Convert UML and BPMN model elements to different types automatically

How to use the model element conversion wizard in IBM Rational Software Architect 8.0.2

Lidija Grahek
August 23, 2011

This article describes the element conversion feature introduced in IBM® Rational® Software Architect Version 8.0.2. Especially in early phases of design, it is not clear exactly which modeling element to use, so you might use a UML class and later realize that you need a UML component, instead. To change it, you previously had to manually perform several steps to replace one element with another. The new element conversion feature automates all of those steps, so you save your time and reduce the chance of making mistakes associated with manual processing.

Sometimes, especially in early phases of design, it is not clear exactly which modeling element should be used, so you might make mistake, such as using a UML class rather than a UML component. In this case, you previously had to manually perform several steps to replace one element with another: create a new element, copy documentation and anything else that could be copied from the old element to the new one, reorient all relationships, delete the old element (which also deleted all references to it), recreate all references to the new element, and fix the shape and connector layout.

The element conversion feature introduced in IBM® Rational® Software Architect Version 8.0.2 automates all of these steps, so you save your time and reduce the chance of making mistakes associated with manual processing.

Note:
All examples in this tutorial are based on a UML model, but the same functionality works for BPMN models.

Activate the conversion feature

You can convert elements from either the diagram editor or the Project Explorer. In both cases, as Figures 1 and 2 show, the element menu includes a Refactor item that offers a Convert to New Element option.
Figure 1. Refactor menu option on a diagram shape or connector

![Refactor menu option on a diagram shape or connector](image)

Figure 2. Refactor option on a Project Explorer model element

![Refactor option on a Project Explorer model element](image)

The conversion feature is also enabled on diagram shapes and connectors that reference semantic elements from another model. If this feature is not on the menu, it means that the element cannot be converted automatically, because there are no known options for new type. For instance, some elements related to sequence and timing diagrams cannot be converted automatically.

You can select several elements of the same type at once to convert them all to the same new type.

**Set up models to use for this hands-on example**

Most of this article uses the following example:

1. Create four models:
   a. PackageA and PackageB in a project named Project1
   b. PackageC and PackageD in a project named Project2
2. Add elements to these models (shown in Figure 3):
   - In PackageD:
     a. Create a class diagram and, using the palette, create four classes with following names: MyName1D, Class1, Class2, and Class3.
     b. Add one attribute and one operation to the MyName1D class, and apply the realization stereotype to it.
     c. Create an association between MyName1D and Class3.
     d. Create a dependency between Class2 and MyName1D.
     e. Create an element import relationship between MyName1D and Class1.
   - In PackageC:
     a. Create a class diagram and, using the palette, create a class named Class1C.
     b. Drag the MyName1D node from the Project Explorer onto this diagram.
     c. Create an association between Class1C and MyName1D.
     d. Drag Class1C node from Project Explorer onto the diagram created in Step 1.
e. Create an association between Class1C and MyName1D.

- In PackageB:
  a. Create a class diagram and, using the palette, create a class named Class1B.
  b. Drag the MyName1D node from the Project Explorer onto this diagram, and change the appearance of created shape. For example, change the fill color to brown and font color to white.
  c. Create a dependency between Class1B and MyName1D.

- In PackageA:
  a. Create a class diagram and, using the palette, create a class named Class1A.
  b. Add an attribute to Class1A and set its type to be MyName1D.

**Figure 3. Setup for the examples in this article**

Select a new element type

1. Right-click on the **MyName1D** class shape (in PackageD), and select the **Refactor > Convert to New Element** from the menu.
2. In the Convert to New Element wizard (Figure 4), select a new element type and the change scope.

Figure 4. Element conversion wizard page to select element type and scope

Tip:
The element type selection tree has a similar structure to the diagram palette. You can manually browse the tree, or enter a partial or whole element type name in the filter field at the top, where the prompt says "Enter filter text here." You can use plus sign (+) to expand the content of the tree, and minus sign (-) to collapse it.

Notice that the tree includes only those element types to which the selected element (in its current container) could be converted. A general rule is that an element type can be converted only to an element type from the same modeling domain. For example, a UML element can be converted to another UML element, and a BPMN element can be converted to another BPMN element. Another general rule is that if you are converting a shape-related element type, you can choose only another shape-related element type. For example, you cannot convert a class to an association. The same holds for relationships; you can convert a relationship type to another relationship type. The only exception is association class, which can be converted to another relationship type or to a class. In this example, conversion to elements found in the Composite Structure Palette drawer cannot be performed; therefore, those elements are not included in the element type tree.

Select the scope of the change

When an element is converted to another type, the unique element identifier (ID) is not kept. Instead, a new ID is assigned (see the note at the end of this section for a brief explanation). This
means that all references to that element need to be updated or they will become unresolved. These references can belong to the same model, to another model in the same project, to another project in the workspace, or to a model that is outside of the workspace. You can control which affected models will be updated by what you choose from the "Change scope" list and further refine this by selecting which affected resources in that scope should be excluded (see the following section for more details).

The "Change scope" list on the element conversion wizard page shown in Figure 4 gives you these options:

**Workspace**
If you accept the default Workspace scope, references to the converted element will be updated in all affected models in the workspace.

**Project**
By choosing the Project scope, you restrict the reference update to only those affected models that are in the same project as the model that contains the converted element.

**Model**
Finally, you might want to consider only references in the same model that contains the converted element. In that case, choose the Model scope.

The Project and Model scope options are useful for those cases where either you are confident that there are no references to the converted element outside of the chosen scope or you do not want to update the references. Selecting one of these two options reduces the scope of search for references, thus speeds up the conversion process (that can be especially noticeable in a large workspace). However, all references to the converted element that were not updated during conversion will be unresolved. Currently, the only way to repair the unresolved references is to manually replace them with references to the converted element.

**Note:**
As mentioned at the beginning of this section, the unique element identifier (ID) is not kept during the conversion process. This is why we use the term "conversion," not "morphing." The reason for not keeping the ID value is that there is a high risk of corruption for models that reference the converted element but were not included in the conversion change scope. If the new element had the original ID, these references would be unresolved rather than incorrect. This could lead to model corruption, which could prevent opening the affected model or some of its diagrams.

**Check the "Affected resources" report**
1. In the element conversion wizard, select the **Interface** element type and leave the default Workspace change scope selected.
2. Click **Next**.
You can now review and possibly clear check boxes for some of the affected resources. Figure 5 shows the affected resources list for this example, when the PackageA and PackageB models are closed and PackageC and PackageD are opened.

In Figure 5, notice that PackageC and PackageD are marked as mandatory. This means that they have to be included in the conversion: PackageD because it contains the MyName1D class element to be converted and PackageC because it contains a reference to MyName1D element, and it is open.

**Figure 5. Element conversion wizard, "Affected resources" view**

![Affected resources view](image)

PackageA and PackageB also contain references to the converted element, but they are closed, so it is not mandatory to update them. Closing the PackageC model would mean that changes to it would no longer be mandatory. Changes to the owning model (PackageD) would always be mandatory. You can uncheck check boxes for any of non-mandatory resources from the list if you do not want to update their references. If you uncheck one or more of mandatory resources, you will get an error message and will not be able to proceed with the conversion.

3. Now, click the Back button, select the Model change scope, and click Next.

Because the Model change scope is selected, only PackageD is listed as an affected resource. If you click OK at this point, you will get an error message (see Figure 6), because PackageC -- a mandatory affected resource -- is not included in the change scope.
Figure 6. Error results because PackageC is mandatory but not included in the scope

4. Click Cancel to close the wizard.

Review conversion results

1. Right-click again on MyName1D class shape, and select the Refactor > Convert to New Element menu option.
2. In the element type selection tree, expand the Class node and select Interface.
3. Leave the default Workspace change scope selected, and click OK.

If some of the affected resources are read-only, you will be prompted to make them writable. If you choose not to, the conversion is canceled.

If you have large models in your workspace and you have not performed a search yet, then it might take a while for search indexing to finish, so that the search for references can be performed. You can monitor the progress bar. When the wizard closes, the element conversion is finished. Notice the following results:

- The MyName1D class is now MyName1D interface.
- The class-specific shape that was pointing to the MyName1D class is now an interface-specific shape that points to the MyName1D interface.
- Operation1() and attribute1 of the converted class are moved to the resulting interface.
- The realization stereotype previously applied to the MyName1D class is also applied to the converted MyName1D interface.
- All relationships that have MyName1D as a source or target are preserved. Their representations on diagrams (connectors) are also preserved in both PackageD and other models (PackageB and PackageC).
- All diagram shapes and connectors related to the converted element preserved their layouts for shapes (location and size) and routing for connectors.
- In PackageC, the class shape on the diagram that pointed to the converted MyName1D class from PackageD is now an interface shape that points to the MyName1D interface.
- The same as above holds for shape in PackageB. Also notice that its appearance is preserved.
- In PackageA, attribute1 of Class1A was Class MyName1D class, but now it is MyName1D interface.

If you did not use the conversion operation in this example, but made the changes manually instead, you would have had to make each of the changes above manually.
Decide whether to proceed or cancel if you get warnings

In the previous example, all of the feature values of the converted element were preserved. However, sometimes the converted element has features that cannot be contained by the new element type. For example, you will now convert a class into a package (it is not very likely that you will need to do this type of conversion, but it serves well to demonstrate the data loss that can occur).

1. Undo the previous conversion by using the **CTRL + Z** key combination or the **Edit > Undo** menu option.
2. Right-click again on the **MyName1D** class shape, and, from the menu, select **Refactor > Convert to New Element**.
3. In the element type selection tree, expand the **Class** node and select **Package**.
4. Leave the default Workspace change scope selected, and click **OK**.

Figure 7 shows a part of the conversion result, and Figure 8 shows data loss warnings, with which converted elements lost specific data and which resources were affected.

5. Review the results and any data loss warnings.

**Figure 7. Partial result of the class-to-package conversion**
Figure 8. Sample of data loss warnings

In this example, there are several features that could not be preserved during conversion. The attribute1 of Class1 is of type MyName1D, but the new element type of MyName1D is "package." Attributes cannot be of the package type, so attribute1 no longer has a type. Also, because a package cannot be a source or target of an association, two relationships have been deleted. A package does not have attributes and operations; therefore, attribute1 and operation1 that were set on the MyName1D class have been lost. Finally, the realization stereotype is not applicable to package; therefore, it has not been applied to MyName1D after its conversion to a package.

6. After reviewing the warning messages, if you decide to revert the conversion-related changes, click **Undo**. Otherwise, click **OK**. This dismisses the warning messages and leaves the conversion results in place.

Convert shapes or connectors without semantics

So far, we have been exploring the conversion of semantic elements. Conversion can also be applied to diagram shapes and connectors that do not have semantic references (geoshapes). For example, a geometric shape Oval can be converted to a UML class. This option is useful if you are making a draft of a model using geometric shapes and later want to use it to create the real model.

1. For example, create two ovals in a diagram with a line between them.
2. Select both ovals, right-click, and select the **Refactor > Convert to New Element** menu option.
Figure 9. An example of converting geoshapes

![Image of geoshapes conversion](image)

Because geometric shapes have no semantic references, there is no need for the wizard-type dialog windows used for semantic elements. You can use the element type selection tree in the same way as for semantic elements.

3. For example, type `Class` in the filter box, and press Enter.

The pop-up window closes and selected ovals are converted to classes.

Figure 10. Result of converting two oval geoshapes to classes

![Image of converted classes](image)

After converting the ovals to classes in previous example, you can convert a line between them to a relationship. A line that connects two shapes with semantic references can be converted to any relationship that is supported between the semantic elements that it connects. In this example, the line can be converted to any type of relationship allowed between two classes.