits of a credit card number is valid,
- that a string *really* does represent a valid web address, or
- that a FedEx Tracking Number is properly represented.

Domains are immutable classes that represent scalar datatypes that often occur in your applications.

Domains to represent the above situations might be

called **TelephoneNumber**, **CreditCard**, **WebURL**, and **FedExTrackingNumber**, respectively.

Domains contain a validation in the constructor which performs the check you need to do to make sure they are proper values. Once-constructed, instances of Domains can be freely passed around in a type-safe way as method parameters. In fact, Domains makes it easy to avoid the mistake of passing a **Quantity** to a method expecting a **Price** since the compiler will flag the error immediately.

Oracle Business Components lets you easily use Domains as the datatype of Entity Object and View Object attributes, which automatically enforces the Domain's built-in validation checks on any attribute values supplied by clients. *Figure 7* shows an example of the constructor-time validation check for a **WebURL** Domain, the XML metadata representing the use of the **WebURL** Domain as the datatype of the **Customer.Website** attribute, and the type-safe attribute getter and setters generated which make use of the **WebURL** class.

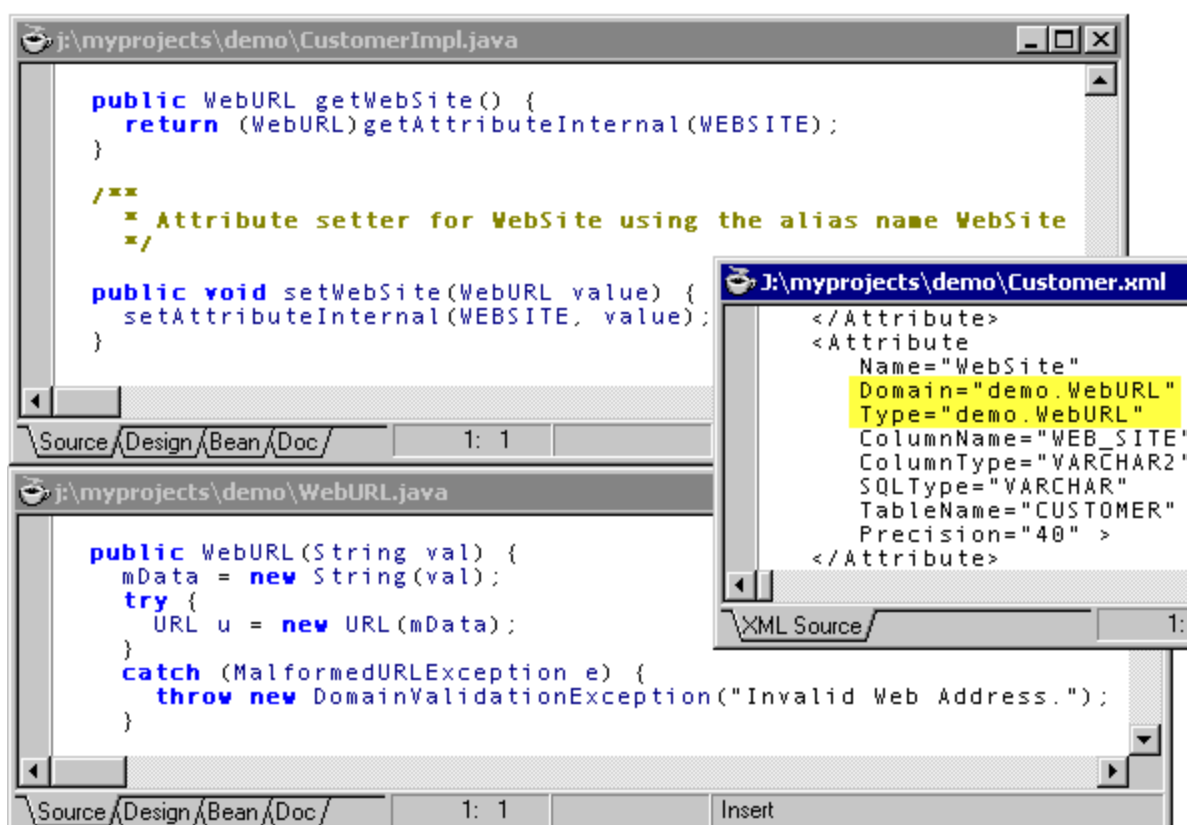


Figure 7: Example of a WebURL Domain and its Use as the Datatype of the Customer Entity's WebSite Attribute

2. Increases Flexibility and Reuse with Updateable, SQL-Based Views

Technical Highlights

- ▶ Create any number of View Objects over existing Entity Objects, to present or manipulate just the "slice" of data you need, exploiting the full power of SQL.
- ▶ Update rows in the View Object's Result Set with automatic, consistent enforcement of relevant business logic.
- ▶ Join any number of tables in your queries with full control over which attributes are updateable.

2.1 Shape Data for the Task at Hand with View Objects



View Object

EJB Entity Beans support a feature called a "Finder Method" which allows the author of the bean to provide alternative "lookup" mechanisms for that Bean. For example, rather than finding an **OrderBean** by its primary key attribute **OrderNumber**, you can support looking it up based on some other parameters instead. Oracle Business Components In addition to these basic EJB "Finder Methods", Oracle Business Componentss in Oracle Business Components give you the ability to:

1. Build any number of views of business data and relate them to existing Entity Objects for consistent business-logic enforcement, without requiring changes to the Entity Objects involved.
2. Use arbitrarily-powerful SQL statements to control exactly the data you need on the client to support a certain business task in an application.
3. Let the database join the relevant information for you, instead of relying on client code to accomplish it. This allows information from multiple entities to be retrieved in a "single shot", including only the combination of attributes which are relevant to the application at hand.

Since SQL has powerful calculation and aggregation capabilities built-in, View Objects can include calculated attributes which are computed by SQL expressions in the **SELECT** list, as well as transient attributes which are calculated by custom Java code in the View Object's Java Implementation Class. Pushing these calculation and aggregation operations into the SQL statement of a View Object allows the database to efficiently compute the answer, rather than having the application code do it. For real-world applications, the ability to hand-tune the data access to improve performance can make all the difference between sluggish and speedy response-time. The Oracle Business Components View Object architecture allows this.

Once defined, View Objects are reusable components that can be assembled as needed into Application Modules or be looked-up by name and instantiated on-demand at runtime.

2.2 Update Data through Any View with Enforcement of Entity Business Logic

Each View Object has related metadata in its XML Component Definition which tracks how the columns in its SQL query's **SELECT** list relate to individual Entity Object attributes. This information allows the framework to coordinate business logic enforcement with values changed by clients on any row queried into the View Object.

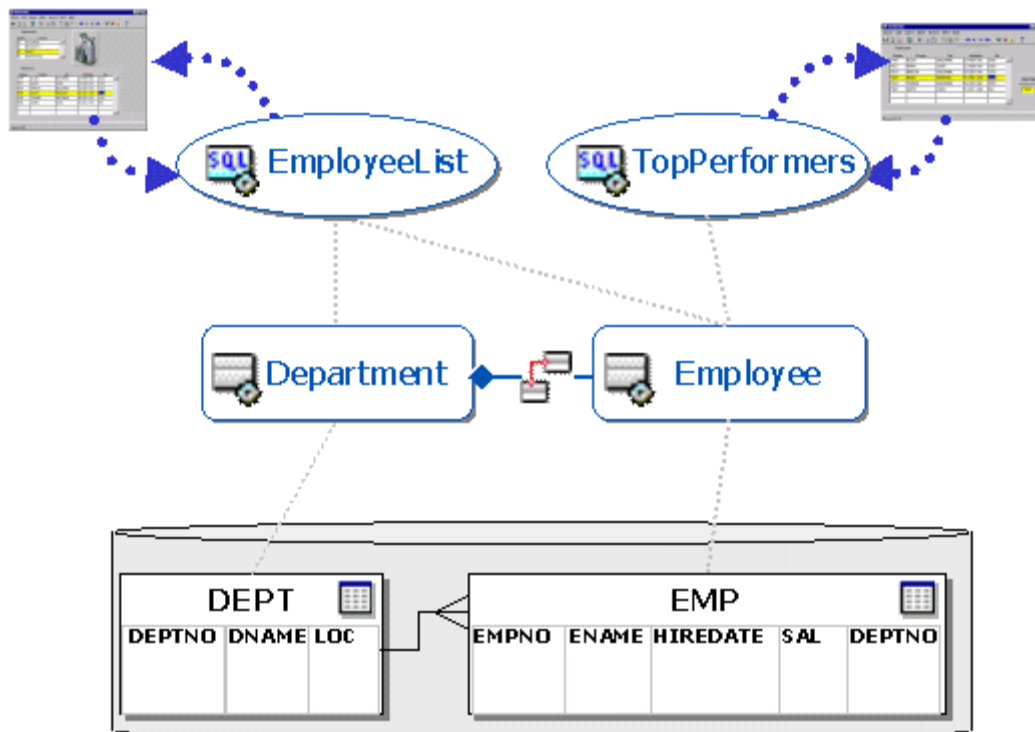


Figure 8: View Objects Cooperate with Related Entity Objects for Business Logic Enforcement

For example, *Figure 8* above illustrates how two different View Object's **EmployeeList** and **TopPerformers** both reference attributes from the **Employee** Entity Object among their result set columns. If we assume the **Employee**'s **Sal** attribute is included in both View Objects, then any validation rules on **Sal** will automatically trigger when either view tries to make modifications to it.

In fact, **Sal** need not even be **included** in the view at all. For example, assume that an **Employee** Entity Object has business logic which raises the employee's salary when she is promoted. And assume that the client is working with a View Object which only includes **EmpNo**, **ENAME**, and **Level**. If the client sets the value of an employee's level to a value which is higher than the current value (indicating a promotion), then if the promotion-related business logic causes the employee to have an invalid salary, the exception will be consistently raised back to the client.

2.3 Full Control Over Every Aspect of the View Object

View Object SQL queries often need to join many underlying tables to have just the right information available for the end-user's screen or web page. Look again at *Figure 3* and think about the information that a **user** requires to complete the task of entering a new **LineItem** on an **Order**.

They need to see **Price** and **Description** information on the item from the **InventoryItem** as well as the **Name** of the **Supplier** at a minimum.

View Objects handles this easily. You simply let the View Object Editor build the join query for you. Then, you use the View Object Editor to specify how the columns selected in your query related to the appropriate attributes of the underlying Entity Objects involved. Typically join queries are not updateable, but with the

Oracle Business Components framework, you can use the View Object metadata to make any join fully updateable for you.

Since sometimes **fully**-updateable is not what you need, you have full control over each attribute included in the View Object and can specify whether it is read-only, updateable, or updateable-only-when-new to ease the burden of having to code this into any subsequent user interface or web page that might make use of the View Object.

3. Reduces Complexity by Coordinating Multiple Views of Data

Technical Highlights

- ▶ Change column values in one View Object and other View Objects with those columns stay synchronized.
- ▶ Coordinate master-detail hierarchies of View Objects to any depth using declarative View Links.
- ▶ Update a column value relating to a referenced entity and related information is automatically looked-up and filled-in.

3.1 Multiple Views Referencing the Same Entities Stay Synchronized

One of the hardest tasks for developers building client applications is to provide end-users multiple, synchronized views of data. How many times have you seen applications that display summary information in a grid, then drill-down to details in a form, and maybe show the big picture by rendering the same information graphically as a chart? These are the kind of applications that end-users love but developers hate, since keeping the various views synchronized as the user makes changes or navigates through the data is non-trivial and hard to generalize. Oracle Business Components solves this problem by providing you framework components which automate all of the hard parts of doing this.

Oracle Business Components' View Objects cache their queried data intelligently by factoring each row into its constituent Entity Object parts, and caching each Entity Object instance only once in memory, regardless of how many View Objects may be referencing it. This fundamental processing behavior has the benefit of automatically keeping multiple views of the same data in sync. When any view makes a change to any attribute, the change is **effectively and automatically** made to the corresponding attribute on the unique cached instance of the Entity Object underneath. This lets all View Objects in the same session instantly see the change, eliminating the need to write any code manually to take advantage of this feature.

3.2 Coordinate Master-Detail Queries with View Links



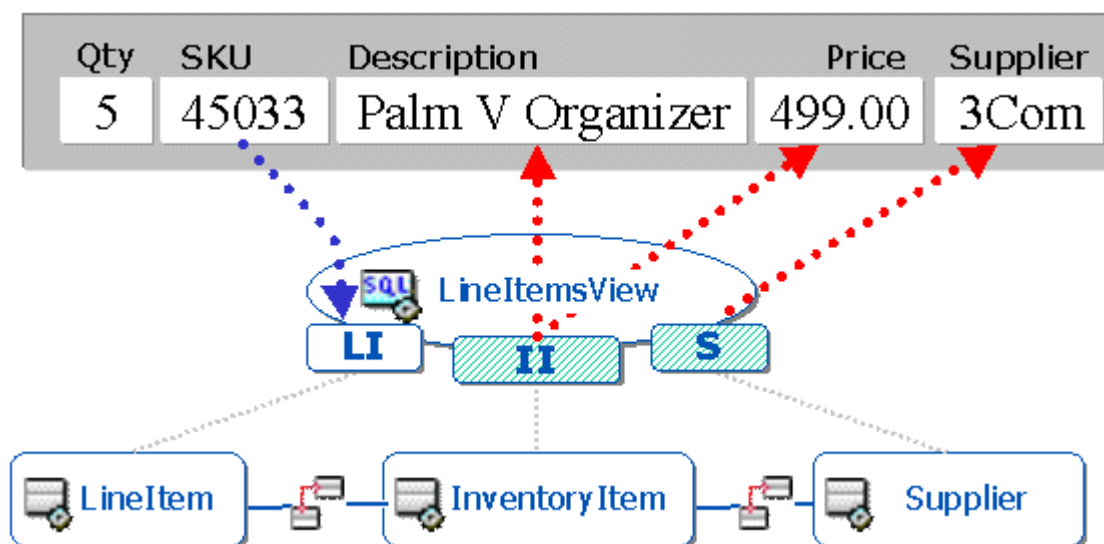
The next coordination challenge for developers is keeping master-detail queries synchronized. At design-time, you can define a View Link to establish a master-detail relationship between two View Objects. At runtime, the View Link automatically keeps a detail View Object resultset synchronized with the master View Object resultset. Of course, View Objects can be instantiated under the coordination control of an appropriate View Link as well as on their own, so reuse of View Objects is improved.

When a JFC/Swing-based user interface or JSP page works with information from two View Objects that are View Linked, it can simply make calls like `masterVO.next()` and the information in the master-detail View Objects is automatically kept synchronized without client code. This can significantly reduce the code required for sophisticated user-interfaces.

3.3 Reference Information Automatically Retrieved Using Associations

Another common task typically relegated to repetitive, unproductive client code is that of managing the retrieval of lookup values. If the user types in the `ItemId` of some `InventoryItem` on a form where they are typing in line items, the client code is usually responsible for looking up the `InventoryItem` by the supplied `ItemId` and retrieving its `Description`, `Price`, and Supplier `Name`.

With Oracle Business Components, since each View Object knows which Entity Objects it is using and since you've already specified how the Entity Objects are related using Associations, Oracle Business Components can keep the information together *for* you to save you from writing the common code to bring in reference information.



**Figure 9: View Object Attributes from Referenced Entity Object Usages
LineItem and Supplier Are Automatically Retrieved When ItemId is Set to
a New Value**

Figure 9 illustrates how the act of setting the **ItemId** attribute on the current row in the **LineItemsView** causes the framework to automatically retrieve appropriate reference information from the **InventoryItem** and **Supplier** Entity Objects used by the View Object. The slanted-line pattern on the **II** and **S** Entity Usages indicates that they have been marked as participating in the View Object as reference information, rather than being updateable.

4. Simplifies Deployment by Making JSP- and Swing-based Clients Easy

Technical Highlights

- ▶ Define the logical data model and business methods needed to support an application task using an Application Module.
- ▶ Bind Swing Controls declaratively to View Object columns, entire VO result sets, or a tree of view-linked result sets from an Application Module.
- ▶ Reference View Objects easily for formatting and updating data within JavaServer Pages (JSP), optionally leveraging DataWebBeans for automatic HTML presentation.

4.1 Capture an Application's Data Model and Business Services Using Application Modules



Application Module

To keep the client user interface code thin, you can pre-build the entire data model that the client application needs into an Application Module. This includes not only the View Objects which the application will make use of, but also any View Links which will establish the automatic master-detail coordination behavior the client may require. By setting this all up ahead of time, the client application can simply instantiate the application module it needs and immediately find all the appropriately-coordinated View Objects ready to go, saving network roundtrips and startup time.

Figure 10 shows the tree of the View Objects and View Links required to implement the OrderSystem Application Module from Figure 3.

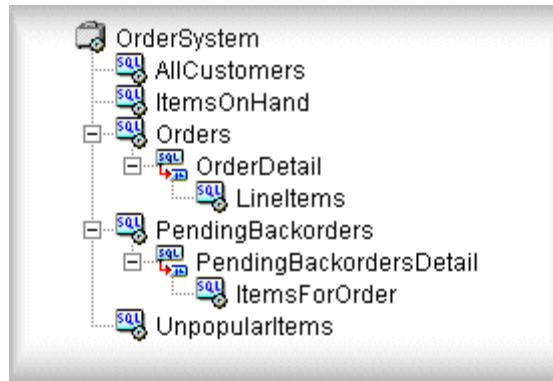


Figure 10: Data Model of the OrderSystem AppModule

But thinking of the Application Module as **only** a data model for your application misses the other important role an Application Module plays in building an efficient, three-tier application: that of supplying server-side services which the application client may need.

Service methods defined on your Application Module can be made remotely accessible simply by selecting the ones you want in the "Exported Methods" panel of the Application Module Component Editor. When some custom methods on the Application Module are exported, Oracle Business Components automatically creates your remote interface containing the custom method signatures and automatically handles the runtime marshalling of method arguments, return values and exceptions back from the server-side method implementations. By moving code like a data-intensive calculation out of the client and into its Application Module counterpart in the application tier, the client can get the answer with a single network round-trip instead of unnecessarily pulling data to the client.

To support straightforward component assembly of applications, Application Modules can be composed of other Application Modules if required. As illustrated in *Figure 11*, if user interfaces forms or panels have been coded to work with a given Application Module, these self-sufficient UI panels can be reused and composed into compound user interfaces whose logical application layer can be a nested set of Application Module components.

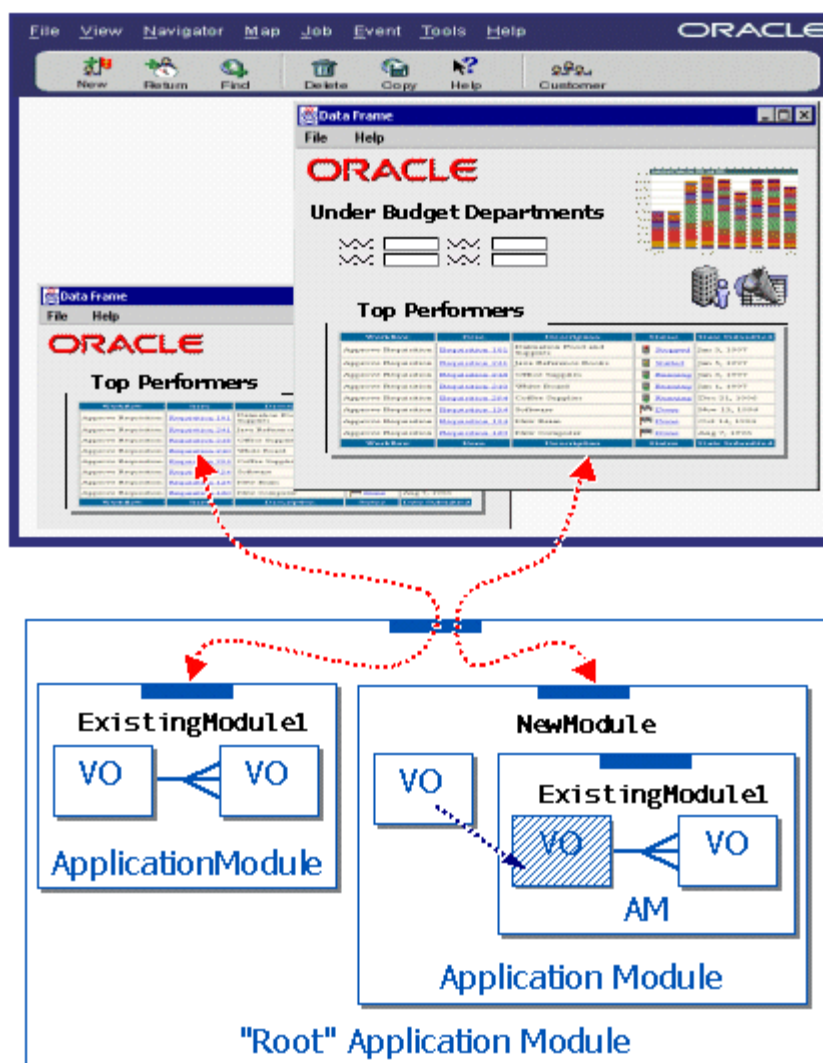


Figure 11: Assembling Application Module Components

The developer can use an instance of the base Application Module component as a generic, initially empty container for dynamically loaded View Objects and Application Modules to support complex, role-based applications where a fixed set of the Application Modules required by the end user are not known in advance.

4.2 Data-Aware Swing Controls Bind Easily to View Objects

Oracle JDeveloper comes with a full set data-aware JFC/Swing controls which bind declaratively to data from database queries. In Oracle JDeveloper these controls enable the same declarative data binding for thin-client Java user interfaces working against the View Objects inside an Application Module.

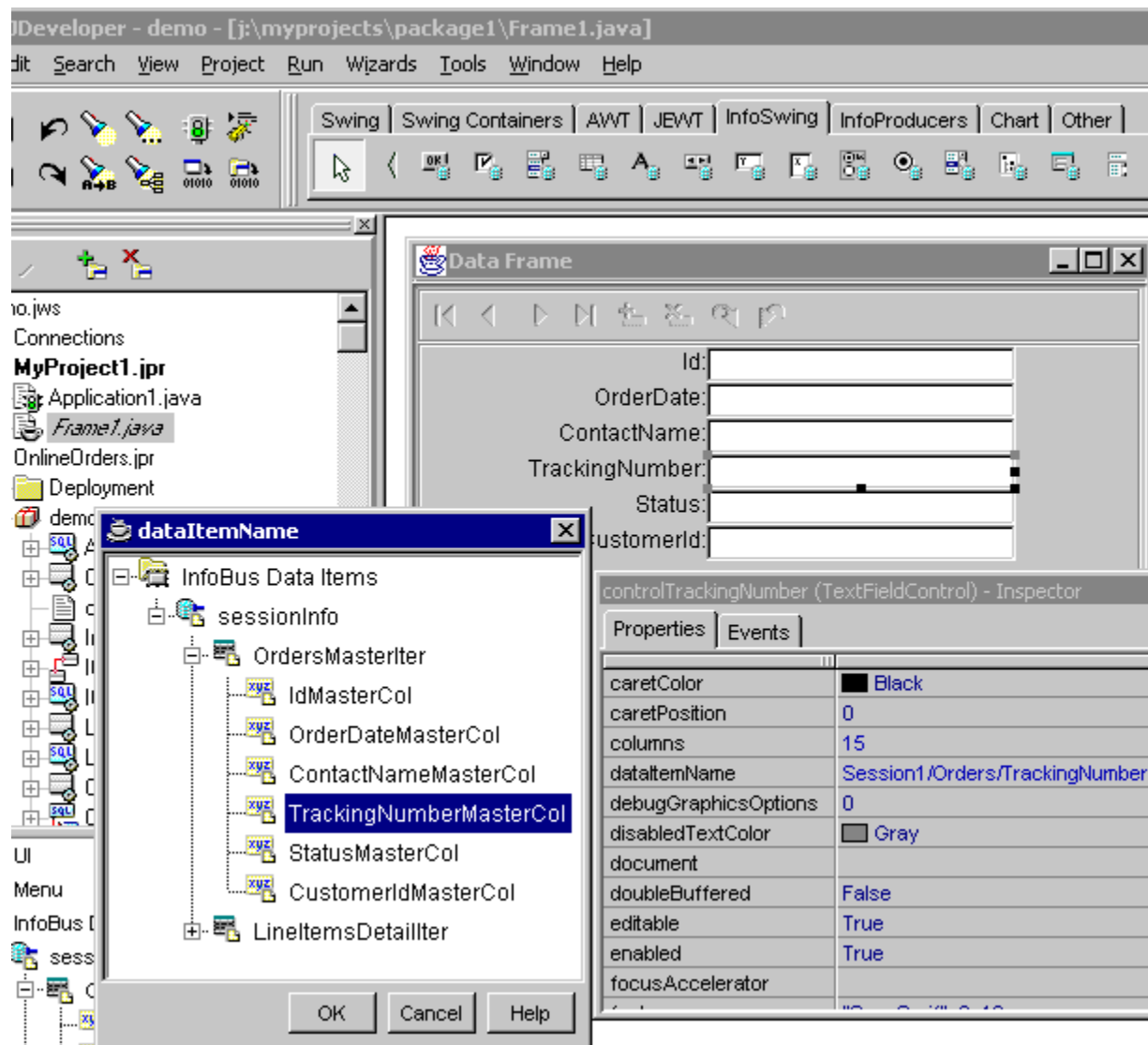


Figure 12: Declarative Data Binding to View Objects Using "InfoSwing" Controls Data

Figure 12 shows the binding editor used to visually select the column or rowset to which an InfoSwing control is bound. In this case, it sets the Infobus "dataItemName" property on the text field control to **Session1/Orders/TrackingNumber**. At runtime, this means the control will display and can modify data from that column in the View Object automatically, without any manual coding from the developer.

4.3 JSP Pages Make Easy Use of View Objects Using DataWebBeans

Similar ease-of-use features exist for the JavaServer Pages for developer wanting to use View Objects and Application Modules. Oracle Business Components supplies a set of JavaBeans which can be used with the JSP `<jsp:useBean>` tag for full programmatic control over View Objects and Application Modules from within your JSP. In addition, Oracle JDeveloper provides WebBeans, which can be used to access values from the current row, scrolling, and setting View Object column values from HTTP request Form parameters automatically. Exceptions raised by the Entity Object validation behind the View Objects

accessed by the JSP can be presented in a custom JSP Error Page with ease.

In addition, a set of pre-built, HTML generation beans is provided which derive from the Oracle Business Components DataWebBean framework class to automate common HTML presentation for View Objects like Web Forms, tables, etc. Of course, developers can use Wizards to create their own custom DataWebBeans as well.

Additional Oracle Business Components features of interest to JSP developers are the **AppModuleRegistry** and the *forward-only* mode for View Objects. JSP developers use the supplied **AppModuleRegistry** to automatically maintain pools of Application Module instances which are used to support large numbers of browser-based users. A benefit of combining the stateless use of a pool of Application Module instances that each retains some state is that frequently accessed data can stay cached in the View Objects of each Application Module for even faster response time.

By setting an instance of a View Object into *forward-only* mode, a web developer can consciously avoid all caching for View Object query results when she knows ahead of time that no scrolling and no updates will be made to the queried data through *that* View Object instance. Typically, this is the case when formatting query results for HTML or XML web documents. The same View Object can be instantiated multiple times, so the same View Object could be used in forward-only mode for fast web-page formatting, while another instance could operate in normal mode to accommodate updates and automatic business logic enforcement when a web user eventually submits an HTML form with pending changes to be made.

5. Saves Time and Avoids Bugs by Eliminating Hand-Written Database Code

Technical Highlights

- ▶ Modify columns of View Objects with support for smart caching and pessimistic locking of associated Entity Objects instances.
- ▶ Commit your work and all modified Entity Objects are automatically inserted, updated, deleted as appropriate.
- ▶ Traverse Associations in your code while the framework handles underlying SQL lookups.

5.1 Automatic Caching and Pessimistic Locking for Entity Objects

Oracle Business Components View Objects automatically cache their queried data factored into Entity Object parts using the metadata all View Objects components have about how their columns relate to one or more Entity Object Attributes. This technique allows the database to process the SQL query at full speed, while making the data returned more powerful once in the cache.

Since data is cached by Entity Objects, **they** can handle the interaction with the database when valid changes are made to any of their attributes. The first of this automatic behaviors is row-level locking.

Oracle Business Components supports a declarative setting for either pessimistic or optimistic locking. When an entity object is set for pessimistic locking, the first time one of its attributes is successfully modified (i.e. passes any validation), the Entity Object will attempt to lock its corresponding row in the database. If the Entity Object is part of a composition of Entity Objects, then the framework coordinates locking the parent Entity Object in the family before proceeding. If the lock cannot be acquired, exceptions can be trapped for the application to handle.

5.2 Entity Object Database Interaction is Handled Automatically

When new instances of Entity Objects are created, or existing instances are modified or deleted, the affected Entity Object instances automatically communicate the changes to the underlying database table at transaction commit-time. All of the pending changes made to Entity Objects occur in the context of the current transaction, which is associated with the root Application Module and shared by all contained Application Module components.

A critically important feature for application developers is that while changes are pending in the cache, Entity Object business logic can walk associations to check on the state of related objects. This business logic will correctly see any pending changes on the related objects so that business logic can be correctly enforced **before** committing the changes as a unit to the database.

When the client application or some code within the application-tier business logic calls `getTransaction().commit()`; all of the modified Entity Objects in the cache flush their changes efficiently to the database, again without developer-written code!

Of course, one of the key values of a Java-based application framework is that when you don't want the **default** behavior provided by the framework, you can override the framework in strategic places to get things to work as **you** need them to for a particular case. Overriding the default database interaction mechanism for an individual Entity Object is as easy as overriding a single method in that Entity Object called `doDML()`. Teams requiring custom storage implementations or, for example, requiring that the deletion of an **Order** actually **update** a **Deleted** column instead of physically removing the row, achieve their desired results quickly using this approach.

5.3 Associations Save Writing Mundane SQL for Traversing Relationships

Figure 4 shows how you can use Associations to walk the web of relationships among your business objects. When you take advantage of this feature, you are actually saving the work of writing the mundane SQL lookups which would normally be required to lookup the related information by its foreign key values.

6. Enables Dynamic Applications through Active Metadata

Technical Highlights

- ▶ Dynamically create new View Objects and View Links at runtime, as well as easily load existing View Objects, View Links, and Application Modules by name.
- ▶ Define custom metadata for all business components and access them at runtime to support metadata-driven user interface or application functionality.
- ▶ Add dynamic attributes to View Objects at runtime to simplify per-row state management.

6.1 Load, Create, and Remove Application Components Dynamically

Oftentimes, an application developer can only know at *runtime* the exact application components that are required, based on the diverse roles and responsibilities of each user of the system. The Oracle Business Components framework caters to this need by allowing individual View Objects or entire Application Modules with pre-coordinated View Objects to be loaded on-demand by the client application at runtime.

By simply referencing the full package-qualified name of the business component, the developer can easily instantiate an application module in the current session, then find one of its View Objects by name as shown in the following code example.

```
ApplicationModule theAM = createApplicationModule("oracle.apps.order.OrderSystem");  
ViewObject       theVO = theAM.findViewObject("LineItems");
```

The following code example shows how to instantiate a single View Object at a time.

```
ViewObject myVO = createViewObject("oracle.apps.order.UnpopularItems");
```

Once the application decides it no longer requires a particular View Object or Application Module, it can simply call

```
theAM.remove();  
myVO.remove();
```

to free up the server-side memory used by the components metadata and any cached rows the components might have been using.

6.2 Use Custom Component Metadata to Drive Generic Applications

Another feature which supports generic, data-driven application behavior is the ability to associate **custom metadata properties** with any business component built using the framework. You can associate arbitrary name-value pairs of information in any component's XML Component Definition, and a simple API makes this custom metadata available at runtime to drive dynamic decisions which can be made by generic code.

Custom properties can even be assigned to Domains as well as to each Entity Object or View Object Attribute. Oracle Business Components users may make clever, comprehensive use of custom properties on their

- Application Modules, to indicate what style of generic user interface template to use,
- Domains, to globally set default display style hints and format masks for certain frequently used kinds of attribute types,
- Entity Object and View Object attributes to, where necessary, override the default hints provided by the Domains.

The net result of using custom metadata properties is a completely generic, template-based, customizable user interface, which is consistent, attractive, and requires nearly zero UI development cost to add new screens.


6.3 Add Dynamic Attributes to View Objects at Runtime

A final small, yet important, feature is the ability to track your own per-row application state on any View Object by adding dynamic columns at runtime. Since the Oracle Business Components framework is designed to keep the client memory and code footprint very thin by doing all of the hard work in the application-tier, you can have the application-tier cache and manage your dynamic state information in the middle-tier as well.

Simply call a method on the desired View Object to **AddDynamicAttribute()** and then proceed to get and set that new attribute with your own information. As with any View Object Attribute, the type of the dynamic attribute can be any **Serializable** object. The net benefit is that for a large View Object result set, your dynamic attributes are cached and managed in the same way as regular attributes on the View Object. This means that the bulk of their data is managed in the middle-tier, rather than hanging around en masse on the client.

7. Improves Productivity through Tight Integration with Oracle JDeveloper

Technical Highlights

-  Define, edit, and extend components using integrated Wizards and Component Editors that automatically maintain framework-related Java code

and XML metadata.

- ▶ Reverse-engineer Entity Objects and Associations from existing database schema, or forward-engineer schema from Entity Objects.
- ▶ Test and debug functionality of server applications on-demand from the IDE without manually creating a user-interface or test program.

7.1 Modify Component Features Easily Using Wizards and Component Editors

While developers **can** leverage **Oracle Business Components for Java** by working on the source of their components' Java Implementation Classes and XML Component Definitions in their favorite editor, Oracle JDeveloper provides integrated design-time support for the Oracle Business Components framework, which can automate many aspects of putting the framework to use for your application.

Figure 13 shows the Oracle JDeveloper Integrated Development Environment (IDE). The Oracle JDeveloper Project Navigator displays business components such as Application Modules, Entity Objects, and Associations. Key details of each business component appear for easy reference in the Oracle JDeveloper Structure Pane. Expanding any component's icon in the Navigator reveals the component's XML Component Definition as well as its Java Implementation Class which can be worked with in the IDE exactly like *any* Java file.

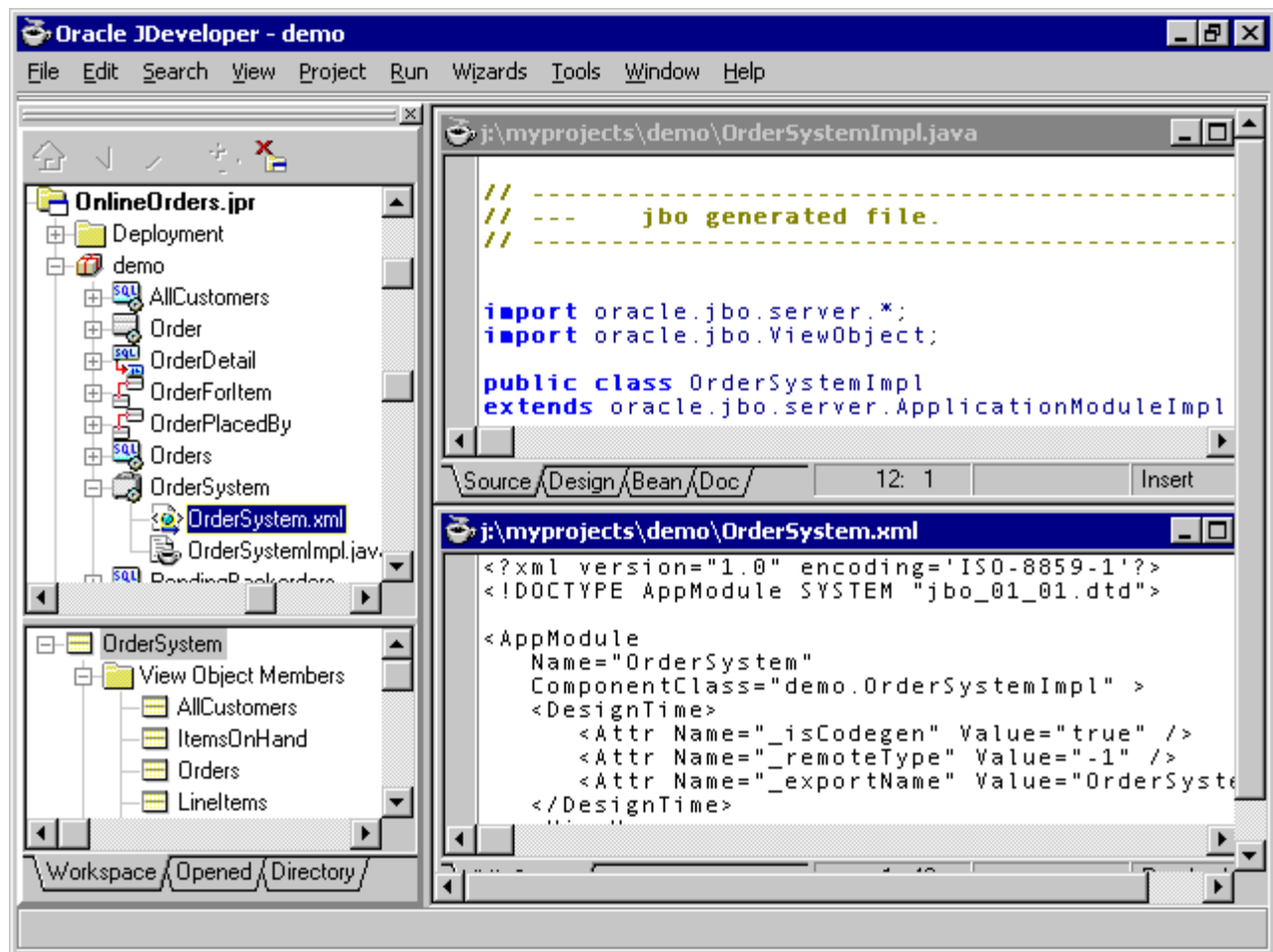


Figure 13: Oracle JDeveloper IDE Offers Built-in Support for Oracle Business Components Framework

Using integrated design tools such as the Oracle Business Components Wizards and Component Editors, you define the characteristics of all the domain-specific components you need for your application. After creating new components with the Wizards or changing any aspects of them using the Component Editors, the Oracle Business Components design-time system modifies any framework-related Java code in your Java Implementation Class and keeps your XML Component Definition file synchronized with your changes. Since **all** of the code in your components' Java Implementation Classes are domain-specific business logic and framework overrides, in practice the only **code** changes to these files that the editors need to make on your behalf are:

- Insuring the datatypes in the signatures of your getter and setter methods matches those you've defined for your Entity Object and View Object attributes and associated objects.
- Optimizing attribute access by maintaining a list of attribute accessor constants,
- Generating any event subscription code for you when your components publish or subscribe to events on other components (another built-in framework feature).

Figure 14 illustrates one panel of the Application Module Component Editor which allows you to define the tree of View Objects and View Links you need for your application with a few clicks of the mouse.

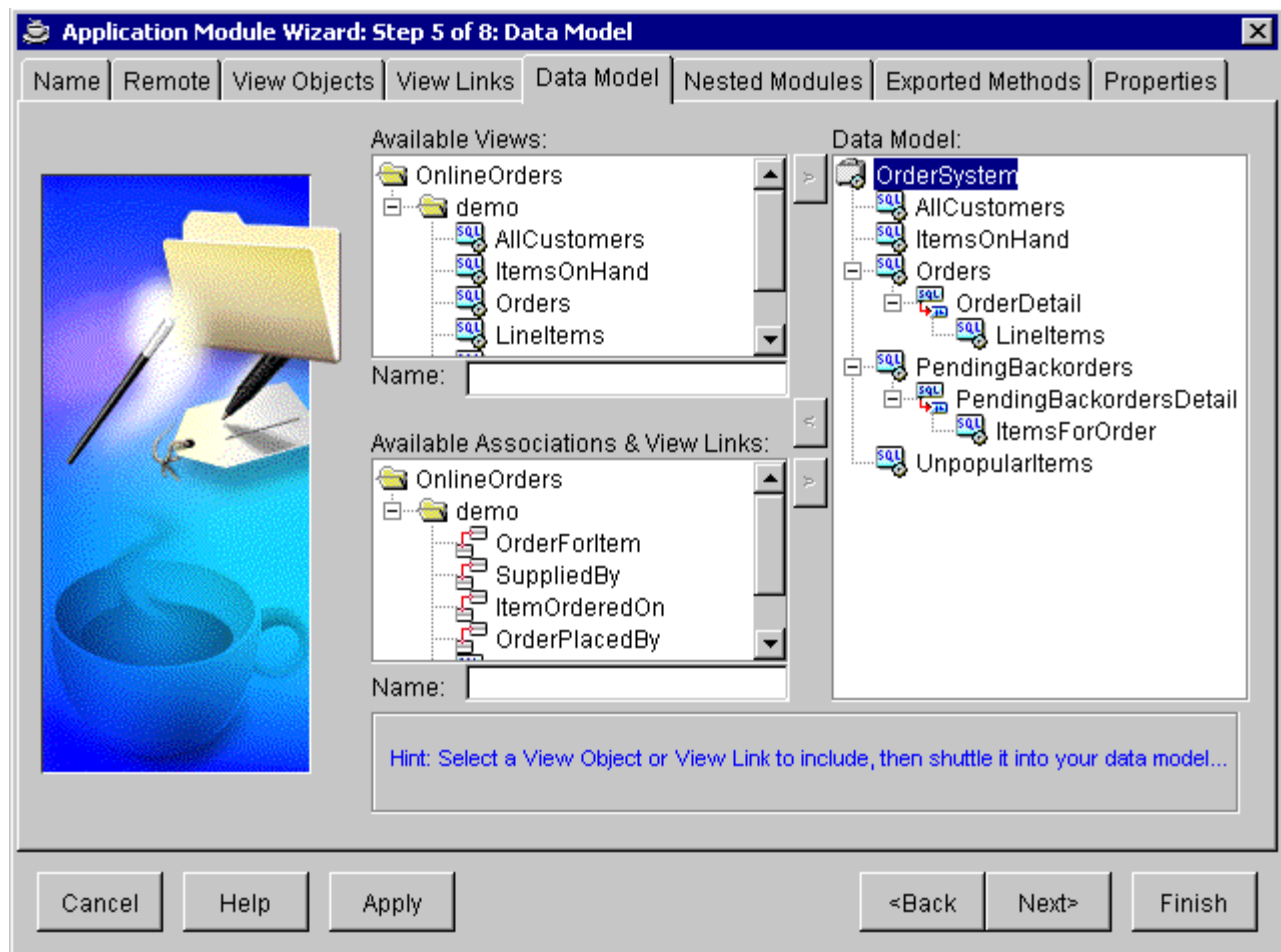


Figure 14: Defining the Set of Master-Detail-Coordinated, Remoteable Collections in the Application Module Editor

In practice, many application developers mix and match development approaches, using the Oracle Business Components Wizards and Component Editors in Oracle JDeveloper to set up the basic features of their components and their relevant XML Component Definitions, then using their favorite editors for marathon coding or bug-fixing sessions where they concentrate for days at a time on the implementation of their components' domain-specific business logic.

7.2 Reverse-Engineer Entity Objects, Forward-Engineer Tables

Oracle Business Components and Oracle JDeveloper support both forward-engineering for creating the database tables from Entity Objects, as well as reverse-engineering of all the basic framework components from the existing tables. For example, in *Figure 15* the user has

- selected the **CUSTOMER**, **INVENTORY_ITEM**, **LINE_ITEM**, **ORD**, and **SUPPLIER** tables,
- checked the **Create Default View Objects?** checkbox, and
- checked the **Create Default ApplicationModule?** checkbox,

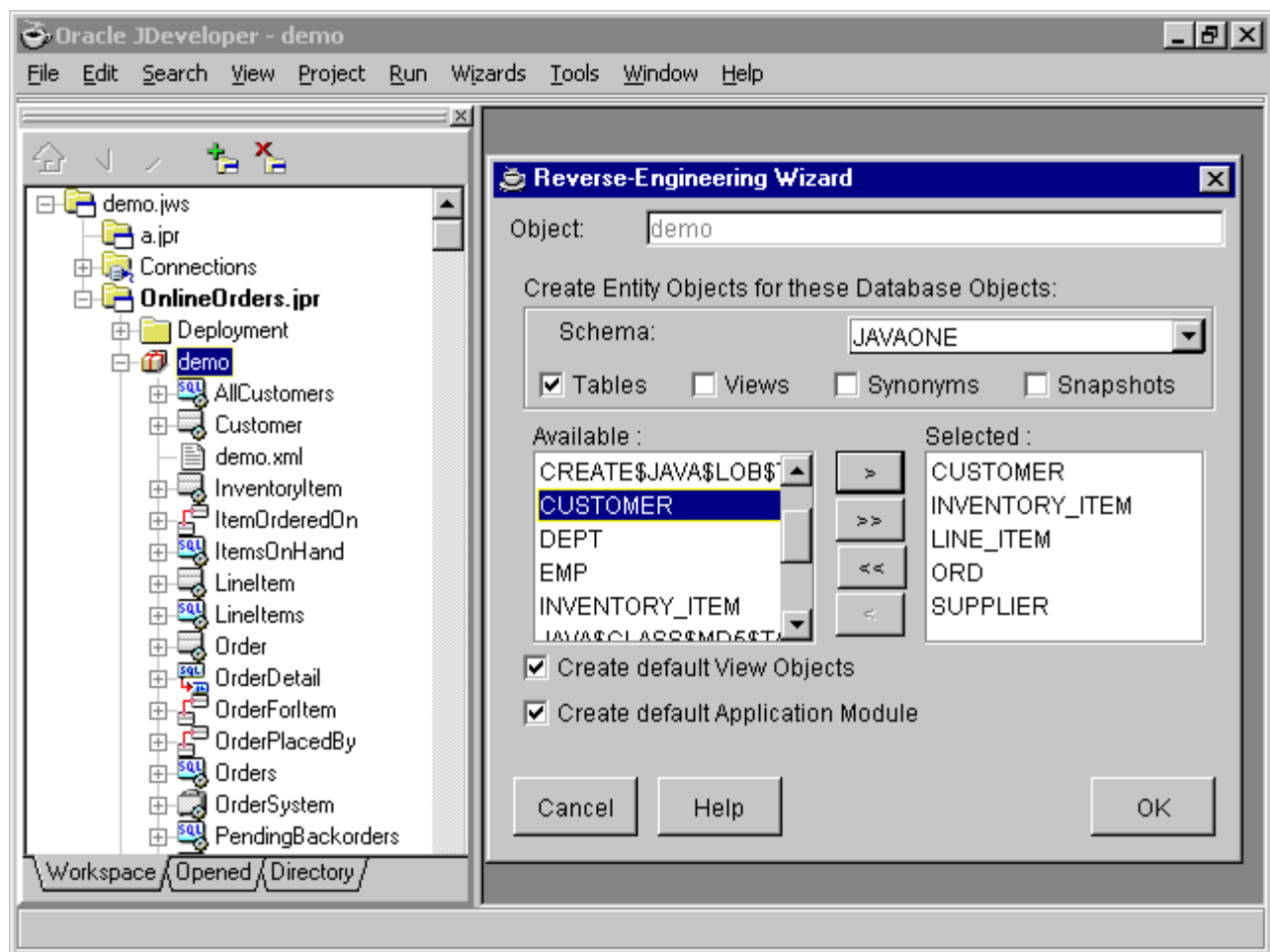


Figure 15: Using the Reverse-Engineering Wizard to Create Default Entity Objects, View Objects, and an Application Module in One Click

When the OK button is clicked, the Reverse-Engineering Wizard automatically creates

- An Entity Object for each table selected,
- An Association between appropriate Entity Objects for each referential constraint defined in your database schema related to the selected tables, properly inferring the cardinality for both sides of the Association
- A default View Object for each table selected,
- A default View Link between appropriate View Objects for each Association to coordinate their query results master-detail, and
- A default Application Module, referencing all of the defaulted components.

Very quickly, this results in an initial, fully functional Application Module ready to test right-away, and ready to be evolved to support the domain-specific features you need for *your* application.

7.3 Test and Debug Server Applications On-Demand

One of the many benefits of cleanly separating the application business from the views of business data is that teams working on business logic can work independently of application teams *reusing* their Entity

Objects in various applications. If creating user interfaces is neither your speciality nor strictly *related* to your job of writing business logic, you'll really like the built-in Oracle Business Components Tester. It gives you a productive way to test-out your business logic immediately and directly from the IDE by selecting any Package or Application Module in your project in the Project Navigator, and picking **Test...** from the right-mouse menu.

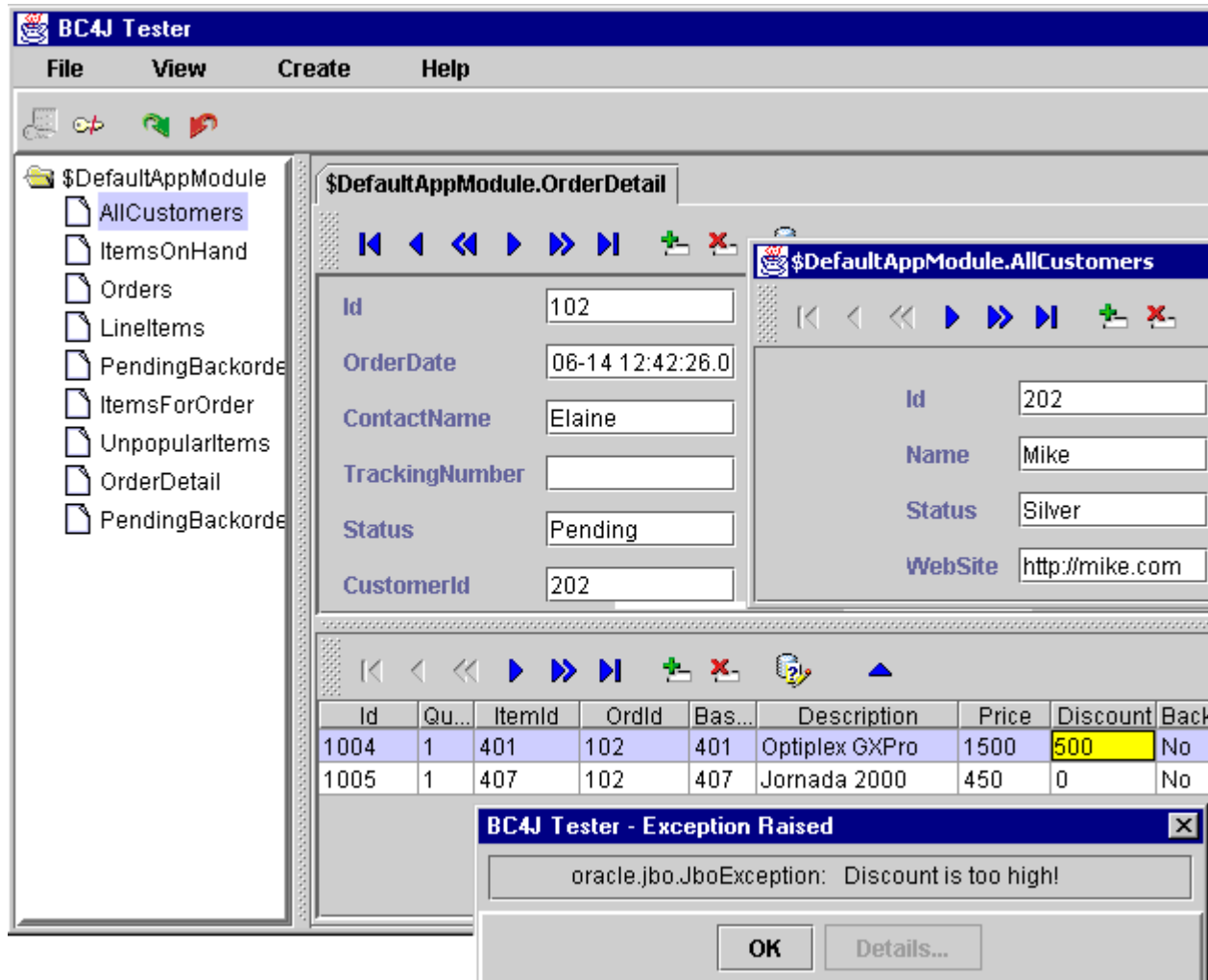


Figure 16: Testing Business Validation on the LineItem Entity Object's Discount Attribute by Trying to Set an Invalid Value for Discount through the LineItems View Object in Oracle Business Components Tester

The Oracle Business Components Tester, shown in *Figure 16*, is an example of a generic, thin Java client application. It uses the dynamic capabilities of the framework's client API's to interrogate the runtime metadata of any Application Module that you want to exercise, and presents a directory of the View Objects and View Links contained by the current Application Module. The Tester creates dynamic Swing-based forms for each View Object you want to browse, and dynamic master-detail forms for each View Link that you test out.

The Oracle Business Components Tester visually exposes all of the runtime features of the framework including:

- View Objects, which can be executed, scrolled, and have the values in any of their columns changed,
- Entity Object business logic, which will be enforced when changes to related View Object columns are made so you can verify that domains are setup properly, that default value logic functions correctly, that validation code is working, and that association lookups do what you expect,
- Multiple View Objects, which include the same entity attributes stay synchronized when changes are made through any of them,
- View Linked View Objects are automatically master-detail coordinated as you scroll through master records to any level of depth,
- Application Modules can be tested in any supported deployment configuration from a *single* application: Local, Remote CORBA/EJB in a physical middle-tier, as well as Remote CORBA/EJB inside of Oracle8i.
- When you commit, all pending changes are communicated to the database by the underlying Entity Objects in the cache.

The Oracle Business Components Tester also supports:

- Running a0 client application (in this case the Oracle Business Components Tester application *itself*) against any supported deployment configuration of your Application Module without client or server code changes of any kind. This includes testing the client connecting to the Application Module locally, deployed as a CORBA or EJB Server in a physical middle-tier, or deployed as a CORBA or EJB server-object inside of Oracle8i.
- Efficient bundling of network traffic between client and application-tier, and
- Lazy coordination with the remote Application Module service to even further reduce network traffic.

Seeing the Oracle Business Components Tester's generic client in action working with the View Objects in your Application Module it becomes immediately clear **how** much of your application logic and coordination you can encapsulate into the logical application-tier using Oracle Business Components, without burdening the client with unnecessary complexity and without weighing down the client with unnecessary code. Finally, besides being a useful testing tool, the Oracle Business Components Tester is also a useful *learning* tool. The full source code for the Oracle Business Components Tester ships as a sample application with Oracle JDeveloper.

8. Boosts Performance with an Efficient, Coarse-Grained Service Architecture

Technical Highlights

- ▶ Access View Object result sets remotely with partial object-fetching and middle-tier caching.
- ▶ Optimize network traffic with client-side caching, bundling of client/server messages, and lazy server-synchronization.
- ▶ Avoid remote interface *explosion* by using at most

one remote CORBA/EJB per coarse-grained Application Module.

8.1 Partial Object-Fetching and Middle-Tier Caching

Based on the metadata in the XML Component Definition about how its columns relate to the attributes of one of more Entity Objects, the View Object partitions its queried data into appropriate row caches at runtime. This architecture enables many of the runtime benefits described in this paper.

As *Figure 17* illustrates, when the client calls `myEmployeeList.executeQuery()` on the remote View Object in the Client Tier, the View Object implementation residing in the Application Tier sends its query directly to the database for immediate execution. The View Object fetches the database query results and does the following:

- Caches column values corresponding to each Entity Object in use in the View Object,
- Caches columns calculated in your SQL statement in the row cache specific to the View Object itself.
- Keeps a reference in the View Object's row cache to the Entity Object instances which it has created.

The net result is a dramatic memory savings for your application data because:

- Unique data from Entity Objects is stored only once in the cache,
- Only Entity Object Attributes corresponding to columns in your View Object query are brought into memory.

If you ever make *modifications* to column values of rows in your View Object, the View Object implementation in the Application Tier delegates these attribute change requests to the appropriate Entity Object in the cache which will, on-demand, fault-in the remaining attributes for itself and perform any necessary business logic enforcement. The result delivers the best of both worlds: you only fetch and work on the information you need for display, and the framework handles the details of faulting in just the instances that you *actually* modify. This capitalizes on the fact that in practice, real-world applications spend a lot more time querying and browsing than updating.

In stark contrast to the approach taken by nearly all object/relational mapping tools/layers, with Oracle Business Components you will **never** query 200 attributes of a complicated **Order** objects, just to reference its **TotalAmount** attribute on a user-interface.



Look again at *Figure 17*. Notice that the caching of the data lies in the Application Tier, and not in the Client Tier. This means what the picture says: the bulk of the data in an Oracle Business Components three-tier application is kept in the logical application tier, managed by your coarse-grained Oracle Business Components Application Module. The client working remotely with the Application Module and View Objects over IIOP or RMI never needs to have more than a small amount of the data currently being displayed on screen down in the Client Tier.

The Oracle Business Components framework components and runtime remote client API provide the automatic management of these remotely-accessible SQL-based View Objects, including the local caching of the (developer-configurable) window of data onto the bulk of the rows kept in the Application Tier.

8.2 Significant Network Traffic Optimizations

Over a standard CORBA and EJB remoting architecture, the Application Module implements additional optimizations to keep the number of network roundtrips between Client Tier and Application Tier to an absolute minimum. The network performance optimization delivered by the Oracle Business Components framework is particularly important for building scalable three-tier enterprise applications. This **coarse-grained** approach of delivering application services as EJB Session Beans which internally manage business entities, provides a solution to highly scalable applications.

8.3 At Most One CORBA/EJB Remote Interface per Application Module

When you deploy the Application Modules as CORBA Server Objects or as EJB Session Beans, by design you never need more than one *single* CORBA or EJB remote interface for your application component. This keeps the environment simpler by avoiding the *explosion* of classes and interfaces typically seen by using EJB Entity Beans for all services.

With Oracle Business Components you need just one remote interface for each coarse-grained Application Module you deploy as a CORBA Server Objects or EJB Session Bean. If your Application Module does not need to expose any remotely accessible business methods, then you don't need **any** extra remote interface for it. The default one provided with the Oracle Business Components framework will be sufficient. All of the Oracle Business Components features described in this section contribute to a simpler, more efficient, more network performant architecture for your distributed enterprise applications.

9. Permits Logical 3-Tier Applications to Target Multiple Physical Deployments

Technical Highlights

- ▶ Achieve logical separation of client, application, and database tiers by coding to the tier-independent client service APIs.

- ▶ Deploy the logical client, application, and database tiers in multiple physical configurations with no code changes.

- ▶ Avoid the complexity of raw CORBA and EJB programming by writing client code that works with simple rowset and session interfaces.

9.1 Application is Logically Separated into Three Tiers

Oracle Business Components presents a productive programming model for distributed application development which is tier-independent and completely based on industry standards. Client applications (both visual or non-visual) use a simple set of common interfaces to remotely access all domain-specific framework components like Application Modules and View Objects. This encourages a strict logical separation of:

- The Client Tier of an application
- The Application Tier, where business logic is enforced and the data access is coordinated
- The Database Tier where data is queried and stored.

9.2 Deploy Logical Application Tiers in Multiple Physical Configurations

With Oracle Business Components any combination of physical deployment of these three logical layers is possible **without modifying developer-written application code** since the client code works against interfaces to work with the Application Module and any View Objects in it.

Figure 18 below illustrates the scenario where the client remotely accesses the Application Module component, either over CORBA IIOP or through EJB RMI-over-IIOP.

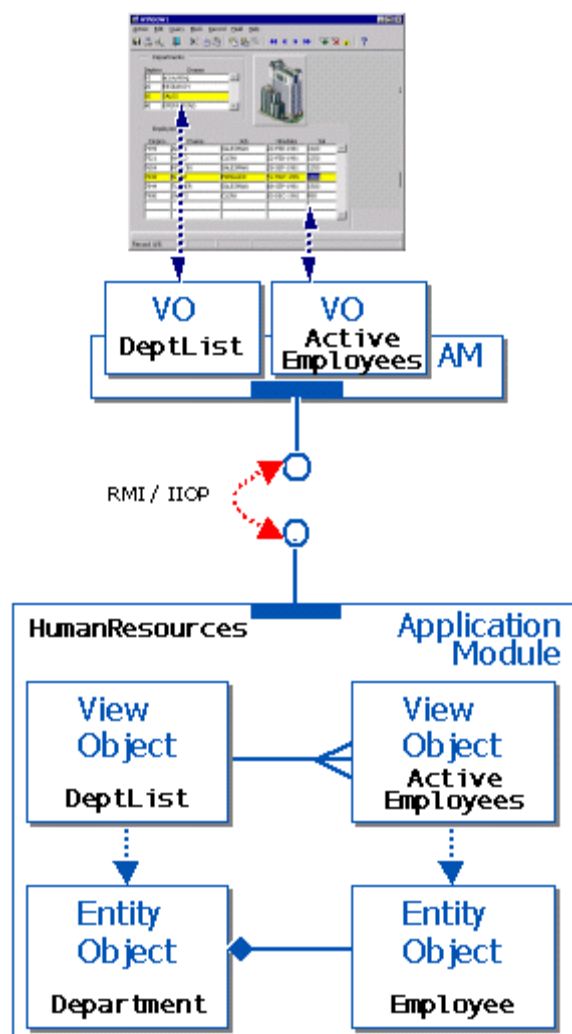


Figure 18: Thin Java UI Client of an Application Module

The figure is the same for any of the following cases where a remote Java application or applet accesses the Application Module deployed as:

- CORBA Server Objects into the Oracle8i database,
- an EJB Session Bean into the Oracle8i database,
- an EJB Session Bean on Oracle Application Server 4.0.8
- an EJB Session Bean on any EJB Server.

If the logical client layer is physically co-located in the same Java VM as the application logic layer (i.e. the Application Module), no remote implementation is required beneath the consistent set of application interfaces used by the client. The application programming model is identical.

Figure 19 below represents the local client case which is identical for the following example architectures:

- A Servlet or JSP working with an instance of an Application Module on any Servlet-compatible web server, including Oracle8i release 8.1.6 and Oracle Application Server 4.0.8.
- A batch program accessing the Application Module's View Object components.

- A remote "fat"-client, which has downloaded the Oracle Business Components logical application layer into the client.

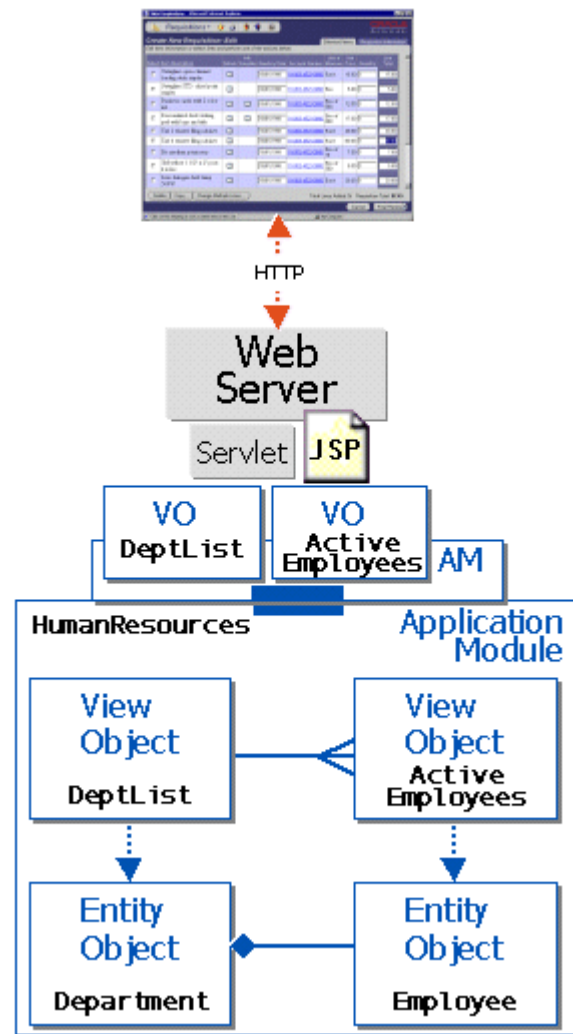


Figure 19: HTML Servlet or JSP Client of an Application Module

Figure 20 summarizes the three typical deployment configurations of an Application Module.

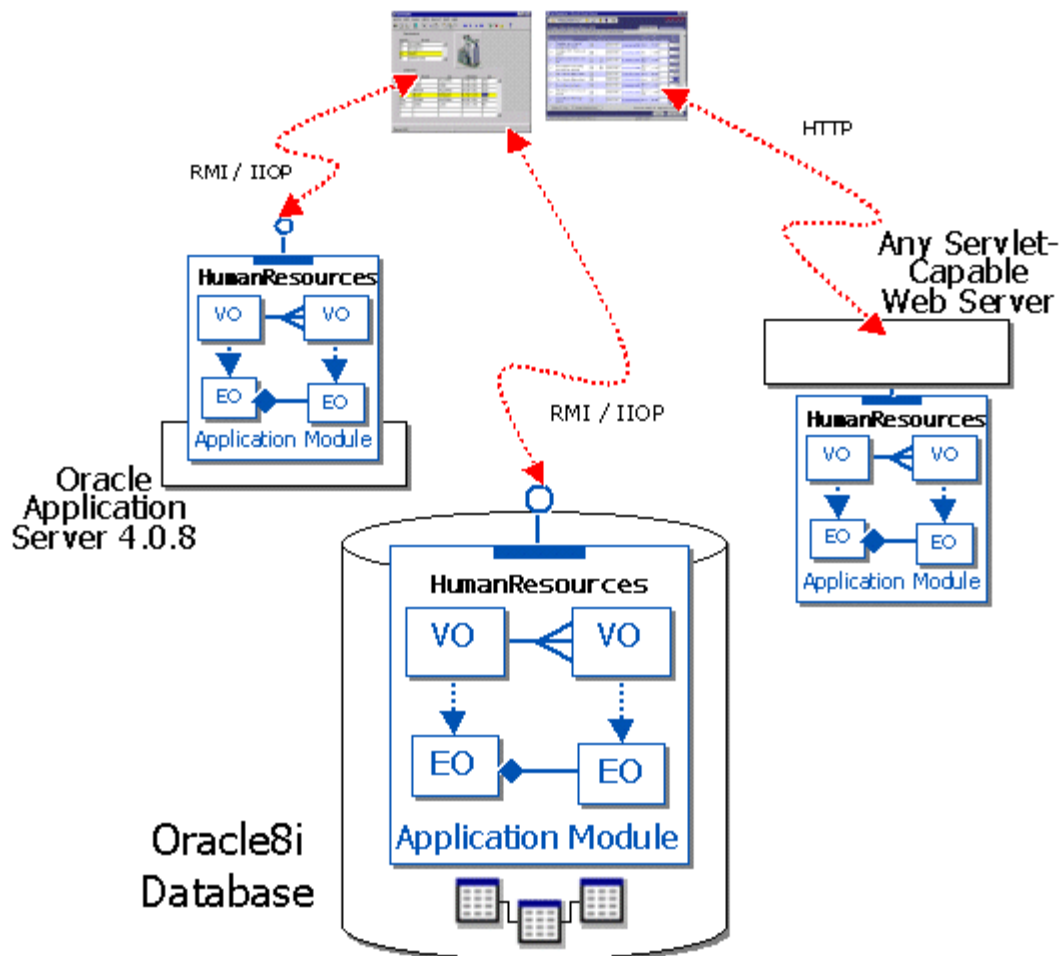


Figure 20: Application Module Typical Deployment Options

9.3 Avoid Typical CORBA and EJB Complexities When Programming Clients

Instead of manually programming against individual remote interfaces which represent all of the different tables in your database as would be the case with an EJB Entity Bean style of remote application development, use Oracle Business Components to deploy an Application Module as an EJB Session Bean, which does all the work for you. The Oracle Business Components Client Service API offers a highly-productive yet simple set of interfaces used to remotely work with the Application Module, the View Objects and View Links inside it. It offers the client-side UI programmers an API that is just like working with JDBC ResultSets when they work with View Objects.

The key difference is that for **no** additional complexity in the programming interface over JDBC ResultSets, View Objects are fully-scrollable, fully-updateable, business-logic-enabled rowsets which automatically work across tiers, can be automatically master-detail coordinated, and effortlessly manage the bulk of the queried data in the Application Tier.

10. Supports Robust, Layered Customizations of Component Code and Metadata

Technical Highlights

- ▶ Extend the functionality of application components delivered without requiring source code modification.
- ▶ Globally substitute delivered component implementations with ones you have extended without requiring or modifying the delivered application.
- ▶ Upgrade applications with confidence that downstream customizations will neither be lost nor need to be manually reapplied.

10.1 Extend Java and XML for Any Component Without Source Code Modifications

Once you've developed and delivered an enterprise application, the next step that typically happens is that the person or company installing the software needs to tailor it to fit their needs.

Oracle Business Components for Java offers a simple, innovative solution to these needs. Oracle Business Components was architected to support the customization of application functionality from day one. All framework components have their Java Implementation Class cleanly separated from their XML Component Definition. This allows the Java Implementation Class for a customized version of any domain-specific component to leverage the Java language's **extend** capability to inherit the base component's Java Implementation Class. This solves part of the problem.

The next part of the customization solution lies in the Oracle Business Components framework's built-in support for the extension of XML Component Definitions. A newly created domain-specific component like an Entity Object, a View Object, an Application Module, etc., can simply extend the metadata of another component declaratively. For example, to extend an **Order** Entity Object from *Figure 3*, use the Oracle JDeveloper IDE to build a new Entity Object which extends from **demo.Order**. Using the XML Component Definition Extension feature, the user only needs to note what's new or different about the **CustomOrder**. The resulting XML Component Definition for the new, extended **CustomOrder** looks like this:

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<!DOCTYPE Entity SYSTEM "jbo_01_01.dtd">

<!--
|
| XML Component Definition for the CustomOrder EO
| which extends the demo.Order EO, adding one
| additional attribute.
|
```

```

+-->
<Entity
  Name="CustomOrder"
  Extends="demo.Order"
  DBObjectName="CUSTOM_ORDTABLE"
  RowClass="demo.CustomOrderImpl" >
  <Attribute
    Name="MyCustomAttributeName"
    Type="java.lang.String"
    ColumnName="MY_CUSTOM_ATTR_COLUMNNAME"
    ColumnType="VARCHAR2"
    SQLType="VARCHAR"
    Precision="30" >
  </Attribute>
</Entity>

```

The Oracle Business Components Entity Object Wizard sets up the `Extends="demo.Order"` metadata attribute automatically, and adds the new or changes metadata features to the XML Component Definition for the customized `CustomOrder`. *Figure 21* shows another example, underscoring the fact that:

- The customized components can "live" in any package that makes sense for the customizer,
- Both the Java Implementation Classes and XML Component Definitions can be extended.
- Any kinds of framework components can be customized in this way.

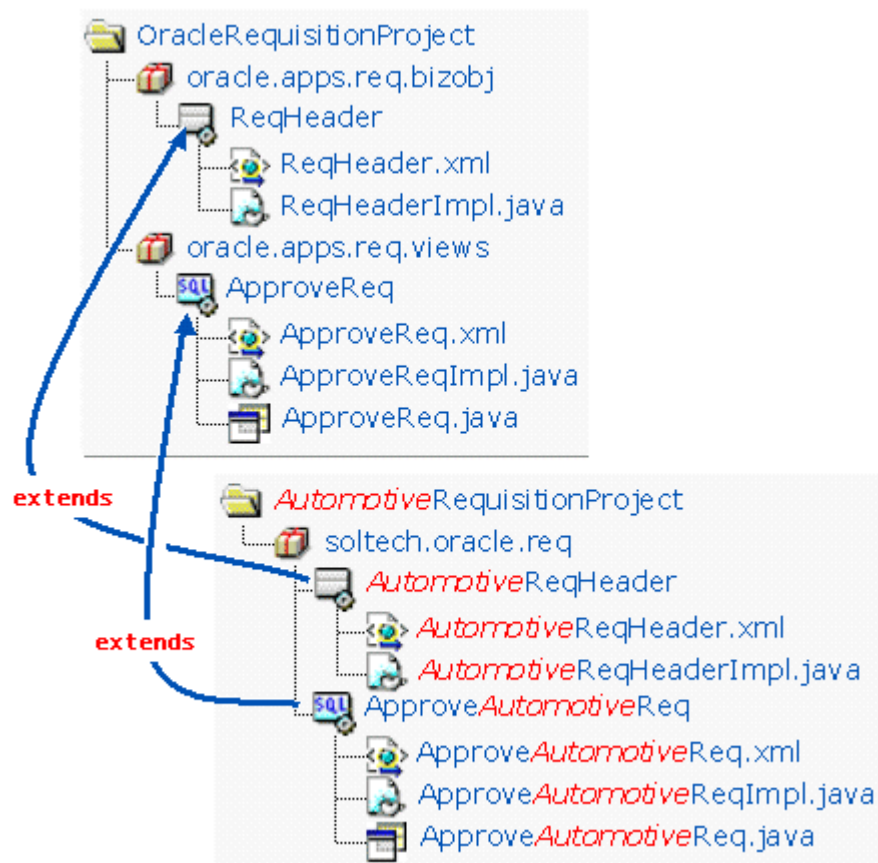


Figure 21: Layered Customization of Oracle Business

Components

10.2 Globally Substitute Extended Components for Those in Original Application

Optionally, the customizer can decide if the customized version of a component should **completely and globally substitute** the component from which it has extended. By making a metadata entry in the Oracle Business Components application's Project File, the customizing developer can assert that `newpackage.CustomOrder` should replace `demo.Order`. At runtime, any request made to work with an instance of `demo.Order` will instead be given an instance of the customized `newpackage.CustomOrder` Entity Object. This optional factory substitution works for all framework components.

The net result is that using a combination of component extension and factory substitution, you can tailor the behavior of any delivered component without modifying its source code (without even **having** the source code for its Java Implementation Class!) and optionally replace the original version of the component with your customized version. If the factory substitution is used, then even the **original** application code which knew *nothing* of your customized component will use the customized version at runtime.

10.3 Drop-in New Versions of Base Components and Extensions Keep Working

The implementation of the XML Component Definition Extension feature in Oracle Business Components is designed to allow new versions of the components which have been subsequently customized to be added, and other components which have made customizations to them will pickup their new features the next time the application is started.

Summary

In summary, **Oracle Business Components for Java** offers a pragmatic approach for developing, deploying, and customizing enterprise-scale applications in Java for delivery on industry-standard server environments like CORBA, EJB, and Java Servlets.

The Oracle Business Components application framework is familiar to Java developers and makes them more productive by handling all of the hard parts of building efficient, distributed business applications, allowing their coding skills to be focussed truly on the **business** at hand, and not on the "plumbing".

Finally, developers taking advantage of **Oracle Business Components for Java** with Oracle JDeveloper will be exploiting the same foundation technology currently being used by Oracle's own Application Development and Consulting divisions for a smooth-slope migration to a component-based Java architecture for enterprise applications.

World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA
<http://www.oracle.com>

1-800-ORACLE1
Fax 650.506.7200

Corporation.
All other company and product names mentioned
are used for identification purposes only, and
may be trademarks of their respective owners
Copyright Oracle Corporation 1999
All Rights Reserved