Bridging the framework modeling and implementation gap

by R. Bunting

The IBM SanFrancisco™ initiative has established a tools strategy to address the complexities of framework development, including repetitive coding tasks, consistent coding style, and an overall compliance with the framework implementation. To address these complexities, the strategy includes an evolutionary approach to cross-tool and cross-tool-provider integration, the ability to address the needs of multiple development audiences, and the existence of multiple development scenarios. Rose SF Bridge, from METEX Systems Inc., implements the principles of this strategy with its integrated set of tools for SanFrancisco application development. This tool set helps application and framework developers to extend the SanFrancisco framework in conjunction with tools for visual modeling and Java™ development.

SanFrancisco* supports an iterative development cycle, as is customary with object-oriented development. Various tools, some from IBM, some from other vendors, simplify the development of SanFrancisco-based applications.[1] The use of visual modeling, a common "best practice" of software design, is supported via Rational Rose**.[2] The visual model is recorded in the Unified Modeling Language (UML).[3] The modeling and subsequent code generation is simplified by the framework models supplied with SanFrancisco and tools including those from METEX contained in Rose SF Bridge,[4] the subject of this paper.

The SanFrancisco Roadmap

The development approach is documented via the *SanFrancisco Roadmap*,[5] which guides developers through the development cycle and provides a set of activities and standard templates to document a domain's business processes, tasks, and scenarios. Throughout the development cycle, the activity of *mapping* is performed, where the development artifacts are compared with UML models that document the SanFrancisco frameworks. The development cycle steps are:

- Collect and document the requirements
- Analyze the requirements
- Design the code
- Generate and test the code

For gathering requirements, the roadmap suggests using either process modeling or use case modeling. Both methods are supported by Rational Rose.

During the analysis step, more details are supplied about users' activity and the business logic required. An analysis object model is created that identifies the domain objects and their static relationships. Scenarios are documented via textual descriptions and analysis-level object interaction diagrams.

During the design step, implementation decisions are added to the analysis results. The analysis model is extended with design details, and the application scenarios and the user interactions are refined with im-

plementation details. A detailed description of each application scenario and a complete design model can be created and managed in Rational Rose. The design model is then used as input to a code generator.

The code generator uses the static model, along with the framework rules and design patterns, to create the SanFrancisco objects, eliminating much of the repetitive nature of framework-based coding.

What is Rose SF Bridge?

Rose SF Bridge assists SanFrancisco developers in new development, or in migrating existing SanFrancisco-based models to Rose SF Bridge-based models. The product includes the SF Code Generation Wizard, the SF Modeling Wizard, and the SF Model Upgrade Tool.

The Rose SF Bridge is designed to work with the Rational Rose visual modeling tool to assist in the creation and refinement of UML models. Rational Rose allows developers to define and communicate a software architecture. A clearly defined architecture improves communication among team members, maps business processes to software architecture, and makes critical design decisions explicit. The Rose SF Bridge links the visual models to source code, providing an integrated development environment for SanFrancisco. Currently code generation support is present for both IBM's VisualAge* for Java**[6] and Inprise/Borland's JBuilder.[7] Rose SF Bridge is currently available for SanFrancisco versions 1.2, 1.3, and 1.4, and METEX Systems will continue to update the code generation rules of Rose SF Bridge as the SanFrancisco project develops.

Why use Rose SF Bridge?

Because of the complexity associated with a large framework, a component-based visual modeling tool is needed. Such tools capture the structure and behavior of architectures and components, show how the elements of the system fit together, hide or expose details as appropriate for the task, maintain consistency between a design and its implementation, and promote unambiguous communication. Rose SF Bridge is an integrated toolkit for SanFrancisco development and bridges SanFrancisco's application framework, connecting the UML model to generated Java code. Using the Rose SF Bridge simplifies the SanFrancisco-based development process. The Rose SF Bridge, along with Rational Rose, assists the developer in overcoming the complexities of object-oriented development, thus simplifying SanFrancisco-based development.

How does Rose SF Bridge work?

The Rose SF Bridge is integrated with Rational Rose[8] and with both VisualAge for Java and JBuilder. The developer interacts with a UML representation of the SanFrancisco framework. An internal rules engine supports numerous design and code patterns specific to the SanFrancisco programming model. The rules engine can rapidly create and extend components that work within the SanFrancisco frameworks by generating implementation code based on a high-level design model.

Compared to the SanFrancisco code generator, the Rose SF Bridge code generator is more tightly integrated with Rational Rose. In addition, it can perform an "assisted merge" during code generation, which allows design changes made in the Rose model to be propagated to existing source code without loss of existing method bodies. This eliminates the need to manually merge newly generated code with existing code. As the application design evolves, the model and code remain consistent.

The Rose SF Bridge Modeling Wizard

The Rose SF Bridge captures semantic information about classes used and extended within the SanFrancisco framework. The modeling wizard component encourages SanFrancisco-compliant design and automates the setting of code generation properties based on the underlying framework programming conventions. The workflows of the modeling wizard include (1) creating a new SanFrancisco-compliant class, (2) extending or modifying an existing class, and (3) creating an aggregation (relationship) between two existing classes. The dialogs of the wizard will vary based on the chosen workflow.

Our example shows the new class creation workflow. The wizard is launched without an existing class selected, and the "Class Type Selection" screen is displayed. (See Figure 1.)

The user is prompted to select one of the basic SanFrancisco classes: Entity, Dependent, Command, or Controller.

Here "Create Entity Subclass" is selected and the "Next" button pressed to advance to the "Class Declaration" screen (Figure 2).

Figure 1 The Class Type Selection window of the Rose SF Bridge Modeling Wizard

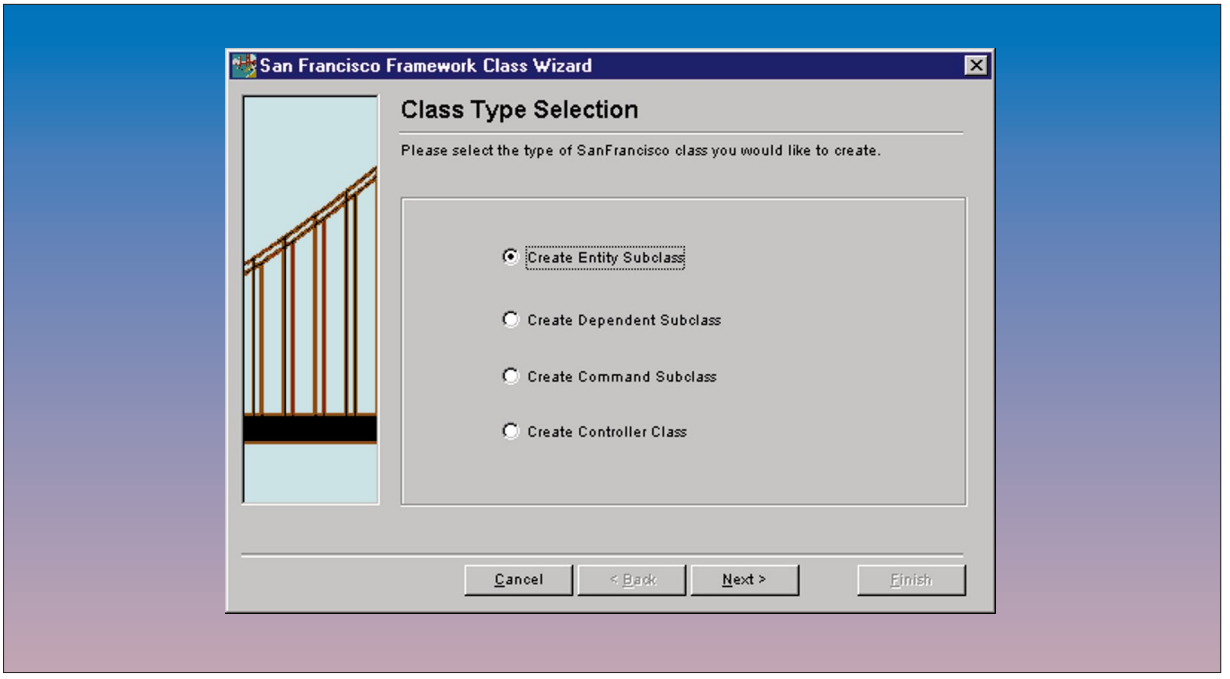


Figure 2 Each class is named, has a custom pluralization, and resides within some package.

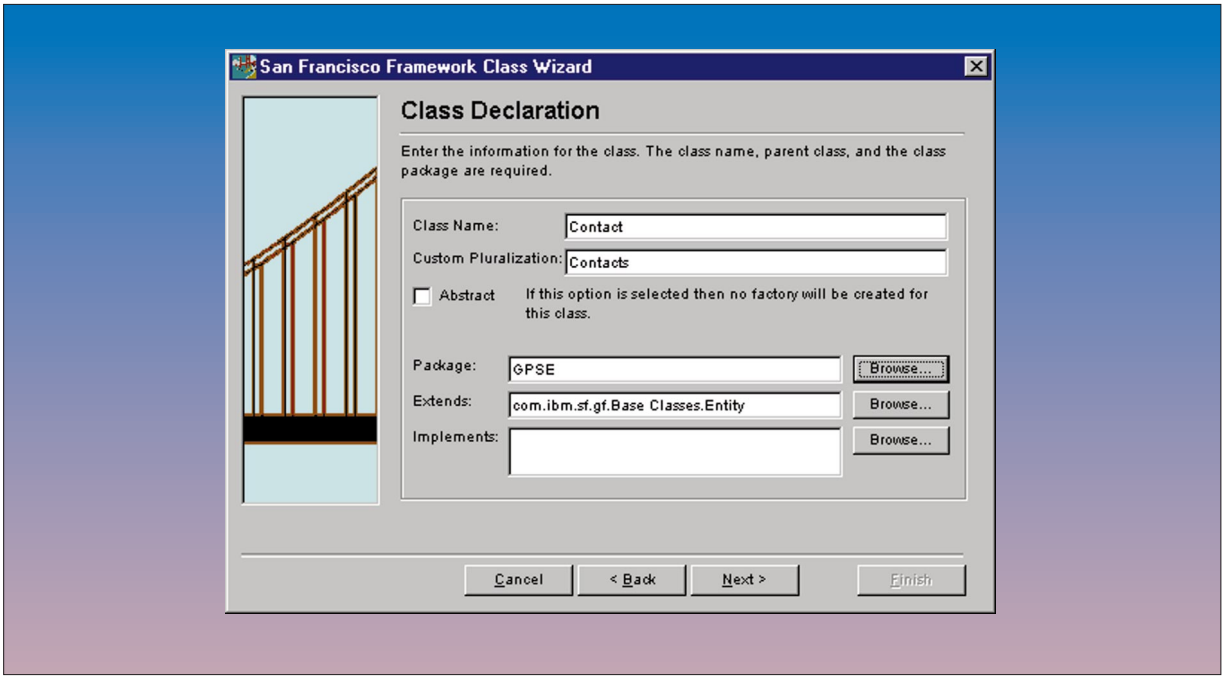
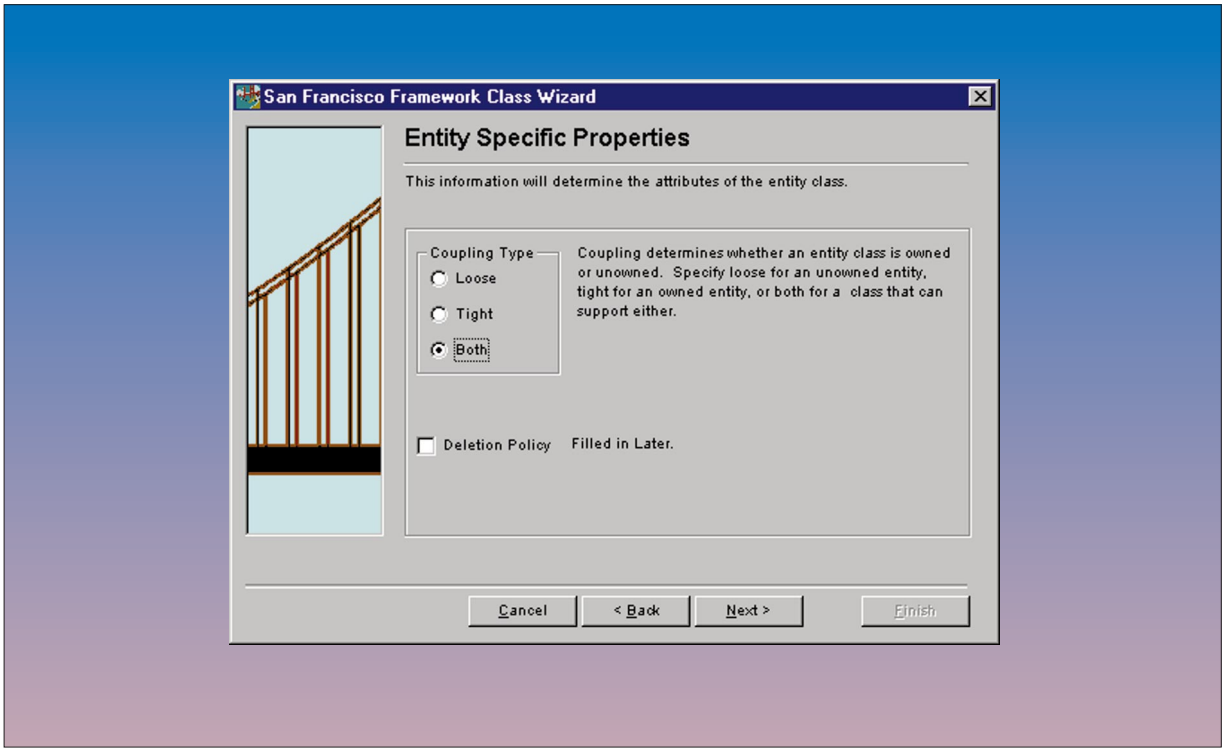


Figure 3 Properties specific to the SanFrancisco Entity class and its subclass



Here, properties common to all SanFrancisco classes are entered including:

- Class name
- Custom pluralization. The default plural form (class name + "s") will be filled in automatically. If the class has a nonstandard plural (e.g., company → companies), the plural can be entered here. Pluralizations are used for collections of objects of the class.
- Package. The user can type in the package name or select a package via the "Browse . . ." button.
- Extends. This field will be automatically filled in with the name of the selected class. The browser can be fitted with a filter to browse only classes that extend directly or indirectly from the selected class. This allows the user to extend indirectly from the selected class.
- Implements. The user can type in, or select via the "Browse . . ." button, the interfaces that the class will implement.

Once these properties have been set, the "Next" button is pressed to proceed to the next screen, which contains type-specific information. These type-specific properties vary depending on the basic SanFrancisco class originally selected. Figure 3 shows properties specific to the Entity basic class.

Properties are also set for the Command and Controller basic classes. There are no type-specific properties associated with the Dependent basic class. If the user is creating or modifying a class derived directly or indirectly from the Dependent class, the user will be immediately presented with the "SanFrancisco Documentation" screen. (See Figure 4.) Here values are entered for: the version number, the purpose, pre- and post-conditions, and any comments.

When "Next" is pressed from the "Documentation" screen, the wizard advances to the "Summary" screen. This screen (Figure 5) shows all properties—generic and type-specific—for review prior to class creation. If changes to the class's properties are required, the "Back" button is pressed; otherwise the "Finish" button is pressed to complete creation of the class. Figure 6 shows the new class "Contact" in the Rose model. The developer is now responsible

Figure 4 Documentation window

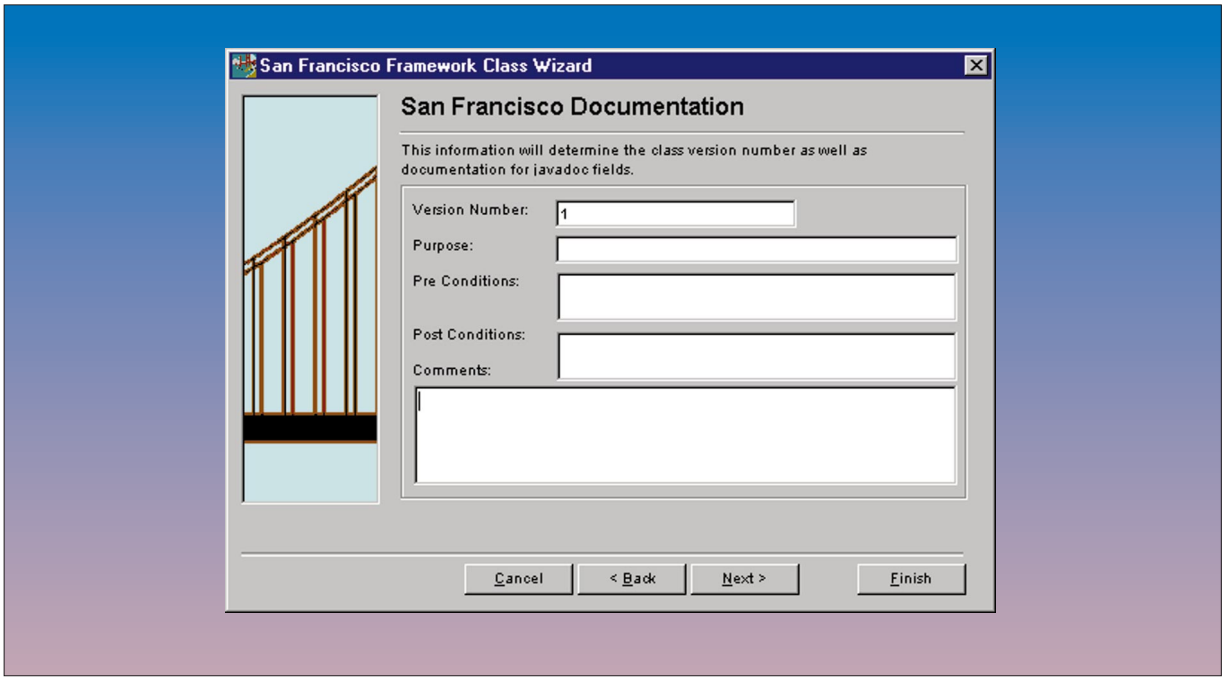


Figure 5 Details of the class to be created

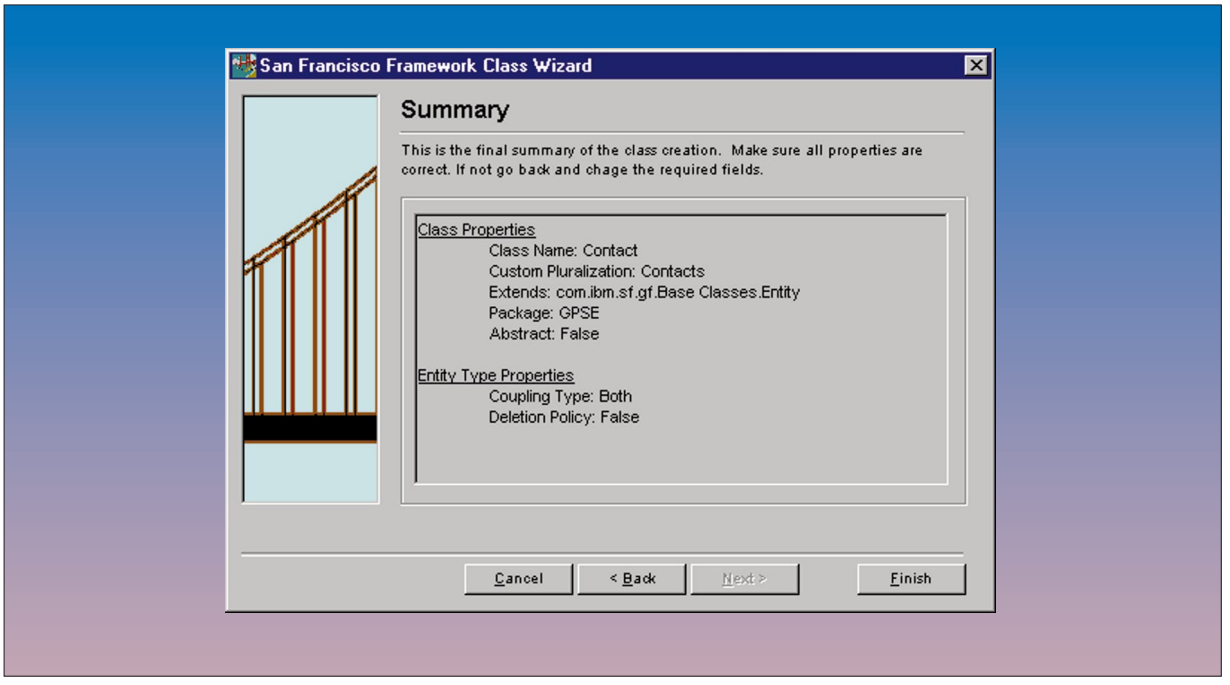
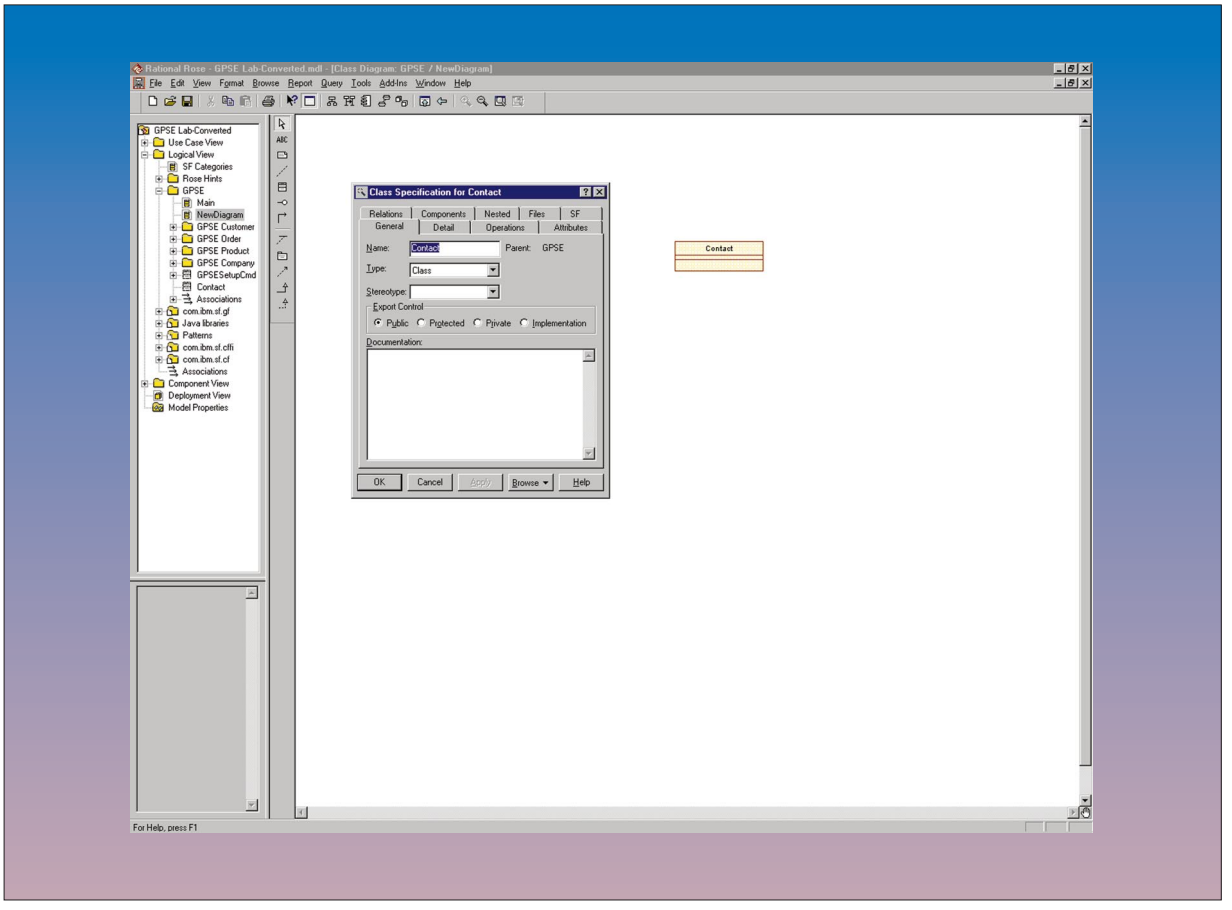


Figure 6 A class created with the window Modeling Wizard and the corresponding properties in the Class Specification window



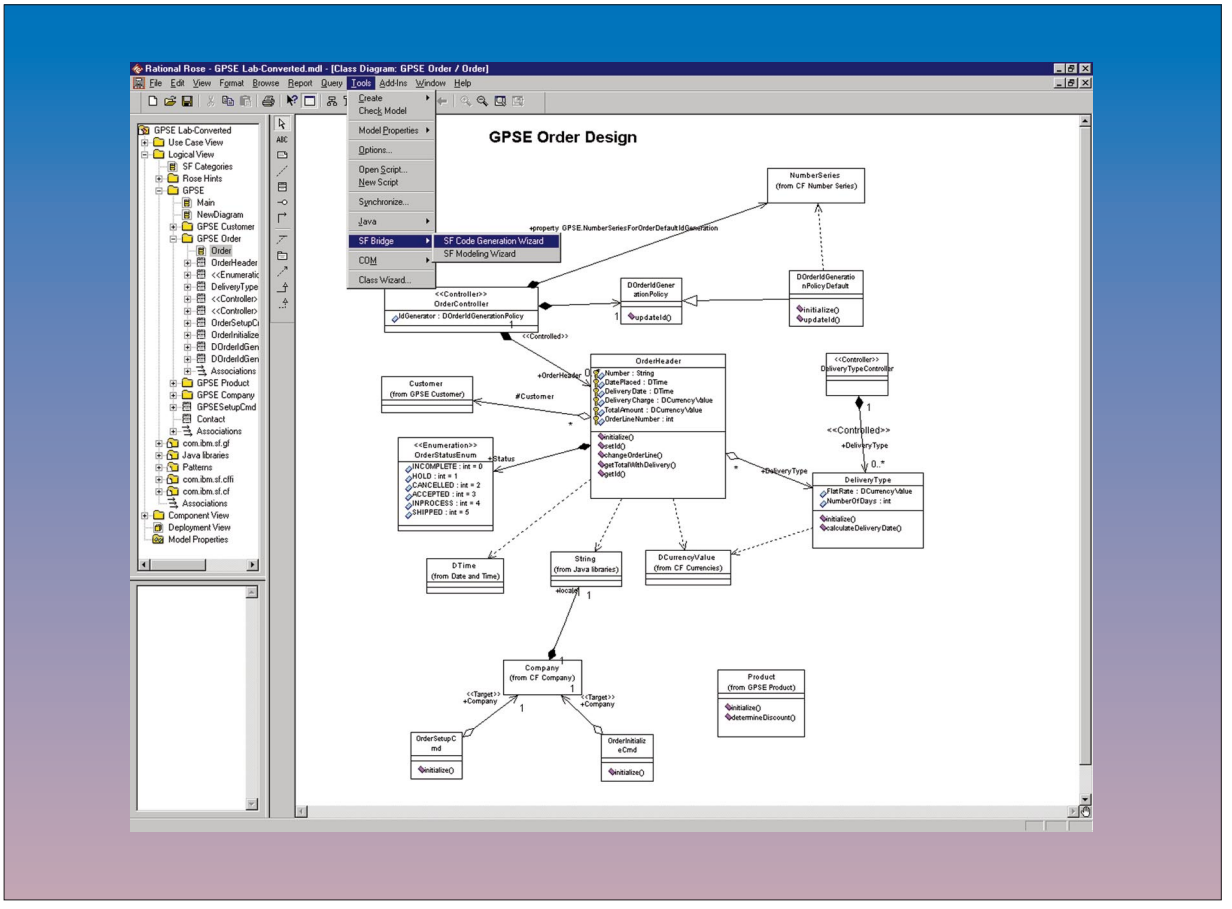
for completing the class design, adding appropriate attributes and operations to capture its business logic. When design is complete, the class is ready for code generation, as shown for the DeliveryType class in the next example.

The Rose SF Bridge Code Generation Wizard

Rose SF Bridge code generation is based on the techniques[9] of traditional language code generators; however, these techniques of mapping a modeling notation to the object model of the target language have been enhanced to include flexible rule sets and well-known design patterns supported in the SanFrancisco framework. Rational Rose is used to capture, and record in UML, semantic information about classes used and extended within the SanFrancisco framework. Using this information, the Rose SF Bridge Code Generation Wizard follows rules derived from the SanFrancisco programming model to generate implementation code specific to the foundation layer of the framework. This code generation shields the complexity of the framework and generates a significant amount of implementation code that allows the application to tie in to the foundation layer.

For this example, we show code generation for the DeliveryType class from a sample application, "Get Physical Sports Equipment" (GPSE). (See Figure 7.) This class is the result of the application of detailed class design. As the DeliveryType class is a subclass of Entity, the SanFrancisco programming model dictates the Abstract Factory design pattern will be ap-

Figure 7 The Rose SF Bridge code generator, launched from the Tools menu of Rational Rose



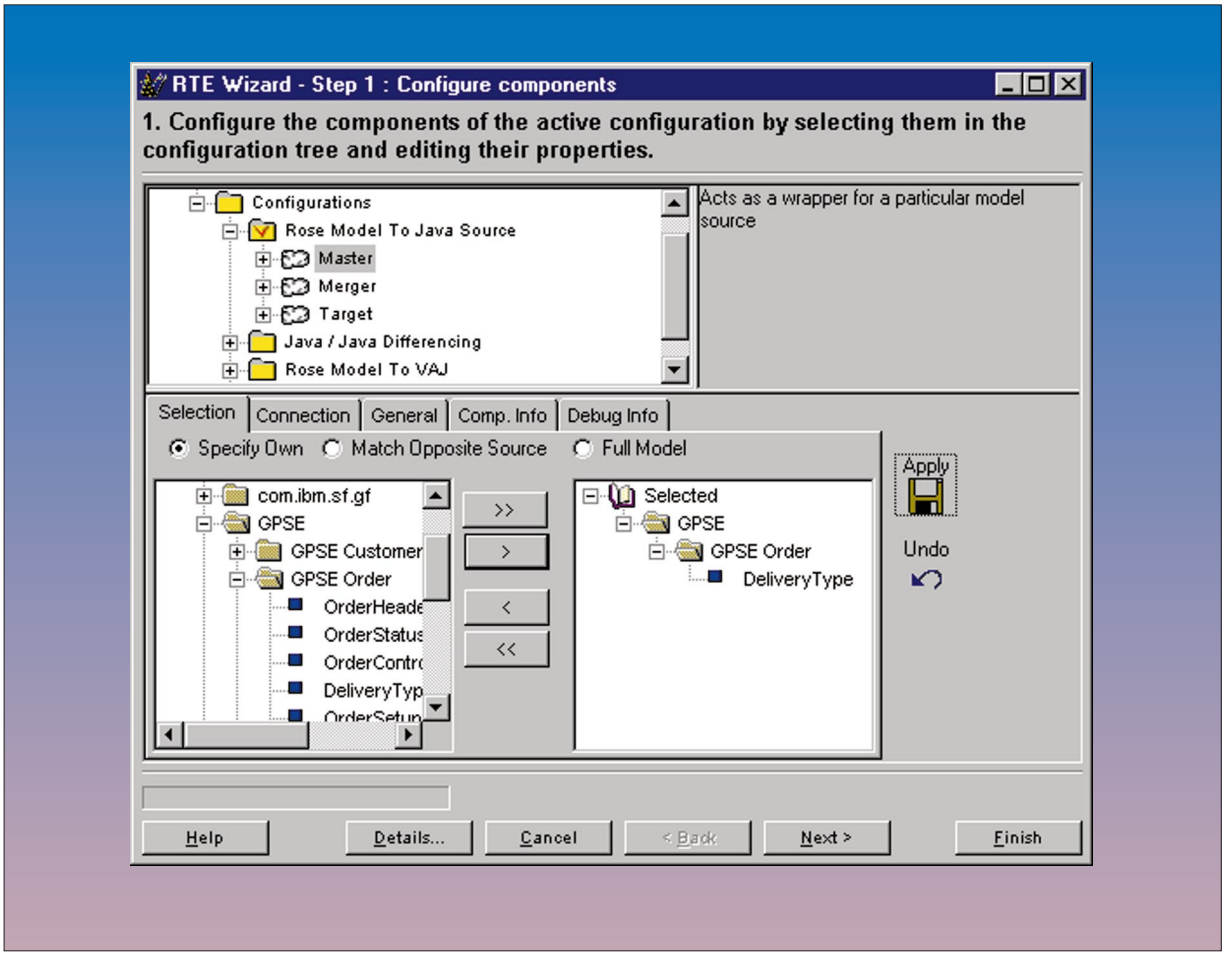
plied to the specification class, resulting in three classes in the implementation model, i.e., the Java source code:

- *Interface class*: Java interface retaining the name of the specification class (Delivery Type) and extending the interface of its superclass (Entity)
- *Implementation class*: Java class with the suffix "Impl" appended to the specification class name (DeliveryTypeImpl) and extending the implementation class (of Entity). The following mandatory operations are declared on the implementation class: toString, destroy, internalizeFromStream, and externalizeToStream.
- *Factory class*: Java class with the suffix "Factory" appended to the specification class name (DeliveryTypeFactory).

Next, creation and initialization logic is added to these three classes. One or more initialize methods on the interface class, and create methods on the factory class, are created for each initialize operation present on the specification class.

If loose coupling is selected on the specification class, the parameters to the initialize method on the interface class will match those found on the initialize or create method on the specification class. If the coupling property on the specification class is set to either "tight" or "both," the first parameter on the specification initialize or create method is expected to be the owning class for the tight coupling case. If the coupling property is set to "both," two initialize methods will be created on the interface class for

Figure 8 DeliveryType class selected for generation



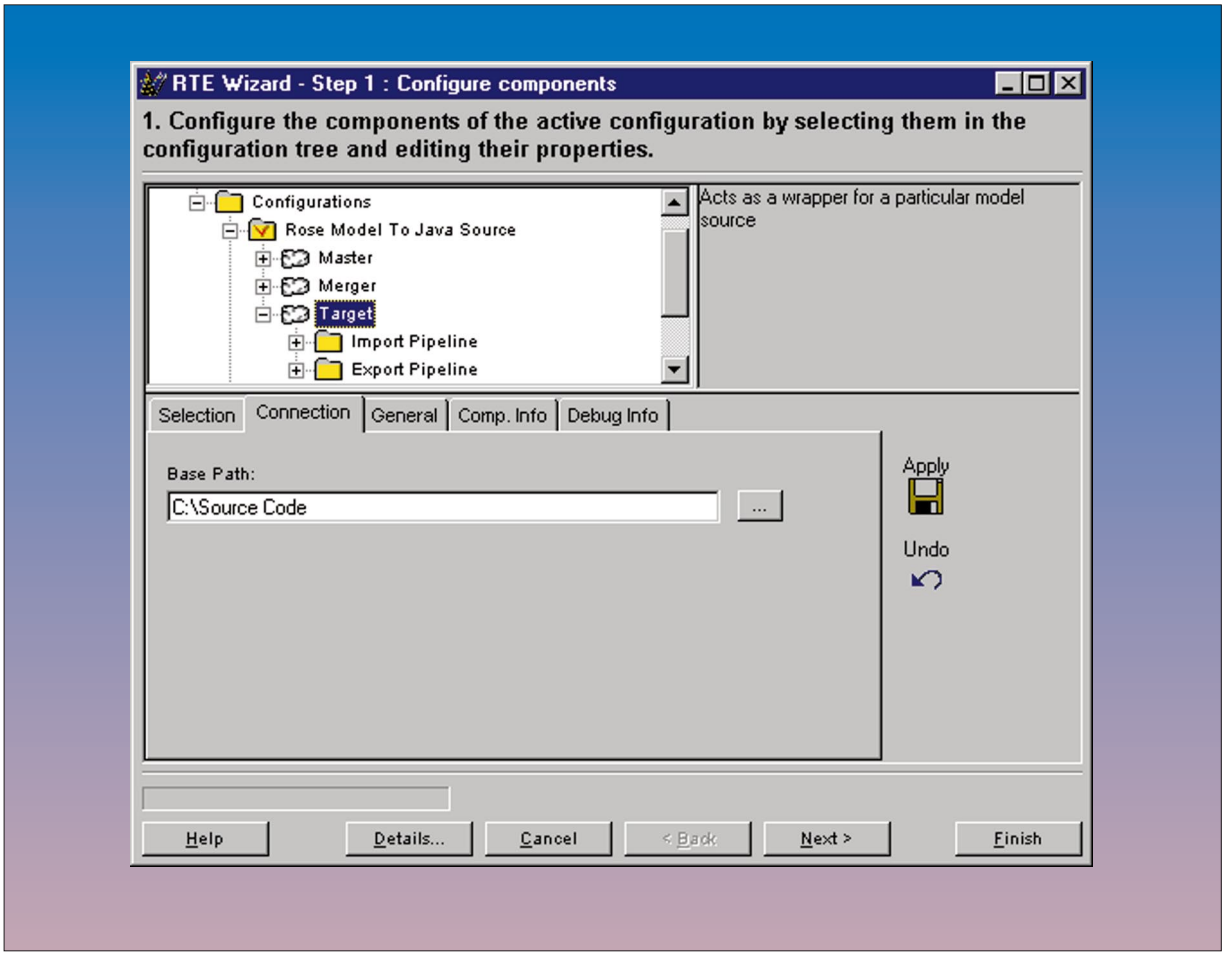
each initialize method present on the specification class.

For the implementation class, an empty constructor, an uninitialize method, and an update method are declared.

If the factory class exists, static create operations are declared for every initialize operation on the specification class. If loose coupling is selected on the specification class, one static create operation is declared. If tight coupling is selected, two static create operations are declared. If "both" is selected, all three static create operations are declared. In addition, if tight coupling is selected, an abstract special factory operation is declared for every initialize operation on the specification class.

Figure 8 shows the Code Generation Wizard with the "Rose Model to Java Source" configuration selected as active. Configurations group together execution "pipelines" of components. For example, generating Java implementations as Java source code files requires a different set of target components than generating Java classes for the VisualAge for Java repository. With a configuration set as active, code generation proceeds by selecting the model elements from the tree view of the master component properties in the "Configure Components" window and clicking "Apply." Next, the target connection

Figure 9 Target Connection set to a directory where the implementation source will reside



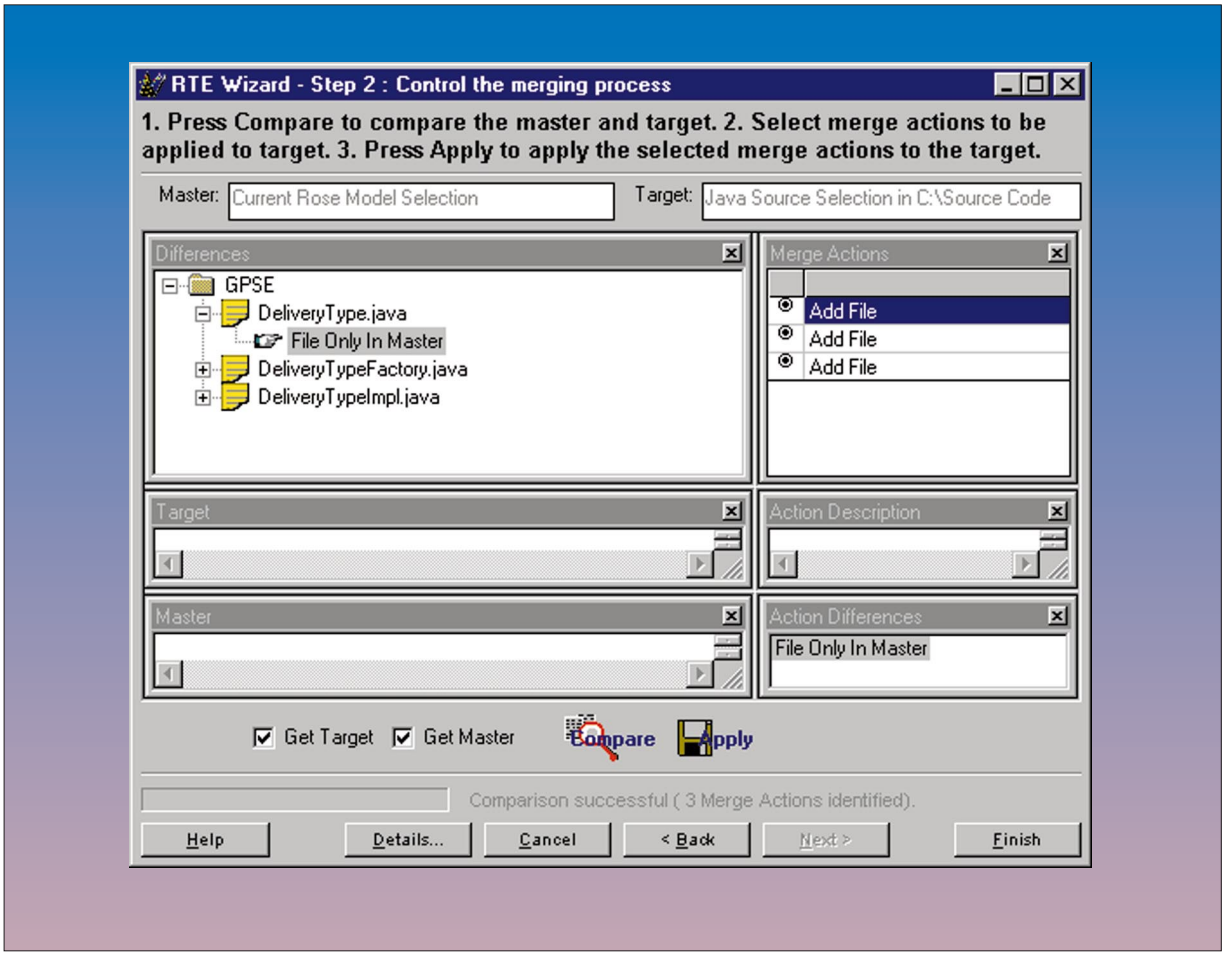
(i.e., base path directory) is set in the "Connection" tab as shown in Figure 9 and once again, "Apply" is pressed.

With the compulsory options set, the "Next" button is pressed to advance to the "Step 2" dialog where the generation is executed and monitored. The master is the current Rose model selection and the target is Java source selection in C:\Source Code. Pressing "Compare" begins the process of exporting the Rose model selection, interpreting its code generation properties, applying SanFrancisco programming conventions, and exporting it to the Java source code format. Then, the existing source code (if any) is imported from the target directory and a preliminary merge analysis takes place, where differences are identified and merge actions are proposed.

Figure 10 shows this step of the code generation process. Here we see that the generated files do not exist in the target model and hence the proposed merge action is to add the files. Pressing "Apply" results in the application of the merge actions. Check marks indicate that the merge actions have been applied successfully to the target model (i.e., the Java source code).

The Appendix contains the code that was generated from this activity. Notice that three classes have been generated: one for the interface, one for the implementation, and one for creation (called the factory). Also, notice that actual method bodies are generated and not just method signatures or skeletons. There is still some code required to complete details of the business logic that extends the framework;

Figure 10 Differences highlighted (left) and proposed merge actions displayed (right)



however, a significant savings in implementation effort was achieved using the code generator.

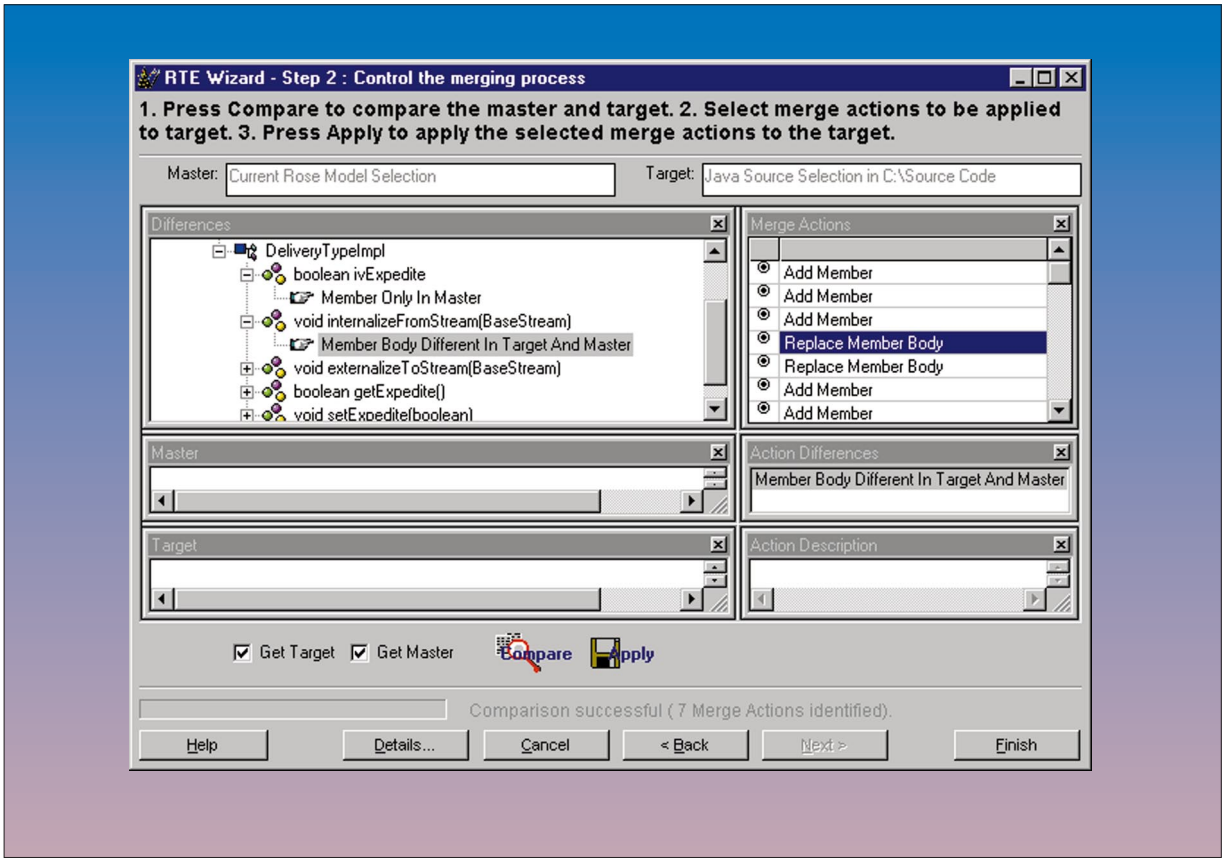
Now that source code exists in the directory specified by the target model, for subsequent generation, as the design evolves, there may be more complex merge actions. Figure 11 shows the generation process once an additional attribute has been added for the DeliveryType class. Notice that the merge actions are more complex than just adding an attribute, because this attribute is referenced in a number of locations within the target implementation; in particular the internalizeFromStream method difference is highlighted. Once again, the value of the code generator is obvious and the power of incremental generation is evident. The alternative would be for a developer to manually search source code listings to ensure the full impact of the additional attribute is realized.

VisualAge for Java integration

In the previous section the "Rose Model to Java Source" configuration was examined. As an alternative, a Rose model may be translated to the VisualAge for Java repository. For consistency and ease of use, the steps are very similar, with the exception of the target connection. Figure 12 shows this connection setting.

In this case, the target identifies the Java servlet that brokers communication between Rose SF Bridge and

Figure 11 Master and target differences and proposed merge actions



the VisualAge for Java repository. The launch of this servlet is shown in Figure 13, which displays VisualAge and the menu option to run the HTTP VA Integrator. With the servlet, the user does not have to perform the typical manual import/export steps of transferring the source into the VisualAge for Java repository.

The model upgrade tool

As some developers may want to modify code generated by the original IBM SanFrancisco code generator, METEX worked with IBM to provide a tool for those developers. The SanFrancisco Model Upgrade Tool performs the conversion necessary for models originally designed for the IBM SanFrancisco code generator to be processed by METEX's Rose SF Bridge.

The upgrade is performed in three steps:

1. Process "#directives" included in the documentation field of the model elements. These were used instead of the more standard code generation properties used by Rose SF Bridge.
2. Analyze and convert "create" and "initialize" operations. "Create" operations are not necessary when using the full features of Rational Rose, such as the abstract property checkbox.
3. If desired, remove #directives.

Summary

As many framework developers and users have observed, the ability to quickly understand the architecture and mechanisms within a framework, and then apply these elements while developing with a framework, is critical. It is essential that a proven development process and a set of mature tools be applied in a holistic manner. We have seen how Rose SF Bridge integrates with the Rational Rose visual

Figure 12 Setting the target connection for the VisualAge for Java repository

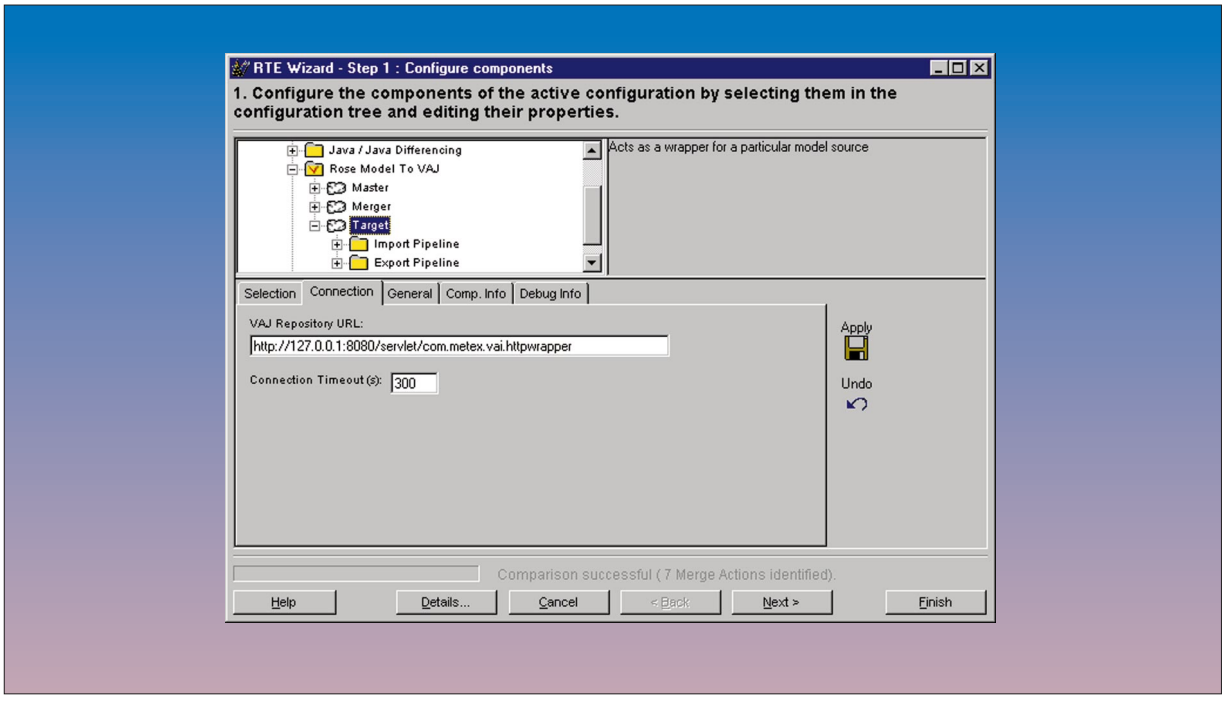
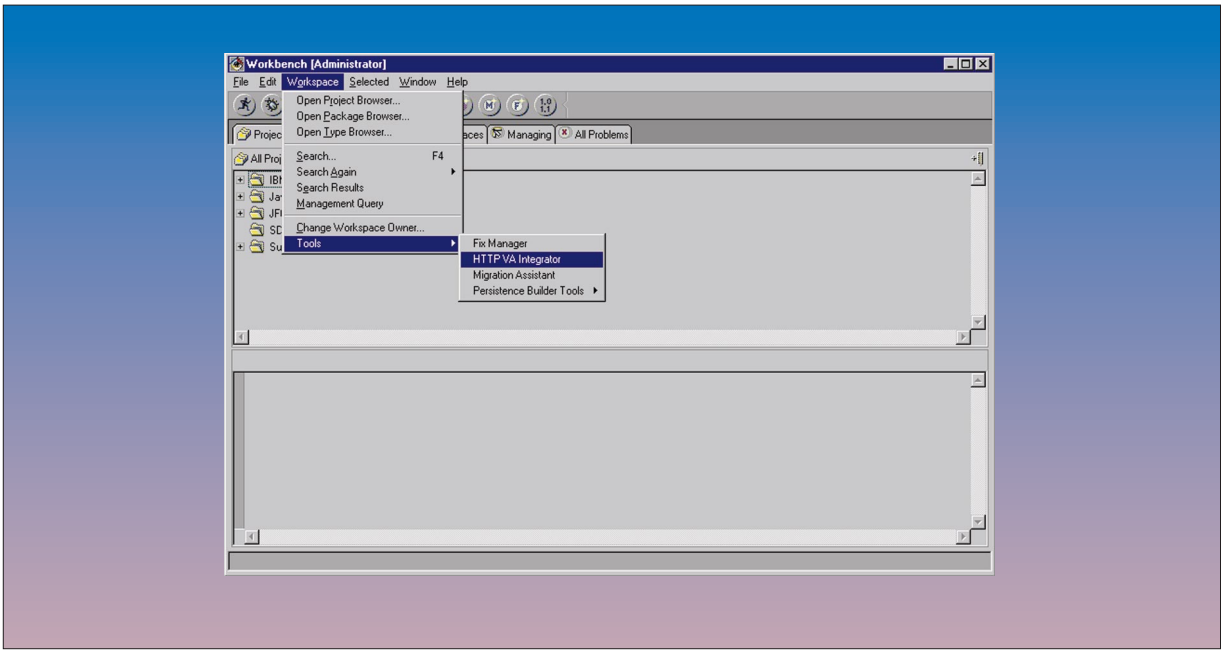


Figure 13 VisualAge for Java: running the HTTP VA Integrator



modeling tool to assist in the creation of UML models that enhance development documentation and provide input for code generation, and also how to reduce development time by applying a set of modeling and code generation wizards. Visual modeling allows developers to define and communicate a software architecture, resulting in: accelerated development, by improved communication among various team members; improved quality, by mapping business processes to software architecture; and increased visibility and predictability, by making critical design decisions explicit visually. As part of the complete set of tools, Rose SF Bridge supplements basic visual modeling tasks by providing assistance in defining and modeling SanFrancisco-specific classes and generating code for these classes. This tool set assists the developer in overcoming complexities of object-oriented and framework development through a series of wizards, builders, and design and code generation rules aimed at simplifying SanFrancisco-based development.

Appendix

This code was generated by the Rose SF Bridge code generator.

DeliveryType interface.

DeliveryType.java

```
package GPSE;

import com.ibm.sf.gf.*;
import com.ibm.sf.cf.*;

/**
 * <TT>
 * <BR><B>Purpose:</B>
 * <BR><B>Description:</B>
 * <BR><B>Note:</B> This documentation has been automatically
 * generated.
 * </TT>
 * @version 1.3.4
 * @since JDK1.0
 */
public interface DeliveryType extends DescribableDynamicEntity {
                /**
                 * The fully qualified name of this Interface class
                 *
                 * Note: This documentation has been automatically generated.
                 */
                public static final String INTERFACE_NAME=
"GPSE.DeliveryType";
                /**
                 * Gets the attribute
                 *
                 * <BR><B>Assumptions:</B>
                 * <BR><B>Note:</B> This documentation has been
automatically
                 * generated.
                 * @return DCurrencyValue
                 * @exception com.ibm.sf.gf.SFException
                 * <BR><B>Result:</B>
                 * <BR><B>PreConditions:</B>
                 * <BR><B>PostConditions:</B>
                 */
                public DCurrencyValue getFlatRate() throws
com.ibm.sf.gf.SFException;
                /**
                 * Sets the attribute
                 *
                 * <BR><B>Assumptions:</B>
                 * <BR><B>Note:</B> This documentation has been
automatically
                 * generated.
                 * @param DCurrencyValue newFlatRate <I>(Mandatory)</I>
                 * @return void
                 * @exception com.ibm.sf.gf.SFException
                 * <BR><B>Result:</B>
                 * <BR><B>PreConditions:</B>
                 * <BR><B>PostConditions:</B>
                 */
                public void setFlatRate(DCurrencyValue newFlatRate) throws
com.ibm.sf.gf.SFException;
                /**
                 * Gets the attribute directly
                 *
                 * <BR><B>Assumptions:</B>
                 * <BR><B>Note:</B> This documentation has been
automatically
                 * generated.
                 * @return DCurrencyValue
                 * @exception com.ibm.sf.gf.SFException
                 * <BR><B>Result:</B>
                 * <BR><B>PreConditions:</B>
                 * <BR><B>PostConditions:</B>
                 * <BR><B>RESTRICTED</B> : This method is NOT for client
use
                 */
                public DCurrencyValue getFlatRateForRestrictedUse() throws
com.ibm.sf.gf.SFException;
                /**
                 * Gets the attribute
                 *
                 * <BR><B>Assumptions:</B>
                 * <BR><B>Note:</B> This documentation has been
automatically
                 * generated.
                 * @return int
                 * @exception com.ibm.sf.gf.SFException
                 * <BR><B>Result:</B>
                 * <BR><B>PreConditions:</B>
                 * <BR><B>PostConditions:</B>
                 */
                public int getNumberOfDays() throws com.ibm.sf.gf.SFException;
                /**
                 * Sets the attribute
                 *
                 * <BR><B>Assumptions:</B>
                 * <BR><B>Note:</B> This documentation has been
automatically
                 * generated.
                 * @param int newNumberOfDays <I>(Mandatory)</I>
                 * @return void
                 * @exception com.ibm.sf.gf.SFException
                 * <BR><B>Result:</B>
                 * <BR><B>PreConditions:</B>
                 * <BR><B>PostConditions:</B>
                 */
                public void setNumberOfDays(int newNumberOfDays) throws
com.ibm.sf.gf.SFException;
                /**
                 * Gets the attribute
                 *
                 * <BR><B>Assumptions:</B>
                 * <BR><B>Note:</B> This documentation has been
automatically
                 * generated.
                 * @return boolean
                 * @exception com.ibm.sf.gf.SFException
                 * <BR><B>Result:</B>
                 * <BR><B>PreConditions:</B>
                 * <BR><B>PostConditions:</B>
                 */
                public boolean getExpedite() throws com.ibm.sf.gf.SFException;
                /**
                 * Sets the attribute
                 *
                 * <BR><B>Assumptions:</B>
                 * <BR><B>Note:</B> This documentation has been
automatically
                 * generated.
                 * @param boolean newExpedite <I>(Mandatory)</I>
                 * @return void
                 * @exception com.ibm.sf.gf.SFException
                 * <BR><B>Result:</B>
                 * <BR><B>PreConditions:</B>
                 * <BR><B>PostConditions:</B>
                 */
                public void setExpedite(boolean newExpedite) throws
com.ibm.sf.gf.SFException;
                /**
                 *
                 * <BR><B>Assumptions:</B>
```

```
     * <BR><B>Note:</B>
     * @param DeliveryTypeController owner <I>(Mandatory)</I>
     * @param DescriptiveInformation description
<I>(Mandatory)</I>
     * @param DCurrencyValue initFlatRate <I>(Mandatory)</I>
     * @param int initNumberOfDays <I>(Mandatory)</I>
     * @return void
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     * <BR><B>RESTRICTED</B> : This method is NOT for client
use
     */
    public void initialize(DeliveryTypeController owner,
DescriptiveInformation description, DCurrencyValue initFlatRate, int
initNumberOfDays) throws com.ibm.sf.gf.SFException;
    /**
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B>
     * @param DTime placeDate <I>(Mandatory)</I>
     * @return DTime
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public DTime calculateDeliveryDate(DTime placeDate);
}
```

DeliveryType implementation.

DeliveryTypeImpl.java

```
package GPSE;

import com.ibm.sf.gf.*;
import com.ibm.sf.cf.*;

/**
 * <TT>
 * <BR><B>Purpose:</B>
 * <BR><B>Description:</B>
 * <BR><B>Note:</B> This documentation has been automatically
 * generated.
 * </TT>
 * @version 1.3.4
 * @since JDK1.0
 */
public class DeliveryTypeImpl extends DescribableDynamicEntityImpl
implements DeliveryType. Distinguishable {
    /**
     * The version number of this class
     *
     * Note: This documentation has been automatically generated.
     */
    static final int versionNumber= 1;
    /**
     * The fully qualified name of this Implementation class
     *
     * Note: This documentation has been automatically generated.
     */
    public state final String IMPLEMENTATION_NAME=
"GPSE.DeliveryTypeImpl";
    protected DCurrencyValue ivFlatRate;
    protected int ivNumberOfDays;
    protected boolean ivExpedite;
    /**
     * Gets the attribute
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
automatically
     * generated.
     * @return DCurrencyValue
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public DCurrencyValue getFlatRate() throws
com.ibm.sf.gf.SFException {
                return (DCurrencyValue)
Global.factory().copyDependent(null, ivFlatRate);
    }
    /**
     * Sets the attribute
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
automatically
     * generated.
     * @param DCurrencyValue newFlatRate <I>(Mandatory)</I>
     * @return void
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public void setFlatRate(DCurrencyValue newFlatRate) throws
com.ibm.sf.gf.SFException {
                setDirty();
                ivFlatRate = (DCurrencyValue)
Helper.setDependentToDependent(ivFlatRate, newFlatRate, this);
    }
    /**
     * Gets the attribute directly
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
automatically
     * generated.
     * @return DCurrencyValue
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     * <BR><B>RESTRICTED</B> : This method is NOT for client
use
     */
    public DCurrencyValue getFlatRateForRestrictedUse() throws
com.ibm.sf.gf.SFException {
                return (ivFlatRate);
    }
    /**
     * Gets the attribute
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
automatically
     * generated.
     * @return int
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public int getNumberOfDays() throws com.ibm.sf.gf.SFException
{
                return (ivNumberOfDays);
    }
    /**
     * Sets the attribute
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
automatically
     * generated.
     * @param int newNumberOfDays <I>(Mandatory)</I>
     * @return void
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public void setNumberOfDays(int newNumberOfDays) throws
com.ibm.sf.gf.SFException {
                setDirty();
                ivNumberOfDays = newNumberOfDays;
    }
    /**
     * Gets the attribute
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
automatically
     * generated.
     * @return boolean
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public boolean getExpedite() throws com.ibm.sf.gf.SFException {
                return (ivExpedite);
    }
    /**
     * Sets the attribute
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
automatically
```

```
     * generated.
     * @param boolean newExpedite <I>(Mandatory)</I>
     * @return void
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public void setExpedite(boolean newExpedite) throws
com.ibm.sf.gf.SFException {
                setDirty();
                ivExpedite = newExpedite;
    }
    /**
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B>
     * param DeliveryTypeController owner <I>(Mandatory)</I>
     * @param DescriptiveInformation description
<I>(Mandatory)</I>
     * @param DCurrencyValue initFlatRate <I>(Mandatory)</I>
     * @param int initNumberOfDays <I>(Mandatory)</I>
     * @return void
     * @exception com.ibm.sf.gf.SFException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     * <BR><B>RESTRICTED</B> : This method is NOT for client
use
     */
    public void initialize(DeliveryTypeController owner,
DescriptiveInformation description, DCurrencyValue initFlatRate, int
initNumberOfDays) throws com.ibm.sf.gf.SFException {

                // NO MATCHING initialize METHOD
                // WAS FOUND IN THE PARENT CLASS
                // PLEASE UPDATE THE FOLLOWING
                // "super.initialize" CALL
                super.initialize(NO_MATCH_FOUND);

                BaseFactory factory = Global.factory();

                // Primitive types initialization
                ivNumberOfDays = initNumberOfDays;
                // No matching initialize parameter found for
attribute ivExpedite

                // Dependent initialization
                setFlatRate(initFlatRate);

                // No matching attribute was found for initialize
parameter owner
                // No matching attribute was found for initialize
parameter description
    }
    /**
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B>
     * @param DTime placeDate <I>(Mandatory)</I>
     * @return DTime
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public DTime calculateDeliveryDate(DTime placeDate) {
                // Please insert your code here
    }
    /**
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B>
     * @return String
     * <BR><B>Result:</B> a String containing the Id of the
     * Distinguishable object
     *
     * #ValidationNotRequired
     * <BR><B>PreConditions:</B> none
     * <BR><B>PostConditions:</B> the object is not modified
     */
    public String getId() {
                // Please insert your code here
    }
    /**
     * Retrieves the description of this object as a String
     * in a format/language determined by the locale that is currently
     * active in the environment this call was made in.
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B>
     * @return String
     * <BR><B>Result:</B> language dependent description selected
     * using the active Locale. Returns "" if
this.getDescriptiveInformation()
     * == null or if a description cannot be found using the standard
     * lookup mechanism.
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B> Object's state unmodified
     *
     * #index 1
     * #ValidationNotRequired
     */
    public String getDescription() {
                // Please insert your code here
    }
    /**
     * Retrieves the description of this object as a String
     * in a format/language determined by the given Locale.
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B>
     * @param String locale determines which locale dependent
     * form of the description to retrieve <I>(Mandatory)</I>
     * @return String
     * <BR><B>Result:</B> language dependent description of
     * object selected using the
     *             given locale. Returns "" if
this.getDescriptiveInformation()
     * == null or if a description cannot be found using the standard
     * lookup mechanism.
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B> Object's state unmodified
     *
     * #index 2
     * #ValidationNotRequired
     */
    public String getDescription(String locale) {
                // Please insert your code here
    }
    /**
     * Retrieves the DescriptiveInformation object encapsulating
     * the locale sensitive description of this object.
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B>
     * @return DescriptiveInformation
     * <BR><B>Result:</B> the DescriptiveInformation object
     * that represents the description of this Describable. Returns
     * null if this Describable has no DescriptiveInformation
     * attached to it.
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B> Object's state unmodified
     *
     * #ValidationNotRequired
     */
    public DescriptiveInformation getDescriptiveInformation() {
                // Please insert your code here
    }
    /**
     * Returns a descriptive string for the object
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
automatically
     * generated.
     * @return String
     * @exception SFRuntimeException
     * <BR><B>Result:</B>
     * <BR><B>PreConditions:</B>
     * <BR><B>PostConditions:</B>
     */
    public String toString() throws SFRuntimeException {

                try {
                        String retVal = super.toString();
                        return (retVal);
                }
                catch (Exception ex) {
                        throw (new SFRuntimeException(ex,
new

        TextResource("MSG_RMTSFEXCEP_DEFAULT","com.ibm.sf.g
f.resources.SFExceptionResources",
                                                        (Object
[]) null, "Runtime Exception has Occurred")));
                }
    }
    /**
     * Constructor of this class
     *
     * <BR><B>Assumptions:</B>
     * <BR><B>Note:</B> This documentation has been
```

```
automatically
                * generated.
                * @return void
                * @exception com.ibm.sf.gf.SFException
                * <BR><B>Result:</B>
                * <BR><B>PreConditions:</B>
                * <BR><B>PostConditions:</B>
                */
               public void DeliveryTypeImpl() throws
com.ibm.sf.gf.SFException {
                            // Please insert your code here
               }
               /**
                * Reads the state of an object from a stream
                *
                * <BR><B>Assumptions:</B>
                * <BR><B>Note:</B> This documentation has been
automatically
                * generated.
                * @param BaseStream stream <I>(Mandatory)</I>
                * @return void
                * @exception java.io.IOException
                * <BR><B>Result:</B>
                * <BR><B>PreConditions:</B>
                * <BR><B>PostConditions:</B>
                */
               public void internalizeFromStream(BaseStream stream) throws
java.io.IOException {

                            // Should we skip this internalization
                            if
(stream.skipThisClass(IMPLEMENTATION_NAME)) {

                super.internalizeFromStream(stream);
                                        return;
                            }

                            // internalize the version number
                            int objectIntStreamVersion = stream.readInt();

                            // read the primitive types
                            ivNumberOfDays = stream.readInt();
                            ivExpedite = stream.readBoolean();

                            // read the DescribableDynamicEntityImpl's (parent
of current class) state
                            super.internalizeFromStream(stream);

                            // read the contained dependents
                            ivFlatRate = (DCurrencyValue)
stream.readDependent(this, ivFlatRate);
               }
               /**
                * Writes the state of an object to a stream
                *
                * <BR><B>Assumptions:</B>
                * <BR><B>Note:</B> This documentation has been
automatically
                * generated.
                * @param BaseStream stream <I>(Mandatory)</I>
                * @return void
                * @exception java.io.IOException
                * <BR><B>Result:</B>
                * <BR><B>PreConditions:</B>
                * <BR><B>PostConditions:</B>
                */
               public void externalizeToStream(BaseStream stream) throws
java.io.IOException {

                            // externalize the version number.
                            stream.writeInt(versionNumber);

                            // Write the primitive types.
                            stream.writeInt(ivNumberOfDays);
                            stream.writeBoolean(ivExpedite);

                            // Write the DescribableDynamicEntityImpl's
(parent of current class) state
                            super.externalizeToStream(stream);

                            // Write the contained dependents
                            stream.writeDependent(this, ivFlatRate);
               }
               /**
                * Destroys the state of an object
                *
                * <BR><B>Assumptions:</B>
                * <BR><B>Note:</B> This documentation has been
automatically
                * generated.
                * @return void
                * @exception com.ibm.sf.gf.SFException
                * <BR><B>Result:</B>
                * <BR><B>PreConditions:</B>
                * <BR><B>PostConditions:</B>
                */
               protected void destroy() throws com.ibm.sf.gf.SFException {
                            BaseFactory factory = Global.factory();

                            // Dependent destruction
                            factory.deleteDependent(this, ivFlatRate);

                            // call parent to destroy its contained objects
                            super.destroy();
               }
}
```

DeliveryType factory.

DeliveryTypeFactory.java

```
package GPSE;

import com.ibm.sf.gf.*;
import com.ibm.sf.cf.*;

/**
 * <TT>
 * <BR><B>Purpose:</B>
 * <BR><B>Description:</B>
 * <BR><B>Note:</B> This documentation has been automatically
 * generated.
 * </TT>
 * @version 3.6.2
 * @since JDK1.0
 */
public abstract class DeliveryTypeFactory extends DynamicEntityFactory {
               /**
                *
                * <BR><B>Assumptions:</B>
                * <BR><B>Note:</B>
                * @param DeliveryTypeController owner <I>(Mandatory)</I>
                * @param AccessMode access <I>(Mandatory)</I>
                * @param DescriptiveInformation description
<I>(Mandatory)</I>
                * @param DCurrencyValue initFlatRate <I>(Mandatory)</I>
                * @param int initNumberOfDays <I>(Mandatory)</I>
                * @return DeliveryType
                * @exception com.ibm.sf.gf.SFException
                * <BR><B>Result:</B>
                * <BR><B>PreConditions:</B>
                * <BR><B>PostConditions:</B>
                */
               public static final DeliveryType
createDeliveryType(DeliveryTypeController owner, AccessMode access,
DescriptiveInformation description, DCurrencyValue initFlatRate, int
initNumberOfDays) throws com.ibm.sf.gf.SFException {

                            boolean finished = false;
                            DeliveryType newDeliveryType = null;
                            DeliveryTypeFactory factory =
(DeliveryTypeFactory)

            Global.factory().getSpecialFactory("GPSE.DeliveryType");

                            if (factory == null) {
                                        /*
                                        CODE GEN: CODE GENERATOR
FORCED COMPILE ERROR: PLEASE CHECK THIS
                                        CODE FOR THE ID PARAMETER
TO MATCH THE ID PARAMETER ON YOUR CREATE
                                        METHOD. THE CODE
GENERATOR IS MAKING A BEST GUESS AT THIS CODE...
                                        ONCE VERIFIED OR FIXED,
REMOVE THESE COMMENTS. REPLACE DUMMY ID
                                        PARAMETER WITH PASSED-IN
ID
                                        */
                                        String key =
StringFactory.createDMethodAccessKey(null, null);

                                        if
(owner.containsDeliveryTypeKey(key)) {
                                                    DResultMessage
resultMessage = DResultMessageFactory.

             createDResultMessage(ResultMessageSeverityEnum.SEVERE_E
RROR, "com.ibm.sf.cf","ACOMMON002");

             DReplacementTextTranslatableString classText =

             DReplacementTextTranslatableStringFactory.createDReplacement
```

```
TextTranslatableString("GPSE","DeliveryType");

              resultMessage.addReplacementText(classText);

              DReplacementTextNonTranslatableString keyText =

              DReplacementTextNonTranslatableStringFactory.
                                                                          //
CODE GEN: REPLACE DUMMY ID PARAMETER WITH PASSED-IN ID

              createDReplacementTextNonTranslatableString(PLEASE_REPLA
CE_THIS_ID);

              resultMessage.addReplacementText(keyText);


              com.ibm.sf.cf.ErrorsException exception = new

              com.ibm.sf.cf.ErrorsException(resultMessage);
                                                  throw (exception);
                                        }

                                        /* CODE GEN: END OF CODE
GEN COMMENTS */
                                        Entity entity =
Global.factory().createEntity("GPSE.DeliveryType", access, owner.getHandle(),
null);
                                        Exception caughtException = null;
                                        try {                          // cast
to correct interface and initialize
                                                  newDeliveryType =
(DeliveryType) entity;

              newDeliveryType.initialize(owner, description, initFlatRate,
initNumberOfDays);

              owner.registerOwnedDeliveryType(newDeliveryType);
                                                  finished = true;
                                         } catch (Exception excl) {
                                                  caughtException =
excl;
                                         } finally {
                                                  if (!finished) {
                                                            try {

              entity.uninitialize();

              Global.factory().deleteEntity(entity);

              DResultMessage resultMessage =


              DResultMessageFactory.createDResultMessage(ResultMessageSe
verityEnum.SEVERE_ERROR,"com.ibm.sf.cf", "ACOMMON052");

              DReplacementTextTranslatableString classText =


              DReplacementTextTranslatableStringFactory.createDReplacement
TextTranslatableString("GPSE","DeliveryType");

              resultMessage.addReplacementText(classText);


              throw (new com.ibm.sf.cf.ErrorsException(caughtException,
resultMessage));
                                                                          } catch
(com.ibm.sf.gf.GFException exc2) {

              DResultMessage resultMessage =


              DResultMessageFactory.createDResultMessage(ResultMessageSe
verityEnum.SEVERE_ERROR,"com.ibm.sf.cf", "ACOMMON05O");

              DReplacementTextTranslatableString classText =


              DReplacementTextTranslatableStringFactory.createDReplacement
TextTranslatableString("GPSE","DeliveryType");

              resultMessage.addReplacementText(classText);


              throw (new com.ibm.sf.cf.ErrorsException(caughtException,
resultMessage));
                                                                          }
                                                  }
                                        }
                                        else {                         // Delegate to special
factory
                                                  newDeliveryType =
                                                            factory.create(owner,
access, owner.getHandle(), description, initFlatRate, initNumberOfDays);
                                        }

                                        return (newDeliveryType);
                              }
                              /**
                               *
                               * <BR><B>Assumptions:</B>
                               * <BR><B>Note:</B>
                               * @param DeliveryTypeController owner <I>(Mandatory)</I>
                               * @param AccessMode access <I>(Mandatory)</I>
                               * @param Handle locationHandle <I>(Optional)</I>
                               * @param DescriptiveInformation description
<I>(Mandatory)</I>
                               * @param DCurrencyValue initFlatRate <I>(Mandatory)</I>
                               * @param int initNumberOfDays <I>(Mandatory)</I>
                               * @return DeliveryType
                               * @exception com.ibm.sf.gf.SFException
                               * <BR><B>Result:</B>
                               * <BR><B>PreConditions:</B>
                               * <BR><B>PostConditions:</B>
                               */
                              public static final DeliveryType
createDeliveryType(DeliveryTypeController owner, AccessMode access, Handle
locationHandle, DescriptiveInformation description, DCurrencyValue
initFlatRate, int initNumberOfDays) throws com.ibm.sf.gf.SFException {

                                        boolean finished = false;
                                        DeliveryType newDeliveryType = null;
                                        DeliveryTypeFactory factory =
(DeliveryTypeFactory)

              Global.factory().getSpecialFactory("GPSE.DeliveryType");

                                        if (factory == null) {
                                                  /*
                                                  CODE GEN: CODE GENERATOR
FORCED COMPILE ERROR: PLEASE CHECK THIS
                                                  CODE FOR THE ID PARAMETER
TO MATCH THE ID PARAMETER ON YOUR CREATE
                                                  METHOD. THE CODE
GENERATOR IS MAKING A BEST GUESS AT THIS CODE...
                                                  ONCE VERIFIED OR FIXED,
REMOVE THESE COMMENTS. REPLACE DUMMY ID
                                                  PARAMETER WITH PASSED-IN
ID
                                                  */
                                                  String key =
StringFactory.createDMethodAccessKey(null, null);

                                                  if
(owner.containsDeliveryTypeKey(key)) {
                                                            DResultMessage
resultMessage = DResultMessageFactory.

              createDResultMessage(ResultMessageSeverityEnum.SEVERE_E
RROR, "com.ibm.sf.cf","ACOMMONOO2");

              DReplacementTextTranslatableString classText =

              DReplacementTextTranslatableStringFactory.createDReplacement
TextTranslatableString("GPSE","DeliveryType");

              resultMessage.addReplacementText(classText);

              DReplacementTextNonTranslatableString keyText =

              DReplacementTextNonTranslatableStringFactory.
                                                                          //
CODE GEN: REPLACE DUMMY ID PARAMETER WITH PASSED-IN ID

              createDReplacementTextNonTranslatableString(PLEASE_REPLA
CE_THIS_ID);

              resultMessage.addReplacementText(keyText);


              com.ibm.sf.cf.ErrorsException exception = new

              com.ibm.sf.cf.ErrorsException(resultMessage);
                                                            throw (exception):
```

```
                                        }

                                        /* CODE GEN: END OF CODE
GEN COMMENTS */
                                        Entity entity =
Global.factory().createEntity("GPSE.DeliveryType", access, locationHandle,
null);
                                        Exception caughtException = null;
                                        try {                          // cast
to correct interface and initialize
                                                newDeliveryType =
(DeliveryType) entity;
                newDeliveryType.initialize(owner, description, initFlatRate,
initNumberOfDays);

                owner.registerOwnedDeliveryType(newDeliveryType);
                                                finished = true;
                                        } catch (Exception exc1) {
                                                caughtException =
exc1;
                                        } finally {
                                                if (!finished) {
                                                        try {

                entity.uninitialize();

                Global.factory().deleteEntity(entity);

                DResultMessage resultMessage =


                DResultMessageFactory.createDResultMessage(ResultMessageSe
verityEnum.SEVERE_ERROR,"com.ibm.sf.cf","ACOMMON052");

                DReplacementTextTranslatableString classText =


                DReplacementTextTranslatableStringFactory.createDReplacement
TextTranslatableString("GPSE","DeliveryType");

                resultMessage.addReplacementText(classText);


                throw (new com.ibm.sf.cf.ErrorsException(caughtException,
resultMessage));
                                                                } catch
(com.ibm.sf.gf.GFException exc2) {

                DResultMessage resultMessage =


                DResultMessageFactory.createDResultMessage(ResultMessageSe
verityEnum.SEVERE_ERROR,"com.ibm.sf.cf","ACOMMON050");

                DReplacementTextTranslatableString classText =


                DReplacementTextTranslatableStringFactory.createDReplacement
TextTranslatableString("GPSE","DeliveryType");

                resultMessage.addReplacementText(classText);


                throw (new com.ibm.sf.cf.ErrorsException(caughtException,
resultMessage));
                                                                }
                                                        }
                                        }
                        }
                        else {                          // Delegate to special
factory
                                newDeliveryType =
                                                factory.create(owner,
access, locationHandle, description, initFlatRate, initNumberOfDays);
                        }

                        return (newDeliveryType);
                }
                /**
                 *
                 * <BR><B>Assumptions:</B>
                 * <BR><B>Note:</B>
                 * @param DeliveryTypeController owner <I>(Mandatory)</I>
                 * @param AccessMode access <I>(Mandatory)</I>
                 * @param Handle locationHandle <I>(Optional)</I>
                 * @param DescriptiveInformation description
<I>(Mandatory)</I>
                 * @param DCurrencyValue initFlatRate <I>(Mandatory)</I>
                 * @param int initNumberOfDays <I>(Mandatory)</I>
                 * @return DeliveryType
                 * @exception com.ibm.sf.gf.SFException
                 * <BR><B>Result:</B>
                 * <BR><B>PreConditions:</B>
                 * <BR><B>PostConditions:</B>
                 */
                public abstract DeliveryType create(DeliveryTypeController
owner, AccessMode access, Handle locationHandle, DescriptiveInformation
description, DCurrencyValue initFlatRate, int initNumberOfDays) throws
com.ibm.sf.gf.SFException;
}
```

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Rational Software Corporation or Sun Microsystems, Inc.

Cited references and notes

1. http://www-4.ibm.com/software/ad/sanfrancisco/tools.html/.
2. Rational Rose—see *www.rational.com/rose/*.
3. UML—see *www.rational.com/uml/*.
4. Rose SF Bridge—see *www.metex.com/products/*.
5. http://www-4.ibm.com/software/ad/sanfrancisco/concepts/ibmsf.sf.T_SFConceptsAndFacilities.html/.
6. VisualAge for Java—see *www.ibm.com/software/ad/vajava/*.
7. Inprise/Borland JBuilder—see *www.borland.com/jbuilder/*.
8. http://www.rosearchitect.com/mag/archives/9810/extend.shtml/.
9. See http://embedded.com/98/9803fe3.html for a discussion of code generation techniques.

Accepted for publication December 10, 1999.

Russ Bunting *Metex Systems Inc., 350 Bay Street, Toronto, Ontario M5H 2S6 (electronic mail: Russ.Bunting@metex.com).* Mr. Bunting is manager of product development at Metex Systems. He earned a B.S. degree in computer engineering from the University of Toronto. Mr. Bunting is a member of the IEEE and is a certified instructor of Rational's object-oriented analysis and design techniques and UML. His current focus is upon applying these techniques, specifically visual modeling, toward the development of infrastructure and tools to promote software engineering best practices and their application to collaborative multiagent software systems.