UML-to-Java transformation in IBM Rational Software Architect editions and related software

Marya Berfeld

December 02, 2008

This article illustrates the application of Java transformations included in editions of IBM® Rational® Software Architect for interactive development in the context of design contract management protocols. These protocols address ways to manage the relationship between design models and the implementation code during the development cycle.

Transformations to support design contract management protocols were introduced in Rational Software Architect Version 7.5 releases. This is an introduction to some of the transformation features.

Note:
For simplicity, the transformations here are described in terms of Rational Software Architect. However, the information also applies to these editions and related IBM® Rational® products:

- Rational Software Architect RealTime Edition
- Rational Software Architect Standard Edition
- Rational Software Architect for WebSphere® Software

The author assumes that you have a basic knowledge of the Unified Modeling Language (UML) 2.0 specification, Eclipse, an edition of Rational Software Architect, and Java development. This article does not show UML modeling and Java code implementation in detail.

Process overview

The development process that includes transformation execution consists of these basic steps:

1. Create a UML model reflecting the structure of the application.
2. Use the UML-to-Java transformation to generate Java code based on the created UML model.
3. Edit the generated code and implement the business logic of the application.
4. (Optional) Keep the UML model and Java application synchronized by using the provided Java transformations.
Create a UML model of an ordering system

1. In Rational Software Architect, open the **Modeling** view and create a new model project by selecting **File > New > Model Project** from the main menu.
2. Name the project **order** and add a blank package called **Order Model** to the order project on the second page of the wizard. (see Figure 1).

**Figure 1. Create UML Model wizard**

3. Add the package called **orderingApplication** to the created model (Figure 2).

**Figure 2. UML sample model is created**
4. Develop the UML model. You can add elements to the Main Diagram of the orderingApplication package from the Class tab of the Palette and use modeling assistance in the diagram.

Figure 3 illustrates the model of a sample application that allows a business owner to generate an order and accepts payment from the customer.

**Figure 3. UML class diagram for the sample model**

![Diagram of a sample application model](image)

Use the transformation to generate Java code based on the model

Next, you need to create a transformation configuration to specify the transformation execution parameters.

**Configure the transformation**

1. To create a new UML-to-Java transformation configuration, select **Modeling > Transform > New Configuration** in the Main menu.
2. In the **Name** field of the opened New Transformation Configuration wizard (Figure 4), enter the name of the transformation configuration: **MyUMLtoJavaTransform** in **Name**.
3. Click the ellipses (…) button to browse to the UML project order.
4. In the list of deployed transformations, open the **Java Transformations** folder and select the **UML-to-Java** transformation.

5. You must select the protocol type on this wizard page (Figure 4).
Figure 4. The Main page of the New Transformation Configuration wizard

The protocol type specifies the Design Contract Management protocol (DCMP) that is recommended to be adopted in the development cycle. Using different design contract management protocols provides different levels of management and control of the design model and the implementation code. Transformations are involved in three types of DCMPs: Conceptual, Mixed, and Reconciled. The protocol type that you select in the transformation configuration determines the synchronization level between the source model and the target code.

**Conceptual protocol**
Choose this protocol if the intention is to use the source model as a primary engineering artifact in future iterative development. Any changes made to the source model will be propagated to the target project when someone runs the transformation.

**Mixed protocol**
This protocol is the best choice if conceptual UML models are iteratively evolved through the development life cycle. When transformation is run, the UML elements in the source model get converted to visual representations of Java elements, and the transformation output becomes the primary engineering artifact.

**Reconciled protocol**
You must select this protocol if conceptual models are maintained and iteratively extended throughout the design process, plus the intent is to evolve the project in both the Java and the UML domains and to synchronize the artifacts at certain points of the development cycle.

**Note:**
Using the UML-to-Java transformation in the context of the Conceptual protocol will be discussed here first. Using the transformation in the context of Mixed and Reconciled protocols will be covered later in this article.


With this protocol applied, any changes to the source model are propagated to the target model. After you run the transformation, it will be also possible to update the target project without changing the source model.

7. Click the **Next** button in the Transformation Configuration wizard (Figure 4) to move to a new page of the wizard, where you can specify the source and target elements for the transformation.

8. Select the model called **Order Model** from the created UML project as a source, and then select **Create a Target Container** (Figure 5). This opens a New Java Project wizard.

9. Create a new Java project named **myOrderCode** and select it as a target in the Transformation Configuration wizard.

**Figure 5. Source and Target Page of a New Transformation Configuration wizard**
10. Click **Finish** (Figure 5) to create a new Transformation Configuration file, **MyUMLtoJavaTransform.tc**, in the specified project **order**.

The Java project and transformation configuration files created are visible in the Project Explorer view, and the Transformation configuration is opened in the Editor view (Figure 6).

**Figure 6. The transformation configuration**

Run the transformation configuration

1. Click **Run** in the Transformation Configuration wizard to run the configuration.

The Java code will be generated in the target project (Figure 7).
Figure 7. Generated transformation output

The transformation generates the Java code based on the content of the source UML model:

- For classes and interfaces, corresponding xyz.java files are generated. The transformation assigns default file names (specified in the UML model) to the files and folders. For example, the Java OrderDetail class that corresponds to the UML OrderDetail class is generated.
- The transformation also generates Java methods and Java properties inside of the Java class files.

For example, the Java property `quantity` that corresponds to the UML attribute `quantity` is generated inside of the Java `OrderDetail` class as Listing 1 shows.

**Listing 1. Java OrderDetail class**

```java
/**
 * @generated UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)
 */
private Integer quantity;
```

For each Java property get and set methods -- for example `getQuantity()` and `setQuantity(Integer)` -- are generated by the transformation. The `calcSubTotal()` and `calcWeight()` Java methods that correspond to the UML operations `calcSubTotal()` and `calcWeight()` are generated inside of the Java `OrderDetail` class (Listing 2).
Listing 2. calcSubTotal() and calcWeight() UML operations in the OrderDetail class

```java
/**
 * (non-Javadoc)
 * @see Authorization#authorize()
 * @generated UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)
 */
public void calcSubTotal() {
    // begin-user-code
    // TODO Auto-generated method stub
    // end-user-code
    }
/**
 * (non-Javadoc)
 * @see Authorization#authorize()
 * @generated UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)
 */
public void calcWeight() {
    // begin-user-code
    // TODO Auto-generated method stub
    // end-user-code
    }
```

As you may notice, the transformation appends the `@generated` annotation to elements in the generated code (Figure 8).

**Figure 8. `@generated` tabs in the target code**

![Figure 8](image)

This annotation is to protect code. If the `@generated` annotation for the element that contains the implementation code is deleted or modified, the code will be protected and not updated when the transformation is run again into the same target project. (The enhanced options to ensure the protection of the user code that are also provided in the transformation design are not illustrated in this article.)

You might occasionally need to update the application. For example, you may want to add a new way of payment, such as accepting money orders. You can update the Java code directly
or add new classes to the model, and then you must rerun the transformation. The code is then regenerated and the new classes are added in the target Java project, leaving the protected code untouched.

**Establish a source-target trace relationship**

Transformation gives you an option to establish trace relationships between source and target artifacts. The trace relationships between the elements will help your team understand specification of elements, design decisions, and changed requirements as the system complexity increases.

1. To generate trace relationships between the transformation source and the target element, select the **Create trace relationships** check box on the Common page of the Transformation Configuration editor (Figure 9).

**Figure 9. The Common page of the Transformation Configuration wizard**

When someone runs the transformation, the annotation for each generated element that supports an `@generated` annotation is modified to include an EMF URI that represents the source UML element, as Listing 3 shows.

**Listing 3. EMF URI representing the source UML element**

```java
/**
 * @generated sourceid:platform:/resource/UMLS/UMLS.emx#_NLHDQIr0Edu0eunV5GC2yg
 */
public class Order {
    
}
```

This established a trace relationship between the source UML element and the target Java element. Established trace relationships allow enhanced merging between the target and source elements when the transformation is rerun. (The enhanced functionality is beyond the scope of this article.)
2. With trace relationships established, for each UML class or interface, you can trace the Java implementation by selecting the element in the Project Explorer and then selecting **Query > Traceability > Implementations** from the drop-down menu.
3. Click **OK** in response to the Save As dialog (Figure 10).

**Figure 10. Save As dialog for Traceability diagram**

A new diagram will be created that traces the relationship between the selected UML class and the Java class that implements it (Figure 11).
Specify alternate Java file names using mapping models

During the development process, it may be necessary to specify alternate (non-default) names for the target Java files or Java packages. For example, you may want to generate multiple Java packages that contain specific information for different clients or provide different class names. In the UML-to-Java transformation, alternative names or logic organization can be specified in the Mapping model.

1. To generate the Mapping mode, open the Mapping tab in the Transformation Configuration editor and select the check box next to Enable the mapping functionality for this configuration (Figure 12).
2. Click the **New** button and then click **OK** to accept the defaults in the Create a Mapping Model wizard (Figure 13).

**Note:**
You can choose any available project to store the Mapping model, but the Mapping model must be created in the same workspace where the source model is stored.
As a result, a new UML model is generated that contains UML artifacts for each UML element of the source model, and manifestation relationships are established between generated artifacts and corresponding UML elements. In this model, you can specify alternate file names for any of the Java files or Java packages that the transformation generates.

3. On the Mapping tab of the Transformation Configuration editor, click the **Edit Mapping** button.
4. When you see the Edit the Mapped names window of the Model Element wizard (Figure 14), select the UML element for which you want to specify an alternate name.
5. In the "Mapped name" field, type a new name for the selected element that is to be generated in the target project.

In Figure 14, **orderingApplication** package is selected and **orderingCD** is specified for the name.
Figure 14. Edit the Mapped Names wizard

6. Click **Apply**.
7. You can select another element then (for example, the **Customer** class), and specify a mapping name for it, too.

**Tip:**
You can use a qualified name, such as `com.ibm.NewCustomer.com`, or an unqualified name, such as `NewCustomer`.

In Figure 15, **NewCustomer** is the mapping name. Notice that specifying alternate names in a mapping model does not modify the source model.
Figure 15. Edit the Mapped Names wizard with the previously specified name for the package

8. Click **Apply** and then **OK** to close the Edit the Mapped Names view of the Model Element dialog.

In the Mapping model, the value of the File Name property for the Customer artifact is set to **NewCustomer** (Figure 16).

Figure 16. Properties of the Customer artifact in the Mapping model

9. Save the modified transformation configuration and run the transformation.

When you run the transformation with this Mapping model selected, the alternative file or package names that were specified in the Mapping model are applied to the elements that the
transformation generates. All classes beneath the package with the specified mapping name will be generated under the new package (Figure 18). This example generates a new Java package, orderingCD, that contains all Java classes available in the source UML model. The UML class called Customer is generated in this package, with the new name, NewCustomer, specified in the Mapping model.

**Figure 17, Output generated by UML-to-Java transformation with the Mapping model enabled**

The types of Java fields and operation parameters are also generated with the new name, NewCustomer (see Listing 4).

**Listing 4. Parameters also generated with the new name "NewCustomer"**

```java
/**
 * package orderingCD; import java.util.Set;
 */
/**
 * <!-- begin-UML-doc -->
 * <!-- end-UML-doc -->
 * @author Administrator
 * @generated UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)
 */
public class Order {
  /**
   * <!-- begin-UML-doc -->
   * <!-- end-UML-doc -->
   * @generated UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)
   */
  private NewCustomer customer;
  /**
   * @return the customer
   * @generated UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)
   */
  public NewCustomer getCustomer() {
    // begin-user-code
    return customer;
    // end-user-code
  }
```
Tip:
The use of the Mapping model in running transformations may be especially helpful when your plan is to deliver several Java applications generated from one UML model.

Decide whether to use the Mixed or Reconciled protocol

When to use the Mixed protocol

Select this protocol if your intention is to continue development in the target domain rather than in the source model. If you choose the Mixed protocol for the transformation, the UML elements in the source model will be converted to visual representations of Java elements during the transformation and the output becomes the primary artifact.

1. On the **Main** page of the Transformation Configuration editor (Figure 18), select the **Mixed** check box as the protocol.

   **Figure 18. Main page of the transformation configuration editor with Mixed protocol selected**

   1. On the **Main** page of the Transformation Configuration editor (Figure 18), select the **Mixed** check box as the protocol.

   2. Save the configuration.

Not all of the transformation features that were available for the conceptual protocol can be used now; the option to create trace relationships on the Common tab is disabled, for example. However, you can still create and use the Mapping model to specify alternative (non-default) file names for the transformation output. When the transformation is run, the UML elements in the source model are replaced with the references to the Java files.
3. Run the transformation and click **OK** in response to the warning dialog (Figure 19).

**Note:**
The displayed dialog informs you that the UML elements in the source model will be gone. Only visual representations of Java elements will be available in the source model if you click **OK** to the warning dialog. If you click the **Cancel** button in the warning dialog, the code is still generated into the target project, and the UML elements in the source model are not replaced with the references.

**Figure 19. Warning when the transformation attempts to replace UML elements**

In the Project Explorer in the UML model (Figure 20), references to the Java files rather than the UML elements are displayed.

**Figure 20. Project Explorer view of the output generated by UML-to-Java transformation using the Mixed protocol**

On the diagram that existed in the source model, the UML elements are replaced with the Java visual elements, as well (Figure 21). Notice that UML associations are replaced with generated attributes in Java classes.
Double-clicking any of the links in the Project Explorer or in the diagram will open the Java code for the element in the Java editor. If you need to restore the UML elements in the model, use this Main menu path: **Edit > Undo UML to Java 5 Transform.**

In future development, you can move completely to the Java domain or continue adding UML elements to the source model and reapplying the UML-to-Java transformation.

**When to use the Reconciled protocol**

If your intention is to evolve the project in both the Java and the UML domains independently and, at certain points of the development cycle, to synchronize the artifacts, choose the Reconciled protocol for the UML-to-Java transformation. This option corresponds to the Design Contract Management protocol named Reconciled Modeling.

On the Main page of the Transformation Configuration editor, select **Reconciled protocol**. The reverse Java-to-UML transformation is enabled and additional tabs specific to Java to UML Transformation appear in the editor (Figure 22): "Java to UML Options" and "Java to UML Associations." You can use these tabs to configure a reverse Java-to-UML transformation.
By running the forward (UML-to-Java) transformation, Java code is generated the same way as described earlier in this article. You can undertake future development in both Java and UML domains independently. At any point in the development cycle, the UML model can be synchronized with Java code by running the Java-to-UML transformation and fusing the result with the existing model. If the model was evolved independently from the code, further synchronization can be achieved by executing the UML-to-Java transformation using the existing Java project as a target. (The detailed description of the iterative development using the Reconciled protocol is beyond the scope of this article.)

The Reconciled protocol becomes especially useful if different teams model, implement, develop, or maintain the application. Reconciled modeling provides a strong support for managing architectural change during the development cycle.

Explore further details and get the sample model

This article covered the basics of the application of Java transformations that are included in the Rational Software Architect family of products for using Design Contract Management protocols for interactive development of Java applications. See the Help documentation in the software for more details.

The UML project that contains the order model can be downloaded, extracted, and imported into Rational Software Architect workspace (see Downloadable resources).
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML model used for this article</td>
<td>order.zip</td>
<td>10KB</td>
</tr>
</tbody>
</table>
Related topics

- Understand more about how the new functionality of the UML Modeler component common to both IBM Rational Software Architect Standard Edition Version 7.5 and IBM Rational Software Architect for WebSphere Software Version 7.5 by reading this article Using the new features of UML Modeler in IBM Rational Software Architect Version 7.5.
- Understand more about the new functionality of Rational Software Architect for WebSphere Software Version 7.5 by reading this article Overview of Rational Software Architect for WebSphere Software Version 7.5.
- Download Rational Software Architect for Websphere Software 7.5.
- Subscribe to an IBM developerWorks newsletter.
- Download a free IBM product evaluation.
- Try IBM Bluemix free for 90 days. Bluemix is a cloud platform as a service (PaaS). Bluemix supports several programming languages and services as well as integrated DevOps to build, run, deploy and manage applications on the cloud.

© Copyright IBM Corporation 2008
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)