Comparing and merging UML models in IBM Rational Software Architect : Part 1

Comparing models with local history

Level: Introductory

Kim Letkeman (kletkema@ca.ibm.com), Development Lead, Modeling Compare Support, IBM Rational

12 Jul 2005

This is Part 1 of a multi-part article discussing how to compare and merge UML models in IBM® Rational® Software Architect. This article covers the *Compare or replace with local history* use case.

IBM® Rational® Software Architect (IRSA) and IBM® Rational® Software Modeler (IRSM) -- hereafter IRSA will denote both products -- are built upon the Eclipse integrated development environment (IDE), and inherit Eclipse's compare support work flows. IRSA UML models are themselves built using the Eclipse Modeling Framework (EMF), which essentially means that they contain complex structured data and cannot successfully be understood as text.

This being the case, merging EMF artifacts using the default Eclipse text compare support is very complex, and is unlikely to be successful. There is, in fact, plenty of empirical evidence to support the assertion that merging EMF and UML artifacts in text compare support is fatal to the merged artifact.

Since users of IRSA will still expect the standard Eclipse compare support to work, a custom EMF and UML compare support has been implemented. This is Part 1 of a multi-part article that focuses on how to compare and merge UML models in Eclipse. 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 any selected 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 this article.
- 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. This use case is covered in Part 2.

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.

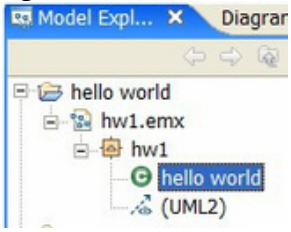
Reference: a pointer from one element in a model to another. The classic example is a class's view on a diagram pointing to the actual class object in a package in the model.

Compare with local history

The compare with local history command is useful for looking at the evolution of a model over time. Local history is a check-pointing system where each saved version is retained for an unspecified period of time. This feature uses temporary storage and is thus not to be relied on in production work flows.

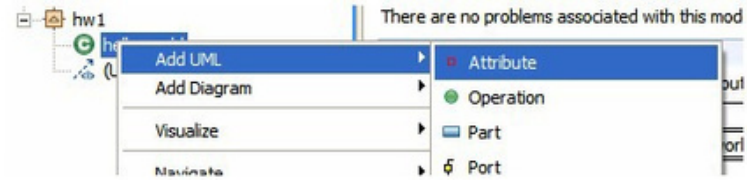
From here, you'll build a small model (Figure 1), and then add to it as new concepts are introduced.

Figure 1. Small Model



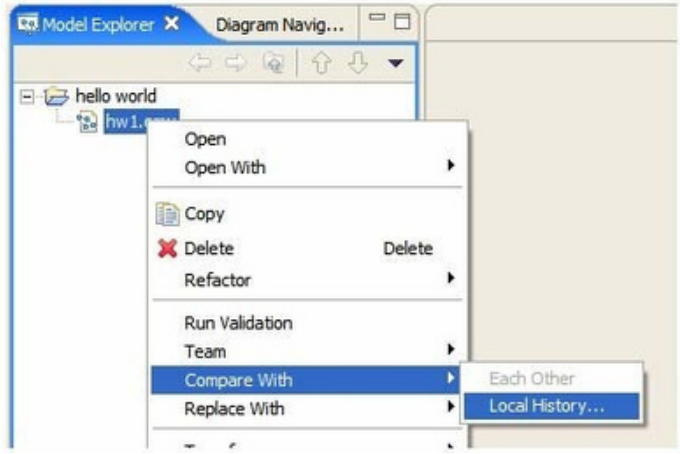
First, add a couple of attributes to the class (as shown in Figure 2) and save the model. Then add a couple of operations to the class and save it again.

Figure 2. Adding a UML attribute



You now have two prior versions relative to what is in the workspace. Select the resource and compare it with its local history, as illustrated in Figure 3.

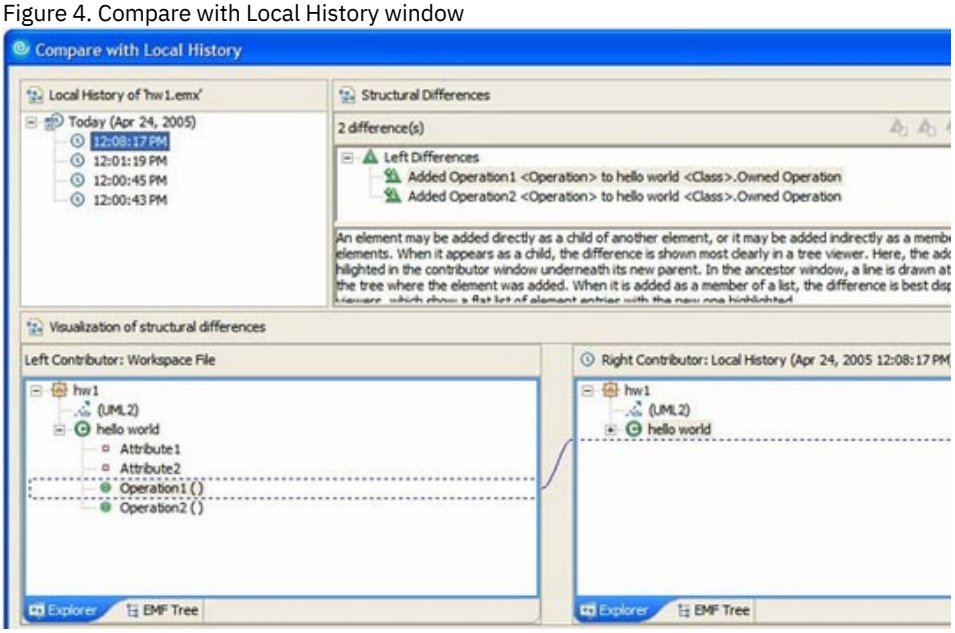
Figure 3. The Compare with Local History command on resource context menu



When you do this, it opens a modal window showing a two-way compare session above theEclipse workbench. You can select each

timestamp and see how that version of the artifact compares to the current workspace version. The current artifact state is automatically compared with a previously saved state. In the case shown in Figure 4, it's just before you added the two operations.

Figure 4. Compare with Local History window



The two changes you made in the latest version are clearly shown in the structural differences pane.

Explore the UML Model Compare window

Before going on, you should explore the compare session displayed for UML. In Eclipse, compare sessions are displayed in compare editors. There are several panes in a compare editor window. The following sections will discuss the various compare editor panes.

Structural Differences pane

Figure 5 shows the Structural Differences pane.

Figure 5. Structural Differences window

Left Differences in the top node denotes the left minus right orientation (execution logic) of the Eclipse compare session. The local version is placed in the left contributor window, while an older version (that is, stored in history) is placed in the right window. Eclipse then creates the delta list by subtracting the right-hand model from the left. Next, Eclipse opens the history list starting with the latest save operation, which in your example has two new operations. In the screen shot in Figure 5, the first addition is highlighted, which drives the highlighting of the affected elements in the contributor panes.

Delta formats

Delta formats are relatively obvious, usually denoting the following:

The *kind* of difference (for instance, Added)

The *element* on which the operation happened (for example, the new operation)
A *target location* (such as the class's owned operation collection)

An element must be attached to another element through a structural feature (attribute, reference, collection, and so on), which can have any name. The name is usually, however, something that implies a containment relationship. In this case that list is called *Owned Operation*, but could as easily have been called *Children* or *My Operations*. The feature name provides a bit of extra context to help you understand the purpose of the delta.

Other deltas look pretty similar. Delete deltas look like add deltas, except that they are *deleted from* a container instead of *added to* one.

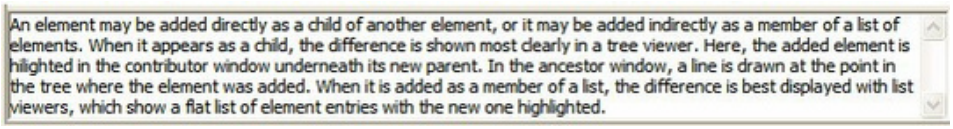
Moves go *from* a container *to* a container. Reorders are specialized moves that denote positional change within an ordered list.

Changes are a bit different, in that they have a target location that names an attribute, and they display the actual *before* and *after* attribute values.

Document pane

Figure 6 shows the Document pane.

Figure 6. Document pane with detailed explanation of an addition

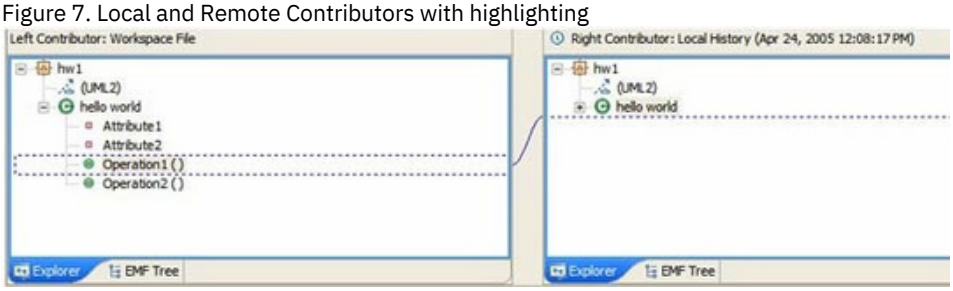


Each difference is explained in some detail in the document pane. In this case, the selected difference is an addition and the explanation tells you how an addition is represented in the model and on the screen.

Contributor panes

Figure 7 shows the Contributor panes.

Figure 7. Local and Remote Contributors with highlighting



The delta that was highlighted in Figure 5 (the Structural Differences pane) is also highlighted here in both Contributor panes.

Highlighting

The elements are highlighted differently, based on the action you perform:

An *added* element is highlighted with a rectangle around the added element in the local contributor pane, and a line under the element in the historical pane (which will become the new parent in later versions).
A *delete* is highlighted the same way, but with the contributors reversed.
A *move* is highlighted in the old position in the remote contributor and in the new position in the local contributor.
Changes (not structural changes, but rather attribute value changes) are shown in the summary that appears in the structural difference viewer. They are highlighted similarly to a move but without the position change (this is shown in more detail later.)

View modes

See those two tabs at the bottom of each contributor pane in Figure 7? They are called *view modes*, and they allow you to view the highlighted difference from different perspectives. In this case, only two tabs are present, and they are both structural tree viewers:

The Explorer viewer is virtually identical to the original editor in which the model was created.

The EMF Tree viewer is a representation of the physical underlying model and is useful only to expert modelers who need to see the guts of the physical model to help determine what is happening in error situations.

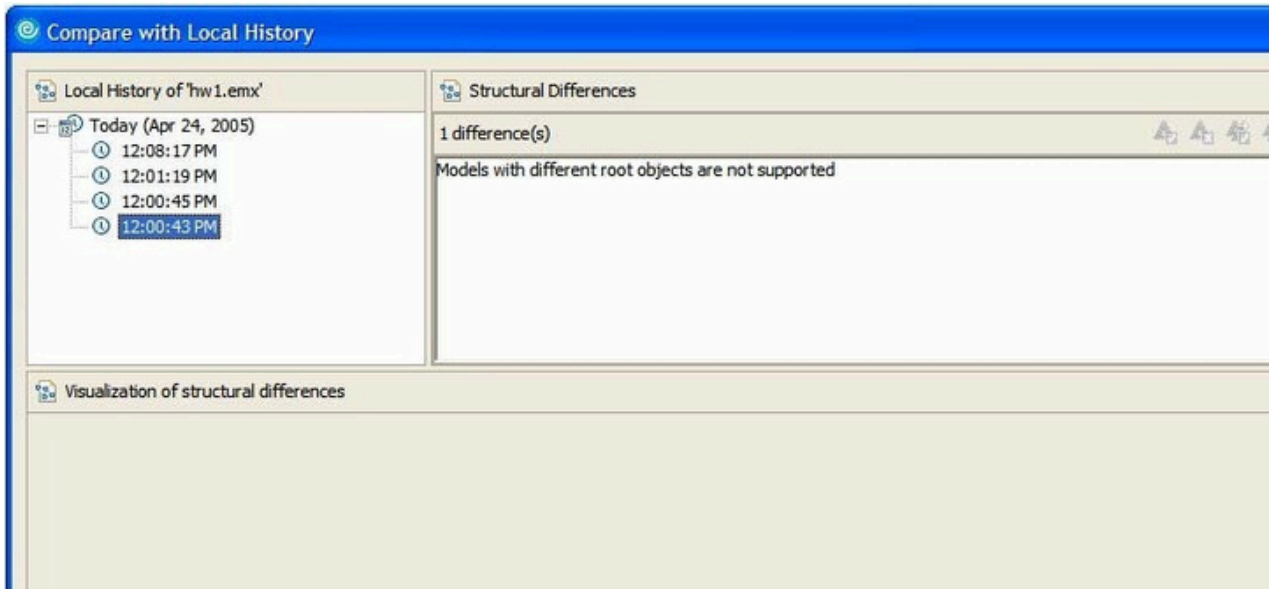
In this example and under normal circumstances, only the Explorer view would be used. It is always the default view mode when a structural difference (add, delete, move) is highlighted.

Switching view modes is as easy as clicking the mode you want in any of the visible panes (there are other panes for ancestor and even merged models in other compare modes). All of the appropriate viewers are slaved to each other, so the view modes will switch in tandem.

Explore the Local History versions

The *Compare with Local History* window shows changes between the current (local) version and the selected historical version. It does *not* show incremental differences between historical versions. Thus, you can only ever see the total list of local changes since the historical version was created. For example, selecting the first version of the artifact shows the information seen in Figure 8.

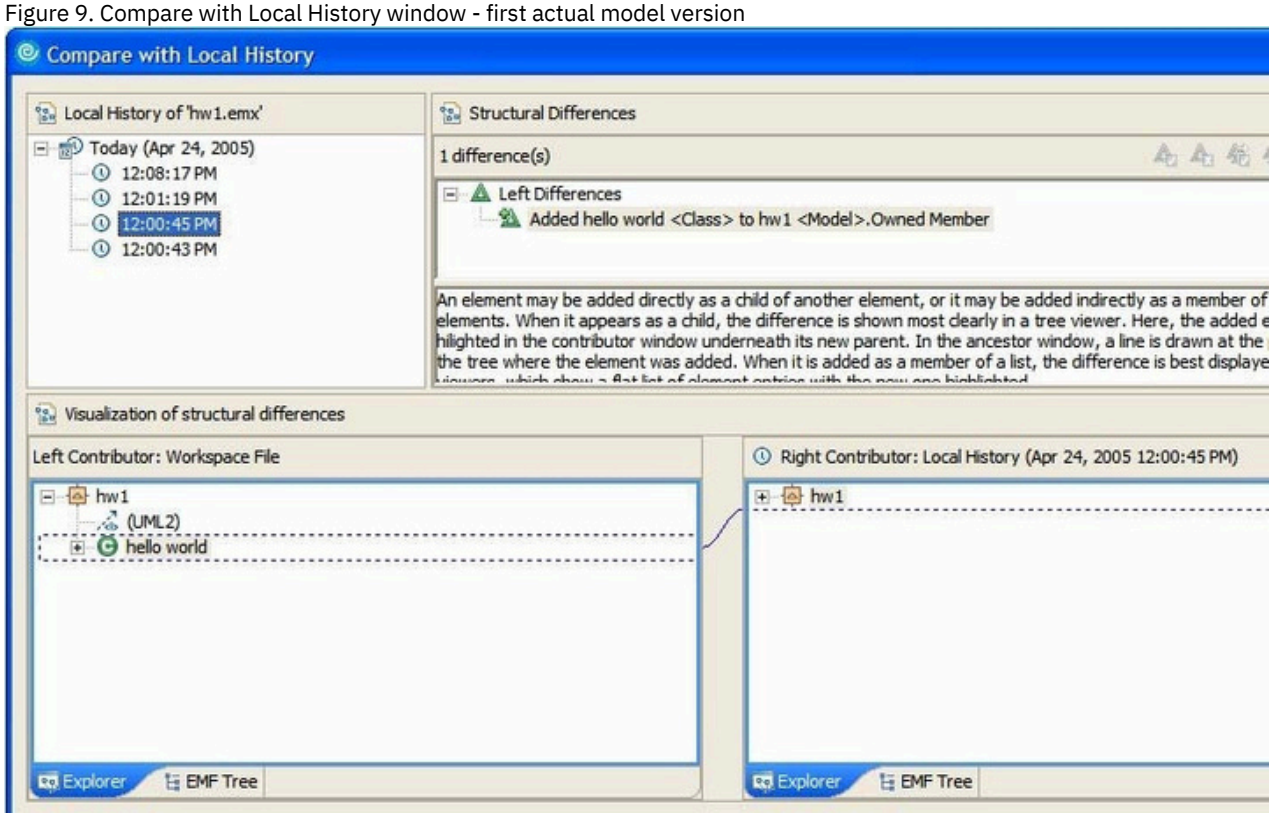
Figure 8. Compare with Local History window - first model version



Oops ? the first version is actually an empty model file without even a root node, saved by the new model wizard before it creates the model. Note that this is the message that you will see whenever you try to compare or merge incompatible versions. What this means in any other context is that the two models being compared do not have the same ancestor (that is, the root element identifiers are different).

Select the first actual model version as shown in Figure 9 and you'll see:

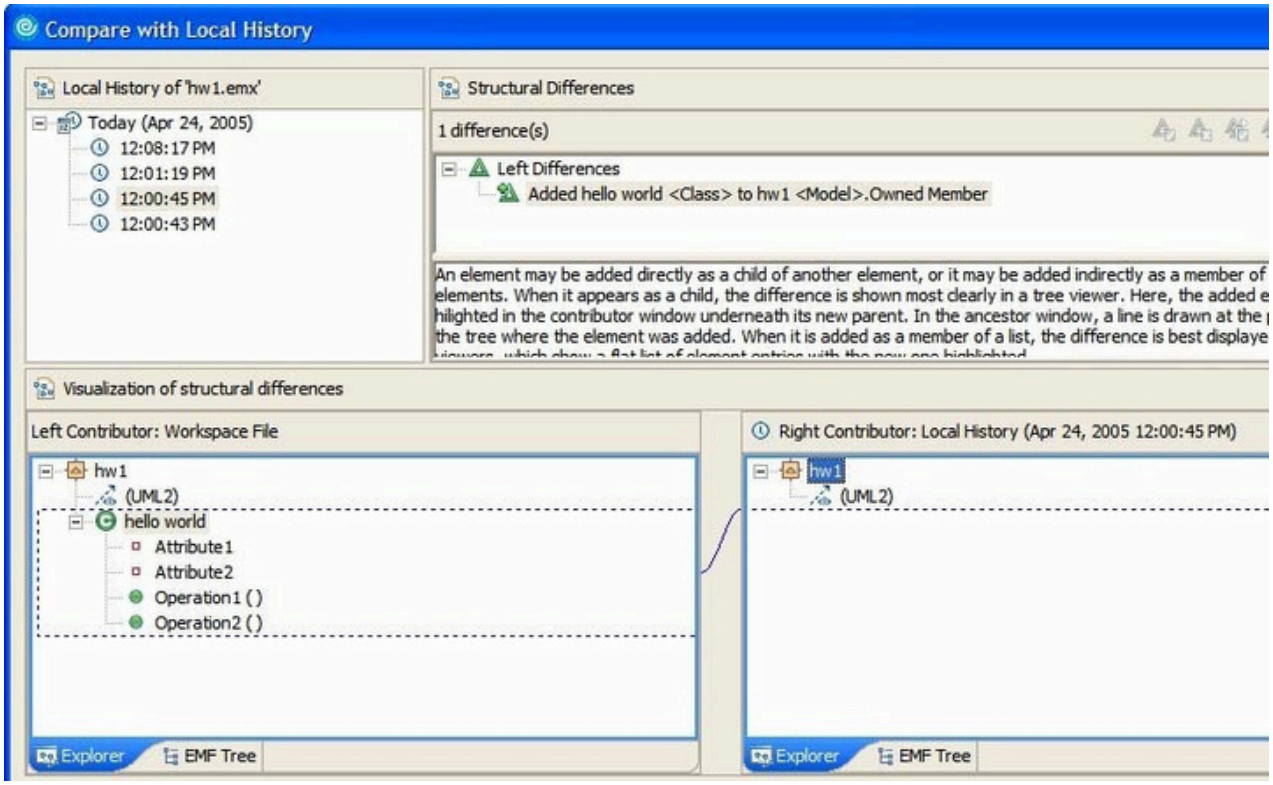
Figure 9. Compare with Local History window - first actual model version



The added class is highlighted in the local contributor, but why is there only one delta? You added the class and then a couple of attributes and a couple of operations in three steps. First, remember that this is the static difference between the first empty version of the model and the complete model with everything in it. All incremental changes have therefore been lumped together here.

A single delta is created because all of your additions are contained by the added class itself. In other words, it looks like a class with some contained members was added. This is more clearly illustrated by opening the models in the content viewers, as seen in Figure 10.

Figure 10. Compare with Local History window - models expanded



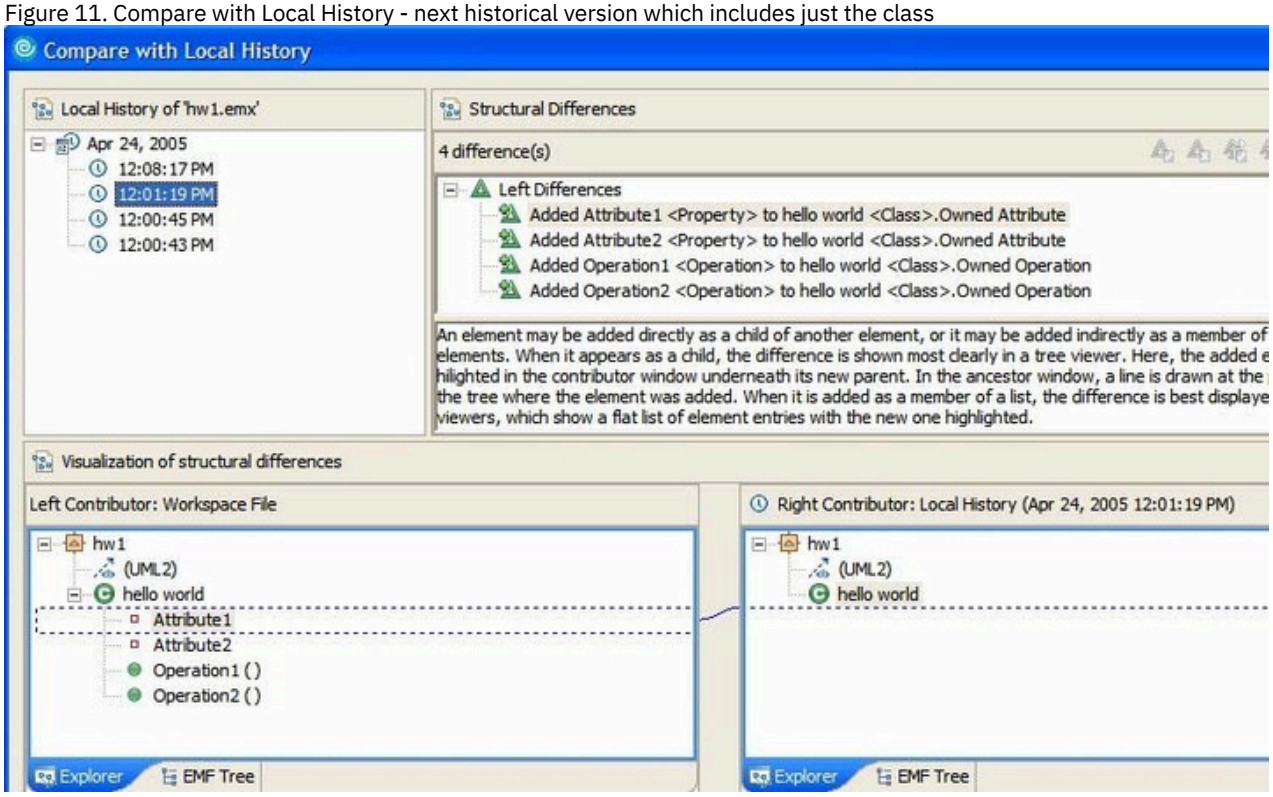
Additions are grouped by *containment*, so the class is shown as an addition and its children (the attributes and operations) are simply coming along for the ride. This can get a bit confusing -- clicking on versions going forward in time would intuitively show incremental changes from one version to the next (in other words, the static difference between two successive versions), but instead it shows smaller and smaller differences between the historical versions and the current local version.

This is not an issue when merging artifacts in a team environment, because the focus is always on final versions from multiple people.

The potential confusion arises here because your mental focus is usually on incremental change, so collecting the changes into larger containment bundles can be disconcerting. A mode to show incremental change between versions could be a useful addition to the local history feature in Eclipse.

Select the next most recent version to see the change in the listed differences. The next increment was the class addition, so you should see the addition of the content. This will be shown as four add deltas (see Figure 11) even though you never did four add deltas at once.

Figure 11. Compare with Local History - next historical version which includes just the class

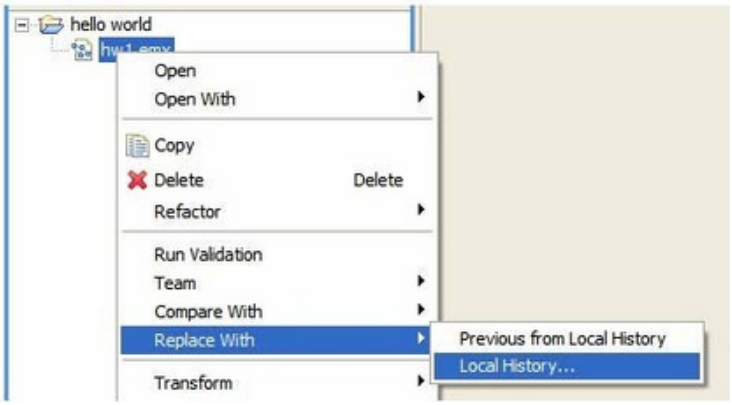


So you have four add deltas for the two attributes and the two operations. The first addition is highlighted in all panes.

Replace With Local History

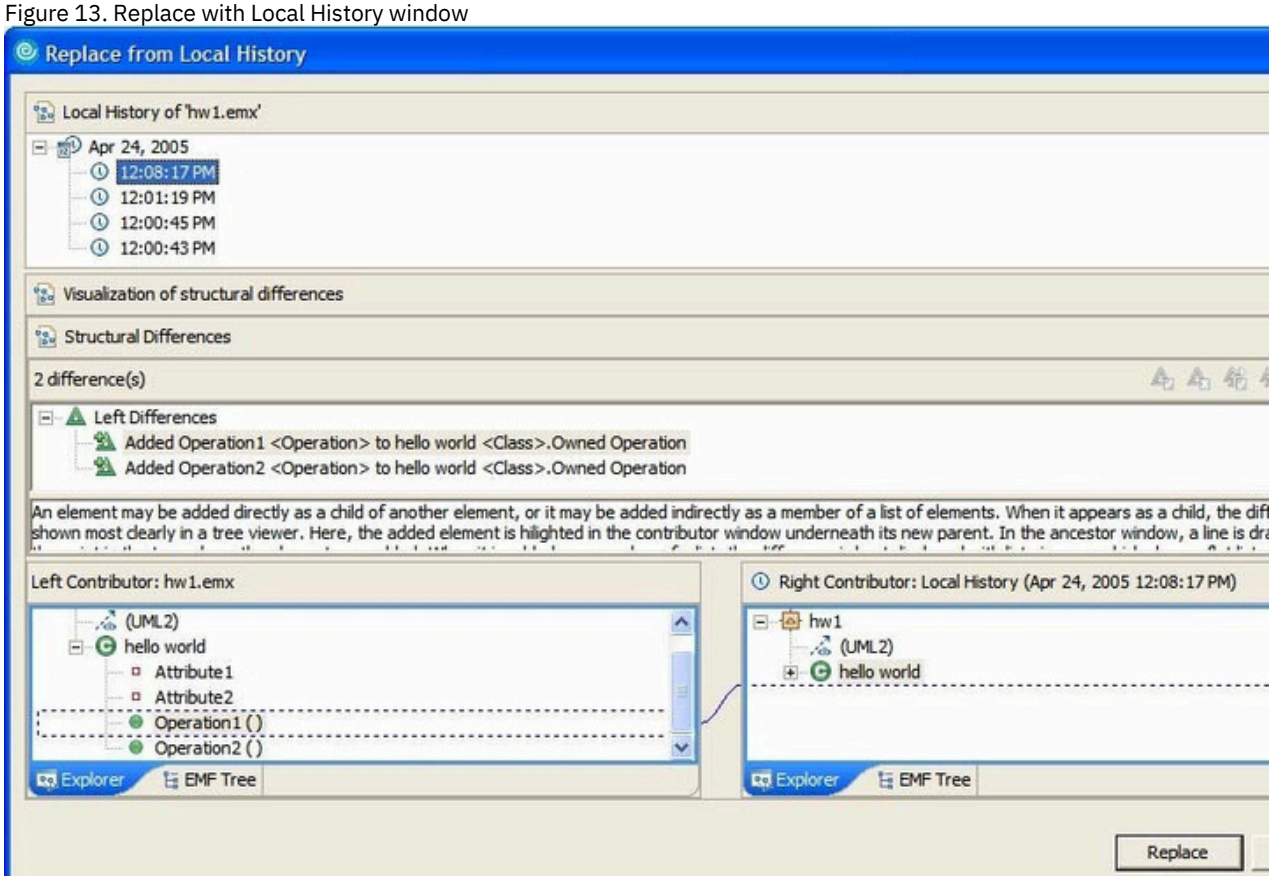
What if you want to revert to a version from your local history? Well, then you should be using the Replace With Local History variation of this command, as illustrated in Figure 12.

Figure 12. Replace with Local History command



The same modal dialog appears, but it has a slightly different layout (Figure 13). This happens because the Eclipse Replace support does not ask for a structural difference viewer, so it is included as part of the content viewers for model files.

Figure 13. Replace with Local History window

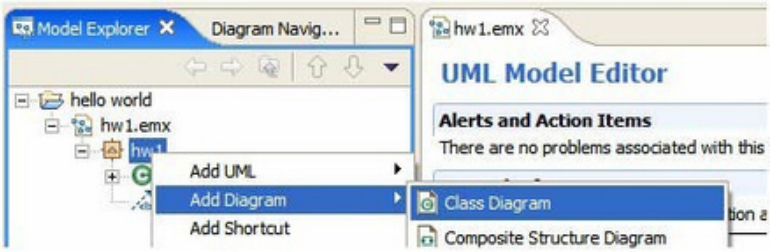


Once the required version is displayed in the right hand window, pressing replace willdeposit that version into the workspace, replacing the local version. If you make a mistake, it is easily fixed by using the Replace With Local History command again.

Diagram Compare

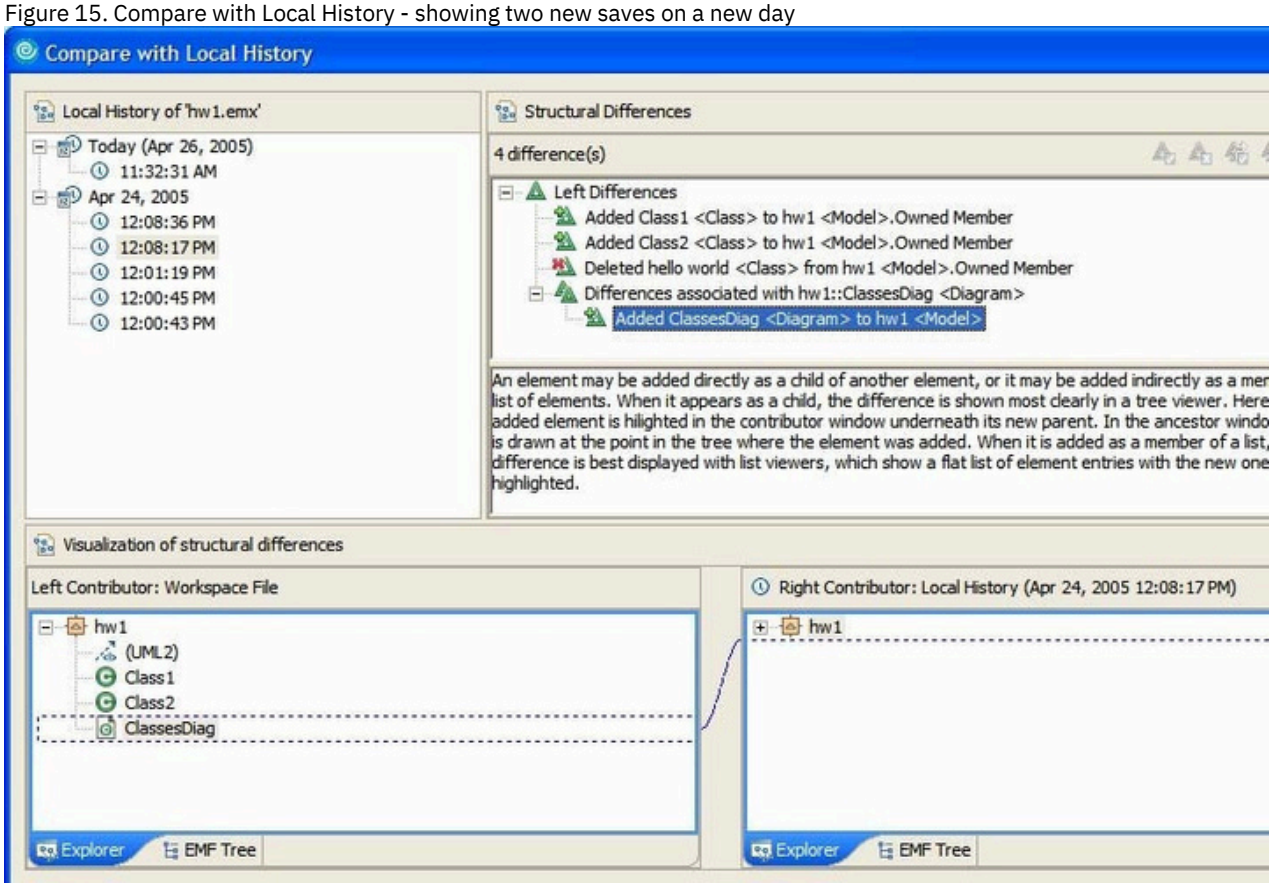
A unique feature of IRSA is the ability to compare and merge diagrams visually. To extend this example, you'll add a diagram (Figure 14) and save it. Next, you'll change the name of the diagram, delete your existing class, and add a couple of new classes right onto the diagram. This example will now help illustrate more view modes, diagram merging, and delete and move differences.

Figure 14. Adding the diagram



Once these two saves have been done, launch Compare with Local History again. It should look like Figure 15.

Figure 15. Compare with Local History - showing two new saves on a new day



Note in Figure 15 that a fifth saved version of the model (12:08:36) has appeared under April 24th. When you are working with a set of changes on a given day, the final saved version is *not* represented in the history version list, even when the in-memory model has changed. There are two issues to consider because of this behavior:

> The question "what have I changed since my last save?" cannot therefore be answered by comparing with local history.
>
> Comparisons start from the latest *saved* version, not the latest version.
> When you save a new version of the model, the version that appears immediately in the history list is the previously saved version, which may have been saved a long time ago.

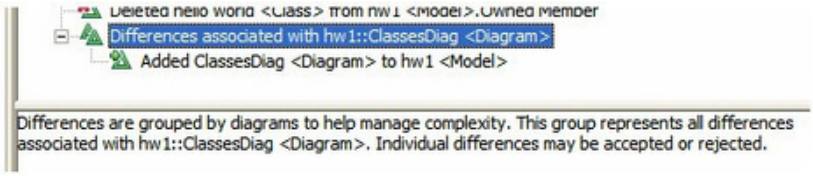
Be aware of these issues when using this feature.

In Figure 16, the last version from the 24th is selected and the deltas shown are exactly what you would expect. You added a couple of classes, deleted your original class, and added a diagram. The diagram is a container, so the diagram contents are not shown as separate deltas.

Composite difference group

You will note that there is a container for the diagram delta in the structural difference viewer. It is shown highlighted in Figure 16.

Figure 16. Composite difference group with explanation in Document pane



This is known as a composite difference group (or composite delta), and is used to reduce clutter in the user interface when multiple deltas are related in some way. In this case, all of the deltas on a single diagram are grouped into a composite difference group named Differences associated with <diagram name>. Associated with is used because differences that were the result of a change made to a diagram -- but which do not reside in the diagram's container -- can still be grouped with the rest of the diagram differences.

A classic example of this is a class that is added to a diagram. The semantic class element is added to the diagram's container (model or package) and a view onto that class is added to the actual diagram container. The reason for this split is that a class can be viewed in many diagrams (for example, one or more class diagrams, one or more sequence diagrams, and so on).

In your example, the composite contains only a single diagram delta (its creation), but the delta still appears in a composite difference group for user interface consistency. Other examples will follow in this and other articles.

Delete delta

You've already seen add deltas; figures 17 and 18 quickly illustrate the delete delta.

Figure 17. Delete delta highlighted in Structural Differences pane

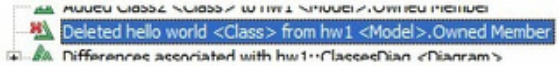
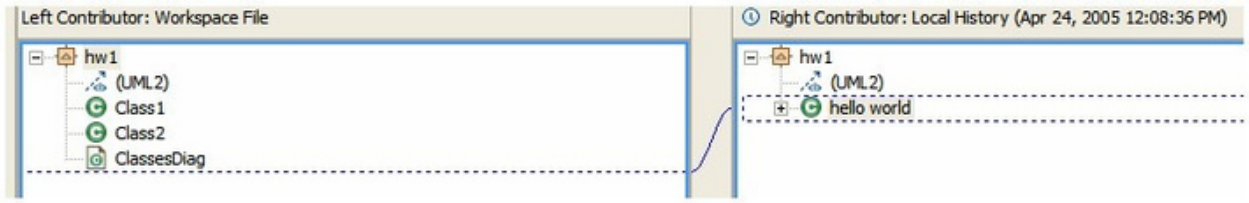


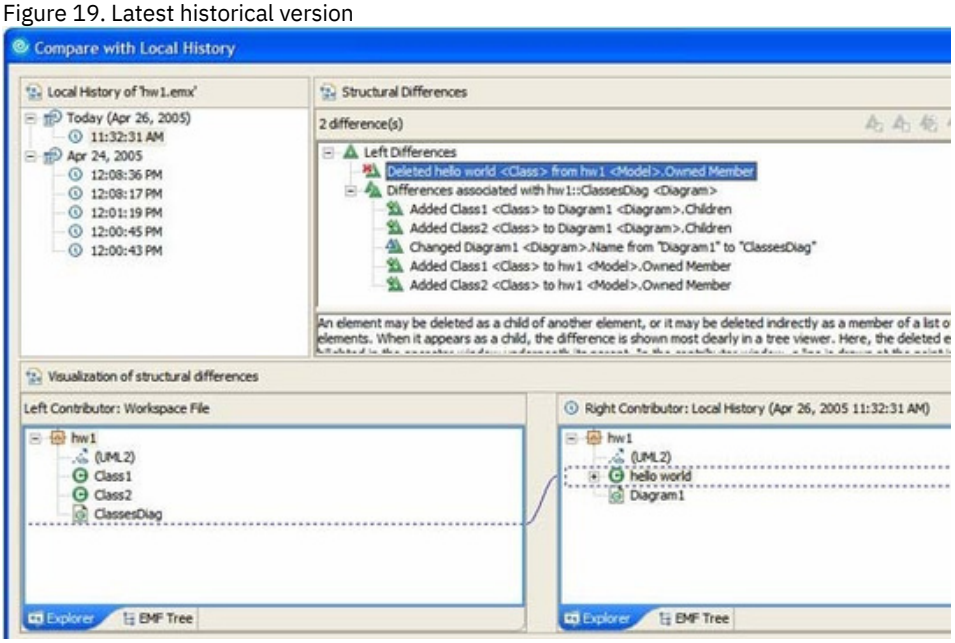
Figure 18. Delete delta highlighted in Contributor panes



View modes revisited

Figure 19 shows the latest historical version.

Figure 19. Latest historical version



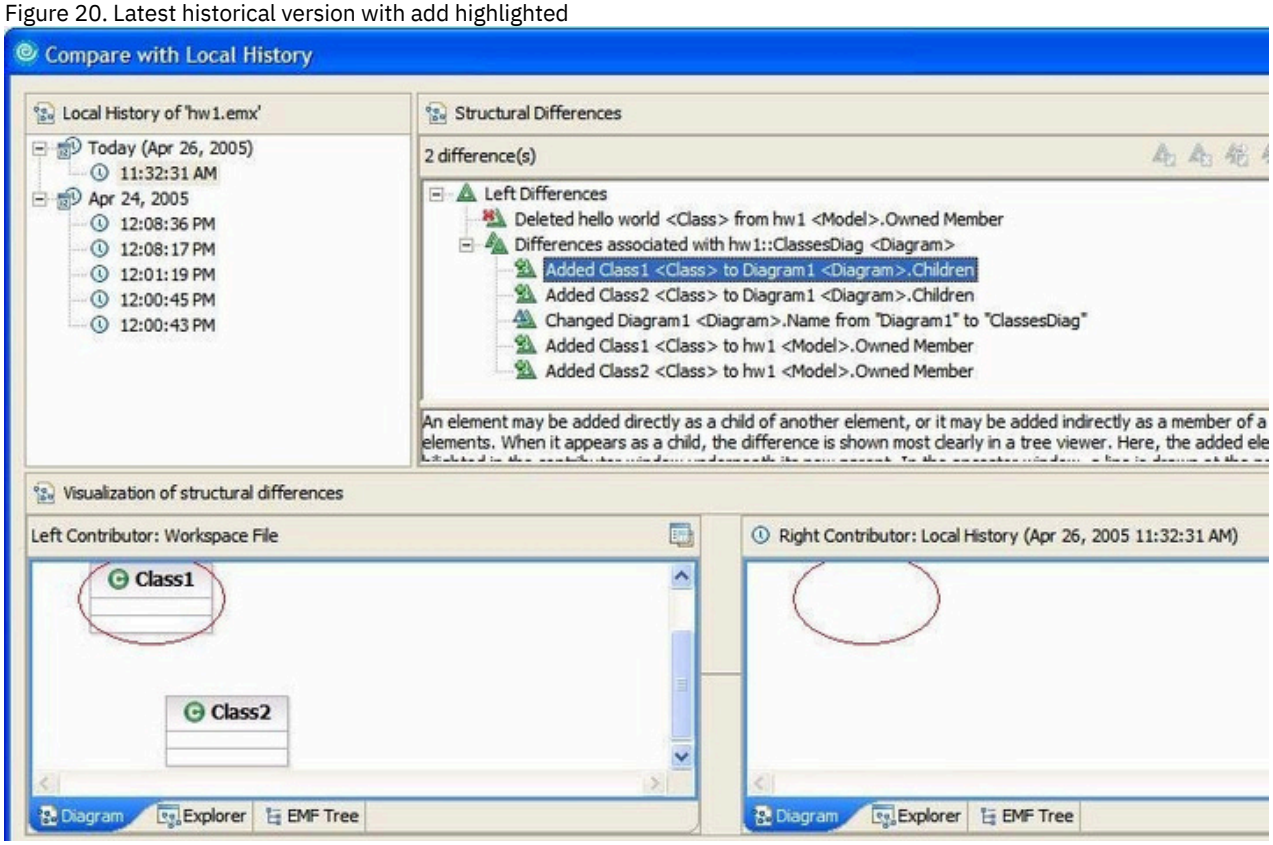
Note that the two added class instances are now shown inside the diagram composite instead of outside of it. This is because the added semantic class elements have been grouped with the added views on those elements. When looking at an earlier version, the new diagram and its contents were shown as a single add of the diagram container itself.

Remember that contained deltas are not broken out separately (because this could result in thousands of extra deltas). Instead, an added sub-tree is shown as one delta. So the first compare session with the new diagram shows only the diagram as a single added container. Once you compare the local version to a historical version where the diagram already exists, the added class view deltas are shown -- along with the class instance add deltas -- in the composite. This is a little complex, but the rule is basically that added diagram

elements (for instance, class views) have a strong affinity for their referenced semantic elements (for example, class instances) *if they were also added in the same version of the model*. Therefore, any pair like this will appear together in the appropriate diagram composite.

You can see that, as before, the default view mode is the Explorer, because thefirst delta is structural in nature. You'll see different view modes as you start navigating around the deltas. Click the first add delta inside the diagram composite, and you'll see something new (Figure 20).

Figure 20. Latest historical version with add highlighted



An added class view is highlighted. You can tell it is a view of a class and not the class instance itself because its new parent is a diagram (not the model or a package). Since it is inside a diagram, there's a new view mode available, as shown in Figure 21.

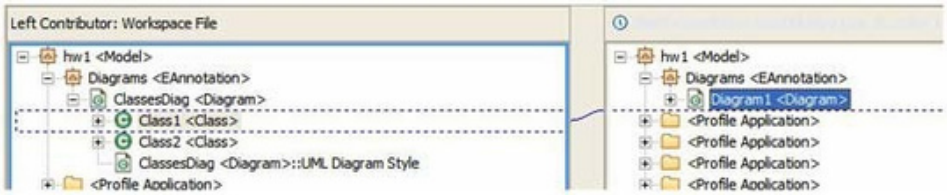
Figure 21. View modes available when diagram delta is selected

Conveniently, the diagram view mode is selected by default when a diagram delta is highlighted. As you can see, delta types are associated with view modes. Each delta can be displayed in one of several view modes, with the default view mode being the most precise. A diagram can also be shown in the Explorer tree, for instance, but it will only highlight the existence of a delta in a diagram (without any detail). For example, switching to the Explorer view mode while the class view addition is highlighted shows you only the information in Figure 22.

Figure 22. Parent element highlighted when detail is unavailable

Obviously, the diagram has been renamed by another delta. The contents of the diagram are filtered out by the model explorer, so this view mode can only default to highlighting the parent element on each side. The EMF tree view mode (Figure 23) shows you a bit more, with the added class view visible.

Figure 23. Class view highlighted



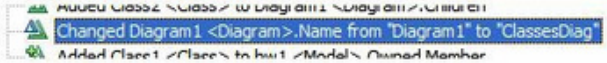
View mode affinity

Each delta type has a preferred view mode:

> *Add*, *delete*, and *move* deltas on elements always default to a tree view mode
> For *UML models*, the default tree mode is the Explorer view mode.
> *Change* deltas are attribute value differences, and the default view mode is a property view mode. This is always shown ina table format with the property name on the left and the value on the right.

This is easily demonstrated by selecting the name change delta, as shown in Figure 24.

Figure 24. Change delta selected



The view mode automatically switches to the Property view seen in Figure 25:

Figure 25. Change delta highlighted in Property view mode

You've already seen this change in the Explorer view mode, where the name change was quite obvious. But another property changed inside a diagram would not be as obvious in a tree mode, so this change defaults to the Property view.

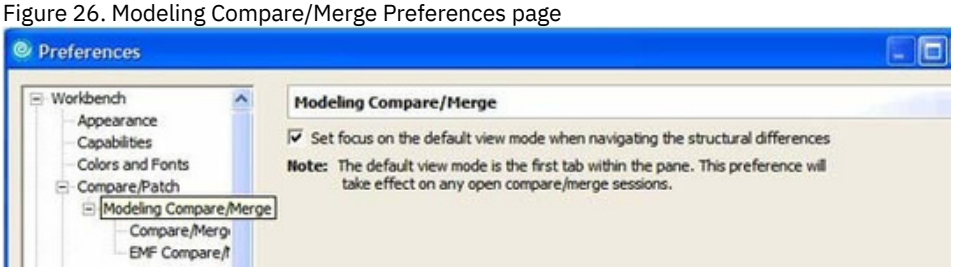
Navigation flicker

When navigating by selecting successive deltas (and delta types), there is a sort of flicker created by changing views. For instance, look at the six deltas in this example, clicking on each from top to bottom in the structural viewer. This should bring the view modes up in this order: tree, diagram, diagram, property, tree, and tree. This switching can get disconcerting.

Change view mode affinity from delta type to previously displayed view mode

If you would like to change the affinity so that the previously shown view mode is used if it can be, go to preference pages and follow the path shown in Figure 26

Figure 26. Modeling Compare/Merge Preferences page

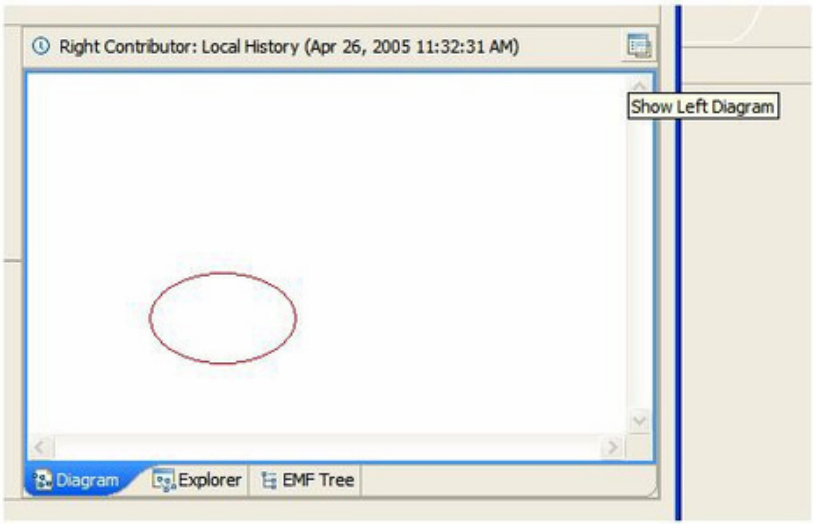


This preference is selected by default, enforcing the view mode to delta relationship. To change this, simply clear this value. Now, as long as both deltas allow the same view mode, the mode will not change as you move from one delta to another. The main value here is to allow property changes to be shown in diagram and tree view modes, which eliminates a lot of the flicker associated with navigating.

Diagram view mode navigation aids

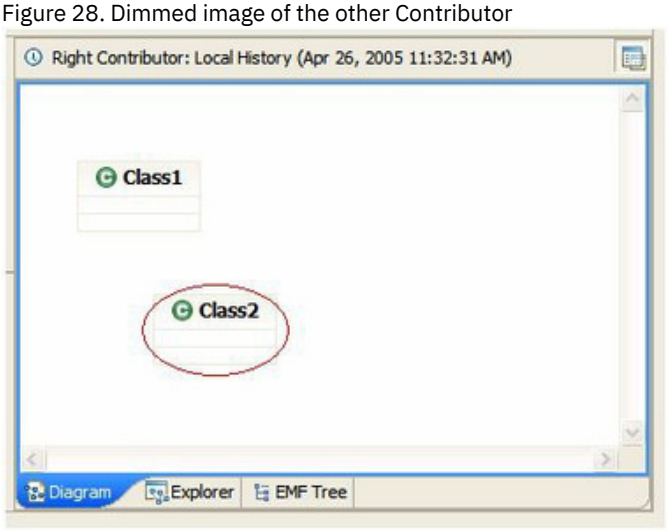
When working with diagrams, context can be difficult to discern at times. The windows are not huge, so any extra context can help. (Obviously, a large screen is recommended when working with diagrams, just as a large screen has always been recommended when working with CAD drawings.) In the top right-hand corner of each contributor pane in the Diagram view mode is a button that underlays a dimmed image of the other contributor beneath the current contributor (Figures 27 and 28). This can be useful when you're trying to figure out positional changes and the effect of add and delete deltas.

Figure 27. Right Contributor with Show Left Diagram tip showing



Pressing the button shows us the left contributor in a lighter color (dimmed).

Figure 28. Dimmed image of the other Contributor

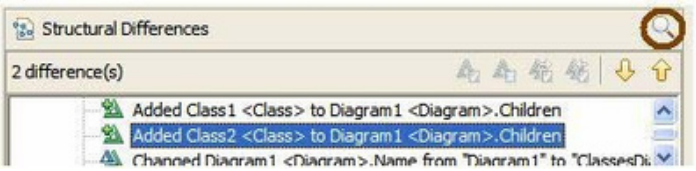


The two added class views are now clearly visible with the highlighted add delta circled.

Browse mode

Before ending, this article will discuss one final navigation capability: browse mode. The icon to start browse mode is on the top right corner of the compare editor itself, highlighted in Figure 29.

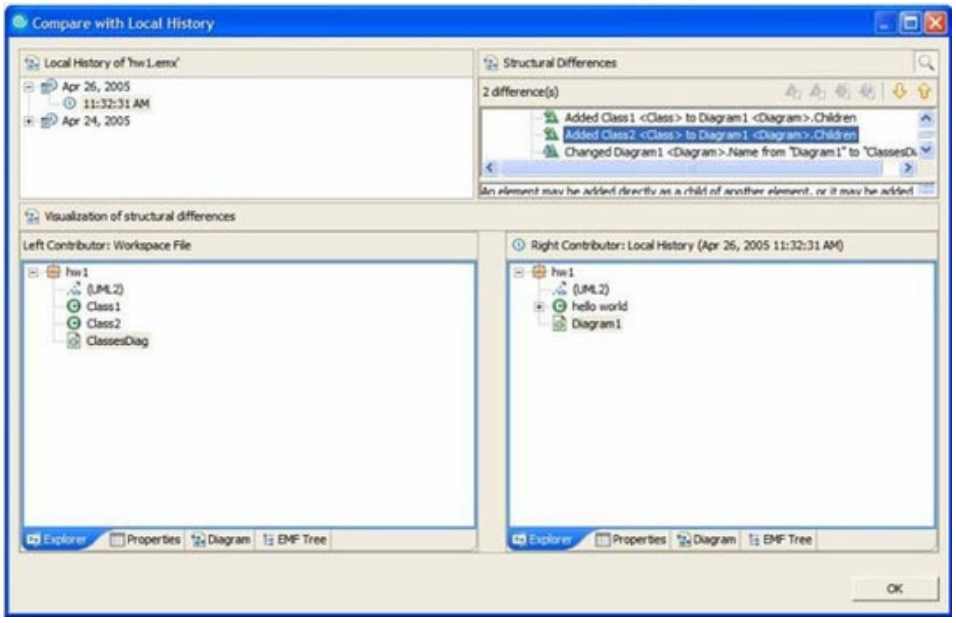
Figure 29. Browse mode button



The browse mode button is a toggle, so you are *either* navigating the deltas or browsing the model, but never both. This is very helpful if you are trying to understand deltas with deep context. When in browse mode, all view modes are available. Navigating the model in browse mode, you can either drag elements or right-click them to display a context menu.

The initial browse mode display looks like Figure 30.

Figure 30. Browse mode



To look at a specific diagram, you need only drag it to the Diagram view mode tab at the bottom of the same Contributor. You can also use the context menu, as shown in Figure 31.

Figure 31. Show in Diagram command

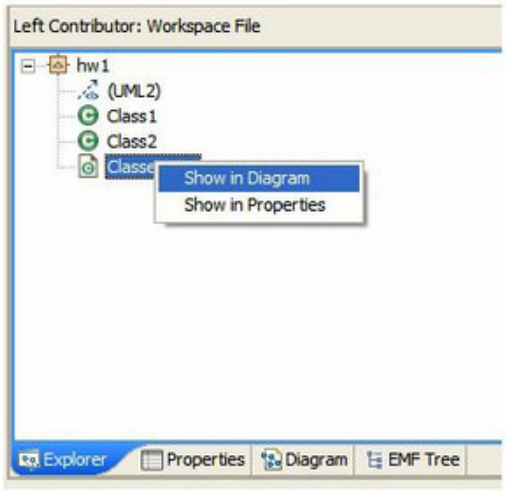
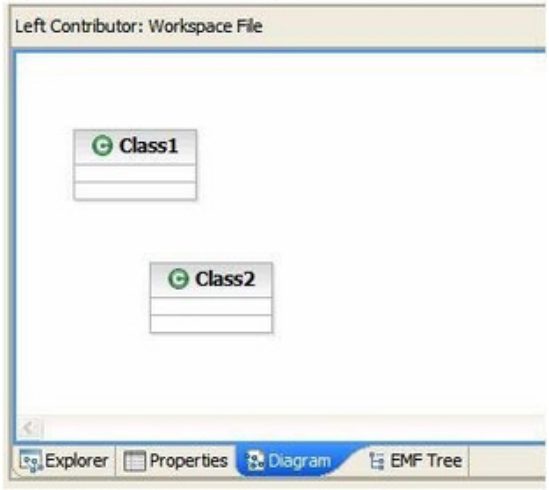


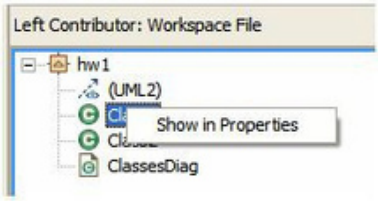
Figure 32 illustrates how, after executing this command, the contributor switches to the Diagram view mode and loads the diagram.

Figure 32. Switched to Diagram view mode



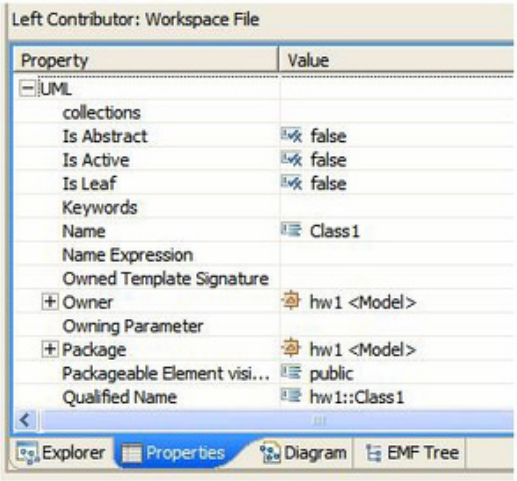
The other view modes are still perfectly valid. For example, you can switch back to the Explorer view mode and dig into the model without losing the information in the Diagram view mode (unless you load a different diagram into it). You can also select another element to load into the Property viewer without losing the Diagram view. For an example, see Figure 33.

Figure 33. Show a class in the Property view mode



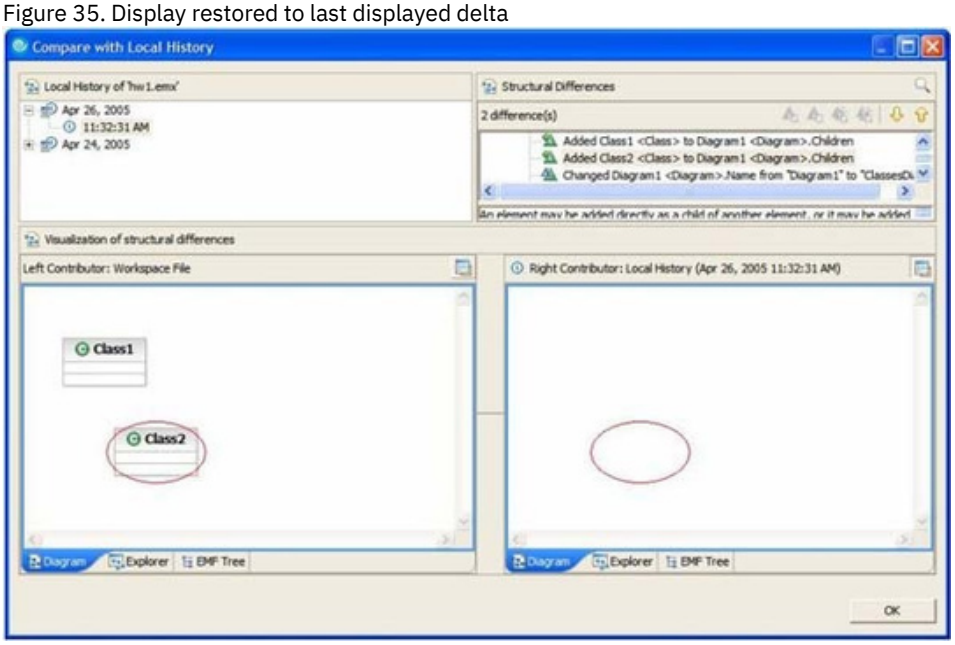
Executing this command switches to the Property view mode with the class's data loaded, as shown in Figure 34.

Figure 34. Show a class in the Property view mode



Rather a lot of data, but the point is that the entire model is open for exploring in browse mode without closing the compare session. To go back to delta navigation mode, simply toggle the browse button off and the display is restored to where you left off, as shown in Figure 35.

Figure 35. Display restored to last displayed delta



Final word

Comparing structured models is significantly more complex a task than comparing text, or even pseudo-structured data like Java. An easy way to get comfortable with the kinds of deltas that occur is to save the model after each structural change, and then explore the local history starting from the latest (fewest changes) and going backwards in time to see what new deltas show up at each interval. You can't isolate those deltas (no incremental display mode for history) but the new deltas should be fairly clear if you keep the changes simple between each step.

In the next article, we'll explore the Compare With Each Other command variation of Eclipse compare support. This will introduce

merging, three-way comparison, conflicts, accept and reject commands, 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.