Comparing and merging UML models in IBM Rational Software Architect: Part 2

Merging models using "compare with each other"

Level: Introductory Kim Letkeman (kletkema@ca.ibm.com), Development Lead, Modeling

Compare Support, IBM Rational 12 Jul 2005

IBM® Rational® Software Architect (IRSA) is built on the Eclipse IDE and shares Eclipse's compare support workflows. IRSA UML models are built using the Eclipse Modeling Framework, so cannot be merged using the default Eclipse text compare support. This is Part 2 of a multi-part article discussing how to compare and merge UML models in Eclipse using a custom EMF and UML compare support solution. This article covers the *Compare with each other* use case.

Part one of this article introduced the compare support for UML models in IBM® Rational® Software Architect (IRSA), and the smaller IBM® Rational® Software Modeler (IRSM) product variant. We looked at the Compare With Local History command, exploring the following:

- The compare editor for models
- How differences are summarized and highlighted
- Navigation
- View modes
- And so on

This part of the article will continue exploring Eclipse compare and merge scenarios. For the other parts in this series, see Resources.

Eclipse compare support workflows

Eclipse compare and merge facilities are used in two common work flows:

- Compare or replace with local history, which may be invoked from the context menu of anyselected resource. A dialog shows all previously saved states for the resource, and when you click on a date stamp, the current resource state is compared with the selected saved state. The replace variant allows the current artifact to be overwritten with any previously saved version of the artifact. This use case is covered in Part 1 of this series.
- Compare with each other, which may be invoked from a navigator view context menu when two or three resources are selected. You would use this workflow to compare or merge artifacts that are managed manually in the workspace. The models must be of common ancestry in order for comparison to work. The two-way and three-way merging use cases are covered in this article.

Terminology

The following terms are used without further explanation in the rest of this article. Please review these before proceeding.

Ancestor: an artifact that is the common parent for other models. In a three-way merge, the ancestor is compared against each of the contributors to generate lists of differences. The differences are then compared to generate a list of conflicts.

Artifact: any file that is stored in an Eclipse project. An artifact may contain a whole model, or it may be one of several physical artifacts that constitute a larger logical model. In Eclipse, artifacts are often called resources.

Automatically resolvable conflict:any conflict where the result of accepting the change in either contributor is identical.

Base: same as ancestor.

Composite difference group:a group of related differences. All diagrams are represented by composite difference groups so that the user can operate on the whole diagram as a group, and so that clutter is reduced when several diagrams have changed. Composite difference groups can be atomic, where accepting or rejecting one member difference automatically performs the same operation on the whole group. Furthermore, a single difference can be a member of more than one composite difference group.

Conflict: two differences that are incompatible with each other. This can only occur during a three-way merge. Some examples are: (1) each contributor changed a class's name; (2) one contributor moved a class to another package while the other deleted the target package entirely; (3) one contributor added an operation to a class and the other contributor deleted the class. When both contributors make identical changes, a conflict is noted but is automatically resolved.

Contributor: one participant in a two-way or three-way merge. A contributor is compared to an ancestor (base) model to generate a list of differences. The two contributors in a three-way merge are never directly compared.

Cross-model reference: a reference that crosses the boundary between two physical artifacts. In IRSA, a good example of this is a reference from a class's view on a diagram in one model artifact to the semantic class object in a package in a different model artifact.

Custom profile: a meta-model extension that is created and managed locally.

Diagram: one specific pictorial view into a model's semantic data (for example, a class diagram or sequence diagram). A model (*.emx) may contain many diagrams.

Diagram (2): the output of a visualization operation, usually stored in an artifact with a .dnx, .tpx, .iex or .idx extension.

Delta: same as difference.

Difference: one change between an ancestor artifact and a contributor artifact. The five differences that are recorded are: add, change, delete, move and reorder. Reorder is actually a specialized move from one position to another inside the same list.

EMF: Eclipse Modeling Framework. This is the low-level modeling format in which you develop and store your UML models inIBM's Rational software products. A single UML construct may require multiple EMF constructs. This shows up in compare and merge sessions because all underlying changes to the model must be accounted for in the difference list.

Model: a UML model created by IRSA. The file extension for a model is .emx.

Profile: an extension to a UML meta-model, stored in an artifact with a .epx extension.

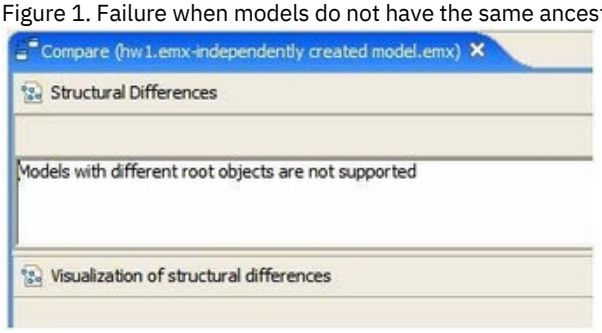
Reference: a pointer from one element in a model to another. The classic example is a class's view on a diagram pointing to the actual class object in a package in the model.

Compare with Each Other

The compare with each other command allows for comparison between two or three versions of a model in your Eclipse workspace. In the IRSA model compare support, a merge session is always started for compare commands (because the compare always generates a separate merged model). You can write the merged model to any location at any time using the Save a Copy button in the top right corner of the merged model pane. Also in that area of the merged model pane are buttons to save over each contributor. These are enabled when their respective contributors are writable.

For comparison to function, the models must come from a common ancestor at some point - which really means that the root model element in each of the files must have the same identity, or the error shown in Figure 1 will occur.

Figure 1. Failure when models do not have the same ancestor

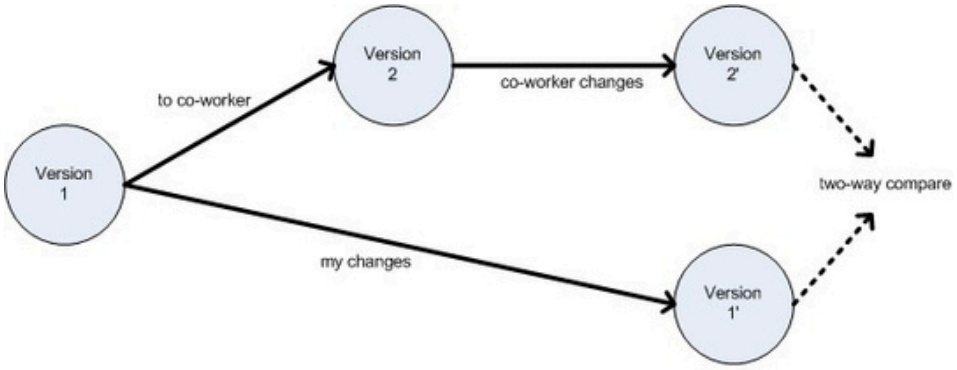


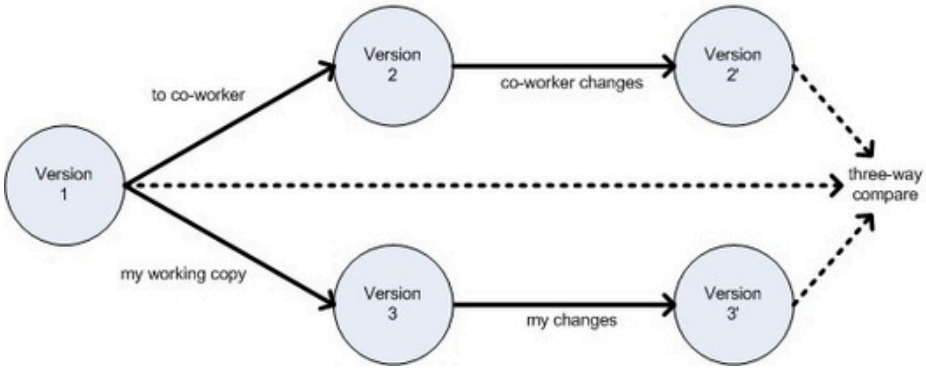
Work flows

The primary reason for comparing two or three models is so that you can understand and perhaps combine changes that have been made in parallel. The examples in this article, therefore, are all based on a desire to do some parallel development on your model with a co-worker. Say you send a copy of the model to a co-worker and continue working on it yourself. When your co-worker sends back the modified copy, you want to merge his or her changes with yours.

To accomplish this result, you can perform a two-way merge on the two changed models, applying the changes from your co-worker's copy to your copy. Alternately -- if you saved a copy of the original model before you made your changes -- you can run a three-way merge and accept changes from each side, resolving conflicting changes along the way. Three-way comparison provides more information as to what was changed by each of you, and allows you to accept or reject changes from either modified model with great precision. The saved copy is, of course, the common ancestor for the two changed models. The two flows look like Figure 2.

Figure 2. Two-way and three-way parallel development work flows





Starting with the model (hw1.emx) from Part 1 of this article, make a copy (hw2.emx) for your co-worker and make changes to each copy independently. In order to be able to contrast the two-way and three-way scenarios, you're actually going to make two copies (adding an hw3.emx) and then compare hw2 (co-worker's copy) to hw3 (your copy) in two-way mode. You'll later compare all three of them in three way mode (hw1 will be the ancestor.)

Copy the models

Copying the models is a simple workspace operation, accomplished by right-clicking the model and using the appropriate commands from the context menu. Figures 3 through 6 show one flow to make this clear.

Figure 3. Copy the model

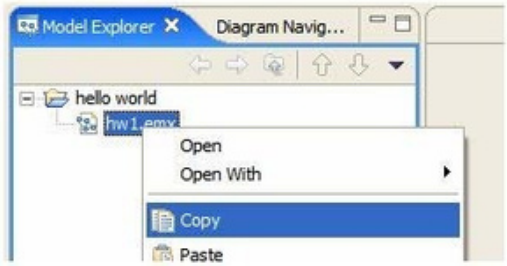


Figure 4. Paste the new model

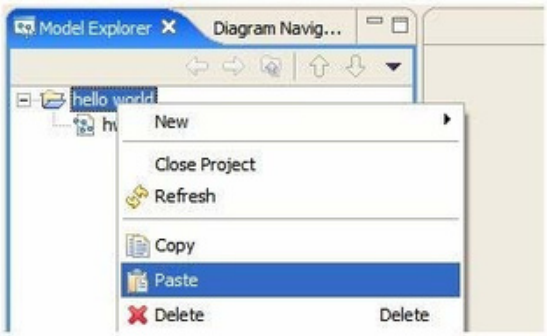


Figure 5. Rename on paste

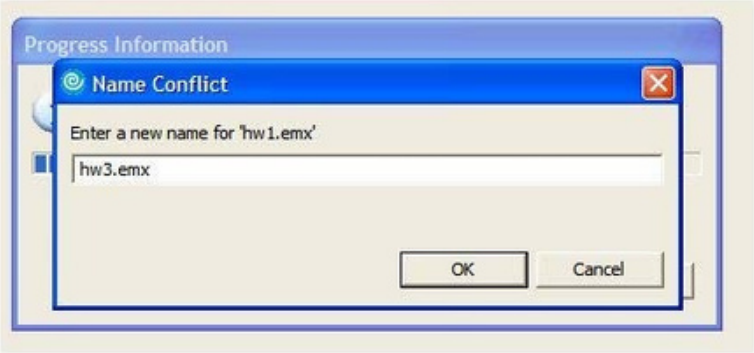
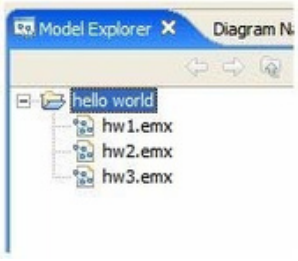


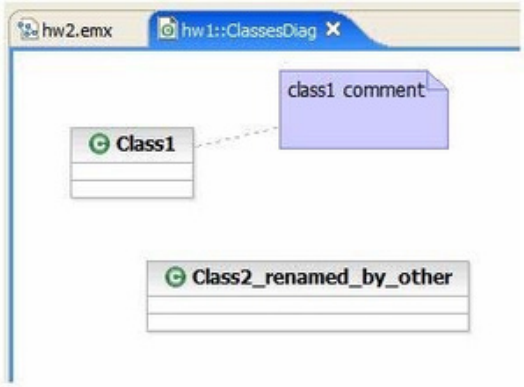
Figure 6. Workspace with three identical copies



Change both copies independently

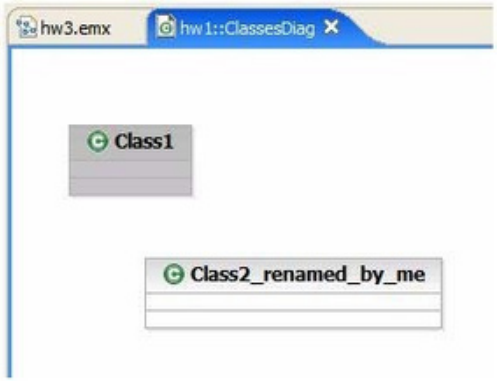
In hw2, add a comment on one of the classes and rename the other, as shown in Figure 7.

Figure 7. hw2 after "co-worker's" changes



In hw3, rename the same class as you did in hw2, and then change the background color of the other class (the one with the added comment in hw2.). Figure 8 illustrates these changes.

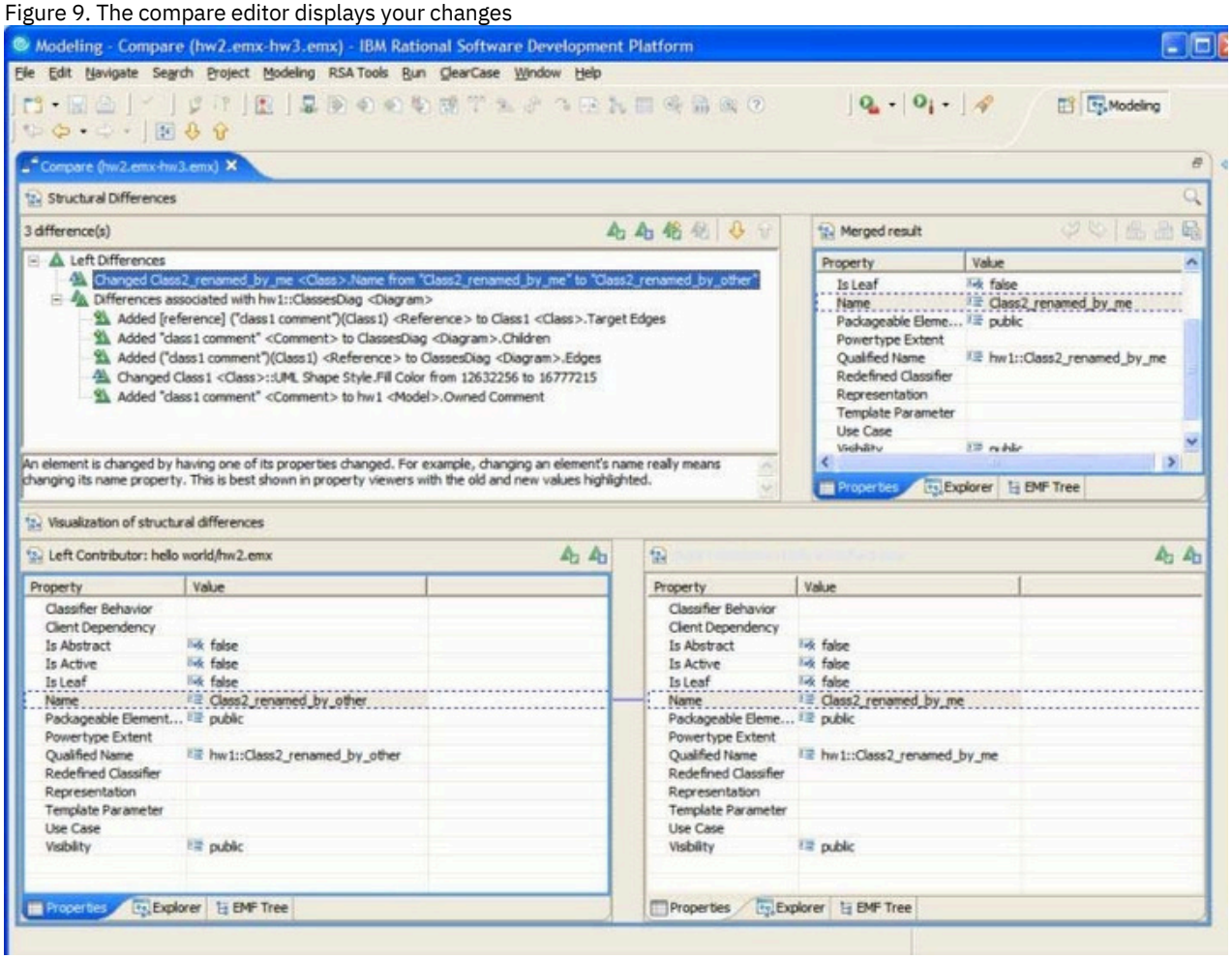
Figure 8. hw3 after your changes



Two-way merging

As mentioned previously, you'll simulate the case where you change the ancestor directly without saving a copy by comparing hw2 (representing your co-worker's changes) and hw3 (representing your own changes). The compare editor shows the following deltas, as seen in Figure 9.

Figure 9. The compare editor displays your changes



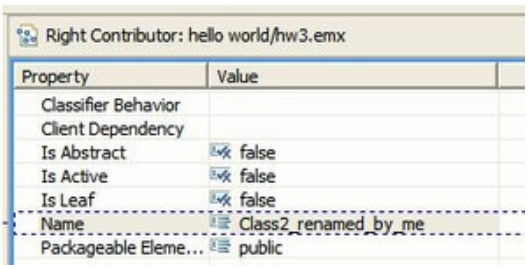
Here are a few things that you should know about this compare editor:

The left contributor is your co-worker's version, and the right contributor is your version. Remember that the comparison is always performed left minus right, which means that the deltas treat the right contributor (your version) as the base (or ancestor) model.

Note: The actual order of display in the navigator view does *not* consistently determine which model becomes left and which becomes right in the compare editor. It could have come up with hw2 on the right. *The order actually depends on the order of the file names in the selection provided to* compare support *navigator from which the compare session was launched.* If the files do not come up in your preferred order, you might try renaming one of them to change their appearance order, or changing the sort order on the navigator (this is possible in the model explorer.) You might also try running the comparison from the Eclipse *resource navigator view* instead of the model explorer.

The header for the right contributor appears dimmed, which indicates that it is unavailable. There is also no blue border around the right contributor (as there is around the left contributor), which also indicates that the pane is unavailable. You would expect the right contributor pane to be, in fact, available, since there is a delta highlighted in the structural difference viewer. Clicking the first delta makes the right pane available again, as shown in Figure 10:

Figure 10. After clicking on the first delta



A merged result pane has appeared beside the structural differences viewer. This version of the model starts out as a copy of the

base or ancestor model, which is always in the right contributor in a two-way merge session. In this case the base version is your version (hw3). *The merged model changes in real time* as you accept or reject changes made by your co-worker.

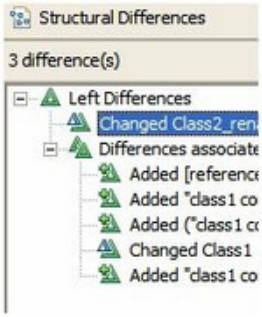
Accepting and rejecting changes

Each change in a two-way merge session must be examined very carefully before accepting or rejecting it. If the other person changed the element, then accepting the change will take the other person's new value. But if the delta exists because *you* changed the value and the other person *did not*, then accepting the change will actually *revert the value back to the ancestor value*.

This is the key flaw with two-way merging. When comparing contributors without the base model, it is impossible to tell which contributor actually made a change to the value. In order to understand the difference in context, it is necessary to open the ancestor model (if it is available) and look at the original value. This is much more cumbersome than three-way merging, as we'll see later.

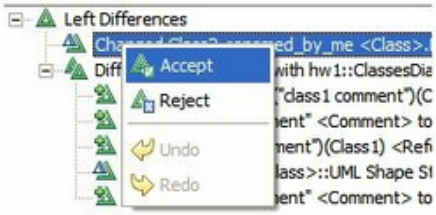
The icons on each delta in the structural differences viewer are undecorated when the merge starts, as shown in Figure 11.

Figure 11. Delta icons



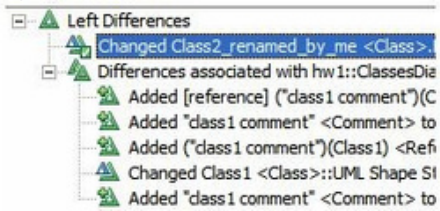
This simply means that they are considered unresolved -- neither accept nor rejected at this time. When you accept one (by right-clicking the delta and clicking Accept on the context menu), the icon changes to the accepted state and the merged model is updated. Figures 12 through 14 show this flow for the first delta.

Figure 12. Accept command



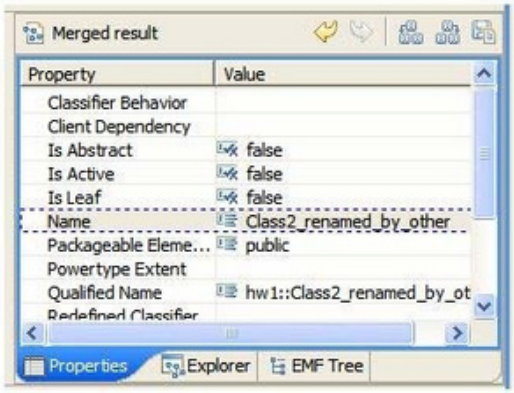
The accept command marks the delta with a small green check box, indicating that it is resolved as accepted.

Figure 13. After the accept command



The merged model has also been updated with the value from the left contributor.

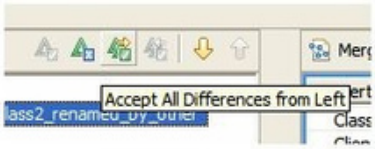
Figure 14. Merged result after the accept command



Shortcuts for accepting multiple differences

From here, you could accept each individual change, which would be tedious were there dozens of differences, much less hundreds. IRSA provides a couple of useful short hands to save you this trouble. If you are confident in the changes, you can use the Accept All Differences from Left button, as shown in Figure 15.

Figure 15. Accept all differences from left button

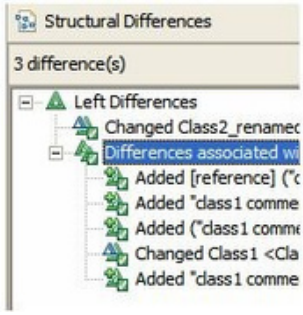


Alternatively, you can Accept all the differences in individual difference groups, as shown in Figure 16.

Figure 16. Accept all differences in a difference group

The end result of both of these shortcuts is that all of the differences from your co-worker are accepted, as shown in the structural difference decorations seen in Figure 17.

Figure 17. Differences marked as accepted



Losing changes in a two-way merge

Now you need to consider whether the result is what you actually meant to happen? The two kinds of changes you need to be careful with are:

- Conflicting changes, which are not flagged but which you should try to look for as you explore the deltas.
- Changes that are made only in our base contributor. These are flagged as deltas, but they *revert to the ancestor* value if accepted. In other words, you will lose your changes if those deltas remain accepted.

You dealt with the only conflict previously. But there *is* a change in your contributor that does not have an equivalent in the other contributor: the background color change for class1. By accepting that change, you actually lose your change in favor of the original value from your co-worker's version of the model. This happens because your contributor is used as the base. Thus, all change differences are relative to your value as the *from* value. Your coworker's value is the *to* value in this case.

This is easier to see when the change is displayed in the diagram view mode (see Figures 18 and 19), since color values are otherwise displayed cryptically as numbers representing a combination of the red, green, and blue values.

Figure 18. Left contributor retains ancestor value

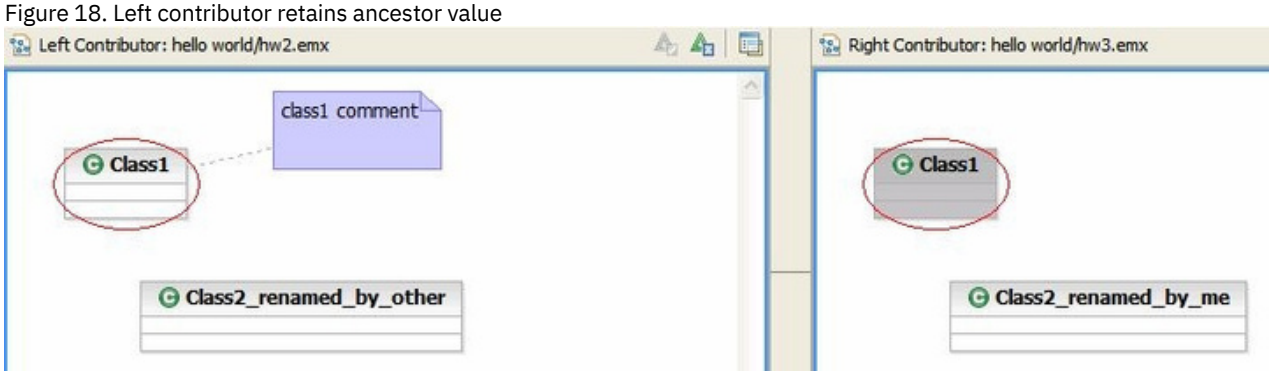
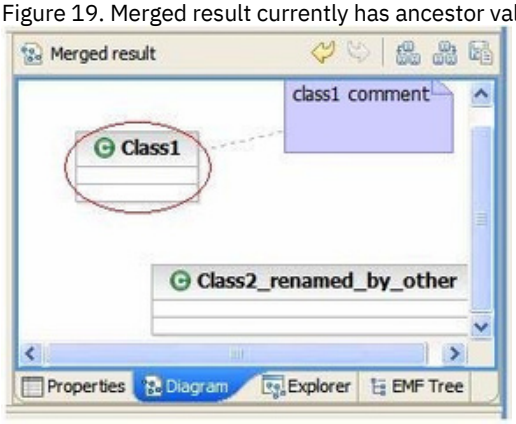
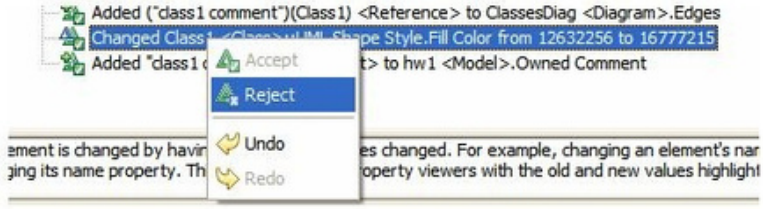


Figure 19. Merged result currently has ancestor value



To fix this, we have to reject the delta on the background color, as shown in Figure 20.

Figure 20. Reject command on background color delta



The result is that this delta is decorated with a small blue unchecked box (Figure 21) -- indicating resolution by rejection -- and the merged model has your value for the background color of class1 (Figure 22).

Figure 21. Rejected delta

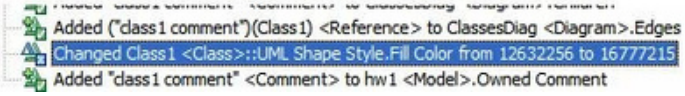
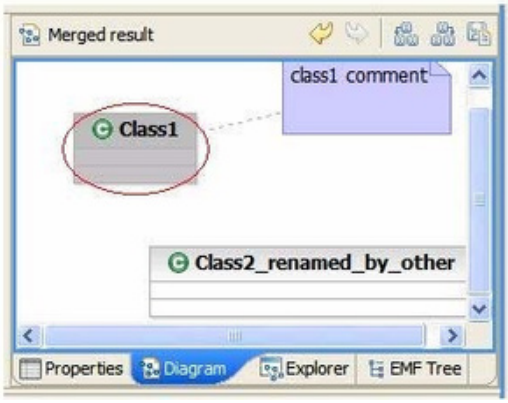


Figure 22. Rejected color change leaves your value in place



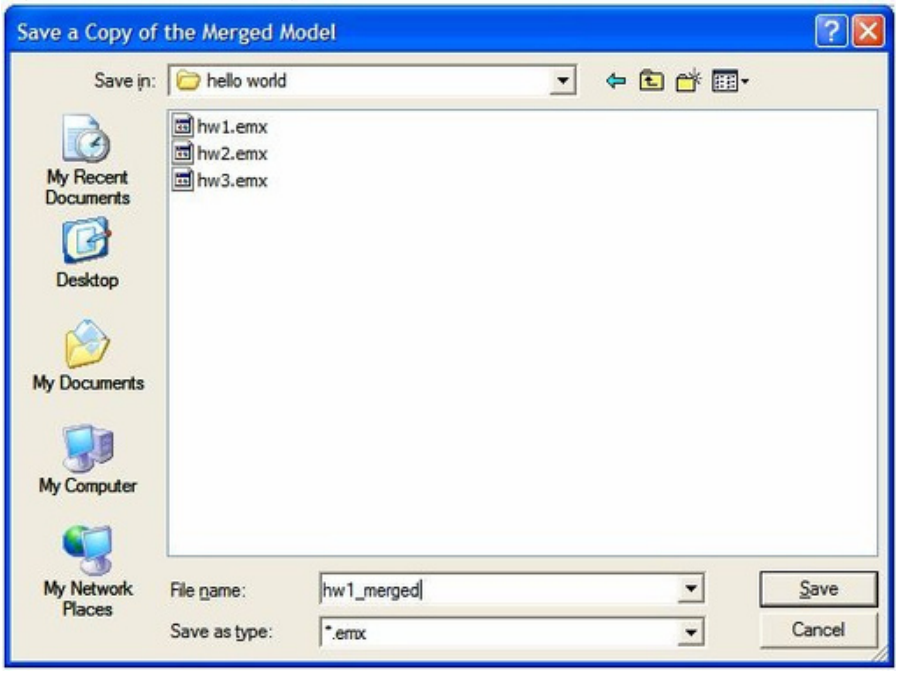
Saving the merged result

We have successfully retained all of the changes from both contributors. The merged model is now complete. The next step is to save it. Because each of your models is writable, you have three choices. There are three buttons in the top right-hand corner of the merged model pane that allow you to save over the left or right contributors, or to save a copy of the current model to a separate target file. This is illustrated in Figure 23.

Figure 23. Save as Left Contributor, Save as Right Contributor, and Save a Copy buttons

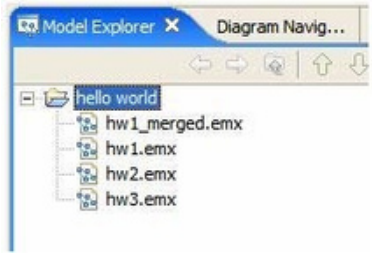
You can save intermediate copies of the merged model as often as you want without damaging the contributors themselves. Remember that you can always use Replace With Local History to restore the model if you make a mistake and save over the wrong contributor. The following represents the *Save a Copy* work flow. After you click the far right-hand button, this dialog appears. In Figure 24, you've drilled down to the workspace project directory and entered a name for the file ? hw1_merged.

Figure 24. Save a copy dialog



After clicking save and refreshing the Eclipse workspace (can't forget to do that) your workspace contains the fourth file, as shown in Figure 25.

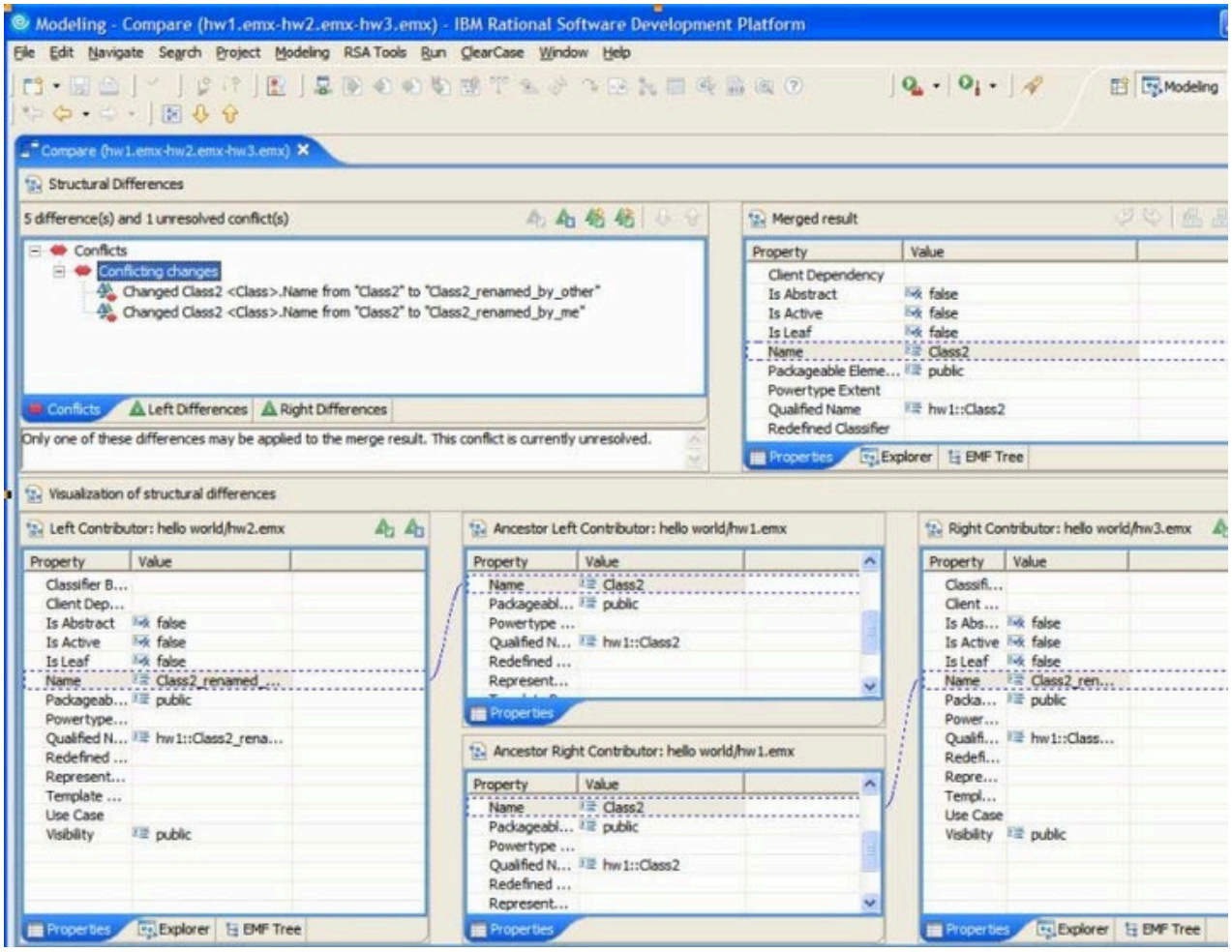
Figure 25. Merged model in the workspace



Three-way merging

Next, this article will discuss the same models in a three-way merge session. You saved the original hw1 version of the model to serve as the common ancestor of your two changed models. Now, by selecting all three, you can use the Compare With Each Other command, which will result in the window shown in Figure 26.

Figure 26. Three-way compare session



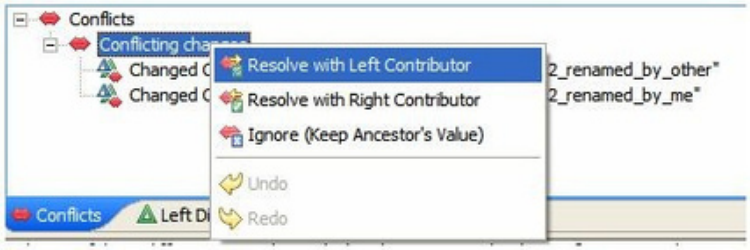
Following are some notable additions in the three-way merge compare editor (as opposed to the two-way merge compare editor):

A Conflict tab in the structural differences viewer. A three-way merge compares contributors separately against the ancestor version and then compares the two delta sets to find conflicts -- changes that cannot be accepted together. The only conflict that we created was the rename of Class2 in both contributors.
Two Differences tabs in the structural differences viewer. It is possible to explore all model differences independently on the left and right sides of the merge session.
A split ancestor pane (between the two contributors). Each of the contributors has a slaved ancestor pane so that its deltas can be properly highlighted. The reason for the existence of the split pane is that conflicts can affect different ancestor elements, or require different default view modes. An example of a situation where both reasons exist is the renaming of a diagram (which affects the diagram element and requires a property viewer) and the deletion of the diagram's parent package (which affects the package element and requires a tree viewer). This cannot be rendered correctly with a single ancestor pane.

Resolving conflicts

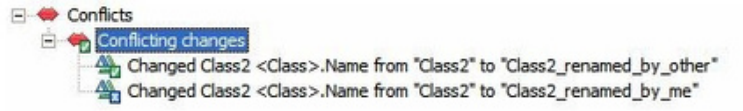
When resolving a conflict, you can choose the left, the right, or the original value. Figure 27 shows you choosing the left value.

Figure 27. Conflict resolution choices



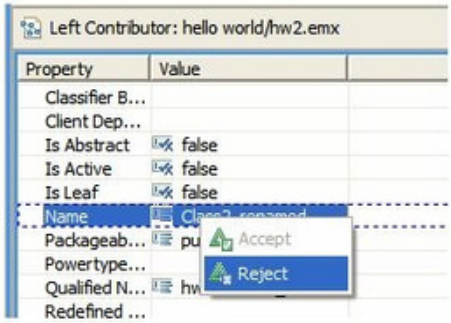
Once you do this (Figure 28), the conflict node is marked as resolved (a green checked box), the left (top) delta is marked as accepted (also a green checked box), and the right (bottom) delta is marked as rejected (a blue box with an "X"). The change is rendered in the merged model pane as well.

Figure 28. Resolved to the left contributor



Note that there are other ways of obtaining the same result. For instance, the deltas shown in the conflicts pane can be accepted and rejected individually. They can also be accepted or rejected in the contributor windows themselves. Furthermore, the deltas are state aware, meaning that only valid commands are available to you. Finally, their effect cascades to all related deltas. For example, with the left conflicting delta accepted already, the only available command in the left contributor itself is the reject command, as shown in Figure 29:

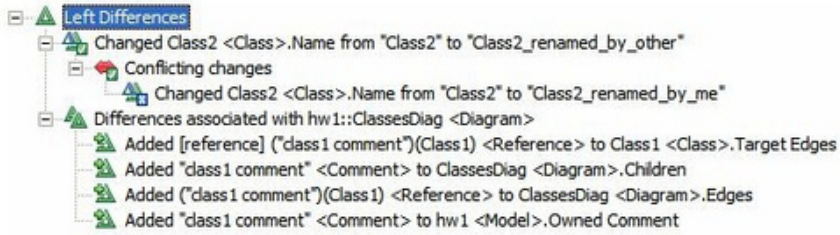
Figure 29. Reject command in the left contributor



Left and right differences panes

Moving along, you can now examine the Left Differences pane, detailed in Figure 30.

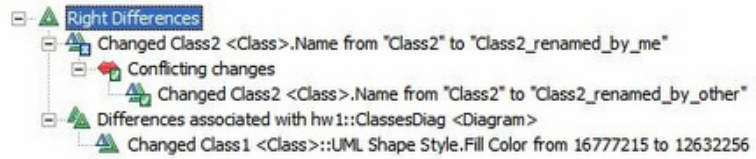
Figure 30. Left differences



Note that the conflicting name change is shown here as accepted, and there is an alternate view of the conflicting change in the other contributor, which is shown as rejected. You would normally leave the conflict collapsed in a differences view.

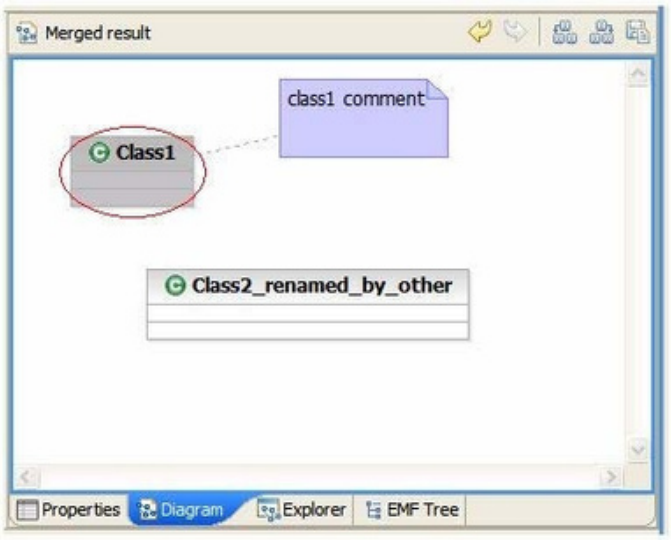
The differences associated with the diagram have been grouped here, and you can accept them all because there are no conflicts (you would see expansion plus signs if there were.) After accepting the diagram differences, move on to the Right Differences pane detailed in Figure 31.

Figure 31. Right differences



You made only one change on the diagram, and it does not conflict with any other change (again, noexpansion marker for a conflict) so simply accept it. The final result of the merged diagram is shown in Figure 32.

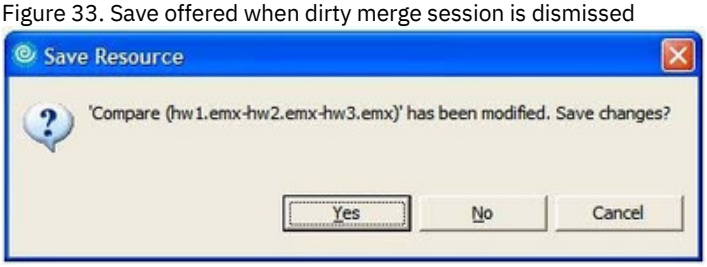
Figure 32. Final merged result



This is exactly where you ended up in the two-way merge scenario. But here all you had to do was accept all the differences from each side (pretty much without thinking), and choose which side you wanted to accept on the conflicts pane, and you were done. You'll save another copy of the merged result under a similar name.

When dismissing the compare session, you find out that the session is still considered *dirty* (unlike the two-way merge scenario, which acted more like a compare session.) You are offered the choice to save (Figure 33):

Figure 33. Save offered when dirty merge session is dismissed

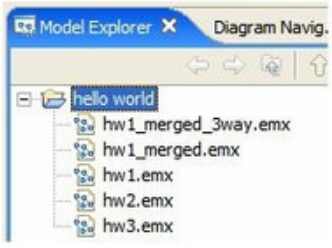


Clicking Yes launches a second dialog, as shown in Figure 34.

Figure 34. Choose where to save when dirty merge session is dismissed

The left and right contributors are both writable, so you are offered the choice of save targets. Since you have already saved a copy of the merged result, you decide to cancel the save operation here, *which dismisses the merge session completely*. The merge session would have continued had you cancelled at the *save choice* stage instead of the *target choice* stage. After refreshing your workspace, you'll see your new model (as illustrated in Figure 35).

Figure 35. Workspace contains both merged models



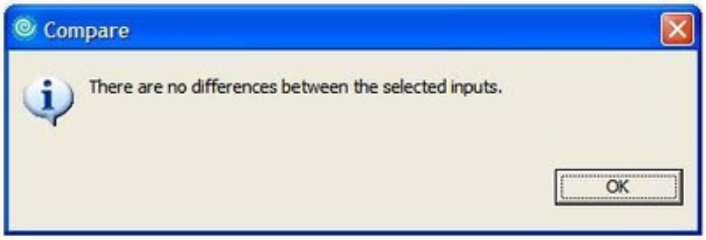
Final word

Two-way merging of changed models requires that you perform some mental gymnastics to figure out where the conflicts are, and to avoid losing your own changes (caused when a change exists only on the base side). Although two-way merging is very common with text, it does not scale well with structured data.

On the other hand, three-way merging clearly denotes every change, and attributes it accurately to the left or right contributors. All conflicts are also clearly indicated. The merge is straight-forward, and you need not worry about losing any changes once the conflicts have been resolved. *Three-way merging is the preferred merge mode for parallel development of models.*

One last thing: let's compare the two merge results to see if these two different techniques produced exactly the same final model. Figure 36 displays the conclusion.

Figure 36. Result of comparing the two merge result models with each other



Future articles will explore advanced merge topics such as merging multi-part models, composite groups and atomicity, in depth diagram merging, merging with ClearCase and CVS, and so on.

About the author

Kim joined IBM in 2003 with 24 years in large financial and telecommunications systems development. His responsibilities with the Rational Modeling Tools team include UML and EMF Compare Support, Architectural Discovery for Java and UML, Traceability, and Test Automation.