Comparing and merging UML models in IBM Rational Software Architect: Part 4

Parallel model development with CVS

Kim Letkeman  August 02, 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 4 of a multi-part article discussing how you can compare and merge UML models in Eclipse using a custom EMF and UML compare support solution. This article covers parallel development with Concurrent Versions System (CVS).

Parallel development

Tiny teams building small applications have few issues when working in parallel. Each person typically manages a separate piece of the system, and there is no significant feature overlap. But as teams scale up in size and scope, overlapping responsibilities create parallel development situations.

Parallel development, in turn, introduces:

- The risk of overwriting other team members' changes during check-in
- Communication overhead for coordinating parallel activities
- A requirement to merge all parallel changes into a final version

Productive teams need to be free of the mundane management of artifacts so that they can better concentrate on their application domain, and specifically their models.

Parallel development of models is considerably more complex than parallel development of text artifacts like Java™ code. For example, when a merge error creeps in during a text merge, the fix is usually obvious by inspecting the merged file. Often, the compiler will flag the error accurately. Not so with modeling artifacts. UML artifacts are implemented on top of the Eclipse...
Modeling Framework (EMF), which leads to a great deal of internal complexity in the file syntax and semantic content. See Part 3 of this series for a detailed discussion of this complexity—see Resources. This additional complication makes it all the more important to clearly identify merge situations and to precisely manage interactions between parallel changes.

Software configuration management (SCM) systems like Concurrent Versions System (CVS) -- and IBM® Rational® ClearCase® (see Part 5 in this series) -- provide:

- Reliable artifact storage
- Artifact versioning
- Parallel change detection
- Multi-site and remote development support
- Release management with baseline labeling, and stream and branch management

This article will focus on parallel change detection.

More specifically, this article focuses on CVS workflows for parallel development. Other SCM systems will provide similar experiences, and their workflows are likely to be similar to CVS. ClearCase has a custom integration to IBM® Rational® Software Architect (IRSA) that is covered in another article in this series.

**Parallel change detection**

Parallel change detection works by flagging incompatible changes during workspace synchronization. The formula for this detection is, in fact, very simple: incoming artifact versions are compared to the latest checked-in artifact version (also called the remote version). If the remote version is the direct ancestor of the incoming version, then the check-in can proceed. If not, then a conflict is flagged and a merge must be performed. This dramatically reduces the potential for data loss. It also frees the team from manual coordination of parallel development. In fact, it is perfectly acceptable to depend completely on the SCM system to perform this coordination at the granularity of individual artifacts.

To reiterate: it is physically possible to do parallel development by manually coordinating and managing the merging of parallel versions. *Don't do that -- it does not scale!*

**IRSA integration with CVS**

The CVS team integration provided with IRSA uses the Eclipse team interface and extension points. CVS is thus able to interact with resources, offering Team commands (see Figure 1) such as checkout, checkin, branch, merge, and synchronize. The IRSA integration adds modeling viewers for structural differences and structured content (see Parts 1 and 2 of this series--Resources--for a detailed introduction to the modeling compare editors and viewers).
Each new version of the Eclipse platform carries a new version of the CVS integration. CVS users will typically work with the Resource Navigator and Team commands, the CVS Repository view, and the Team Synchronization view.

This article focuses on the Eclipse plugin client of CVS that ships with IRSA. The client is designed to communicate with repositories at URLs anywhere on the network. The CVS Repository model allows for excellent performance over the WAN or Internet. Projects are checked out directly into your workspace, so no intermediate copy exists. The basic workflow is shown in Figure 2.

Figure 1. CVS Team context menu

![Figure 1](image)

Each new version of the Eclipse platform carries a new version of the CVS integration. CVS users will typically work with the Resource Navigator and Team commands, the CVS Repository view, and the Team Synchronization view.

This article focuses on the Eclipse plugin client of CVS that ships with IRSA. The client is designed to communicate with repositories at URLs anywhere on the network. The CVS Repository model allows for excellent performance over the WAN or Internet. Projects are checked out directly into your workspace, so no intermediate copy exists. The basic workflow is shown in Figure 2.

Figure 2. CVS workflows summary

![Figure 2](image)

Each numbered flow works as follows:

1. **Check out** projects you will work on. You can also use the Eclipse *Team project set* import facility to accomplish the setup of your workspace.
2. Periodically **synchronize** your workspace, projects, or working sets. CVS will identify incoming, outgoing, and parallel changes for you in the *Team Synchronization* view.
3. Accept *incoming* files with the **Update** command, push *outgoing* files with the **Commit** command. Merge conflicts in the Team Synchronization view and identify merged artifacts.
using the Mark as Merged context menu item. The artifact's synchronization status will then change to outgoing.

**Connect to repository**

It is necessary to connect to a CVS repository in order to store your artifacts or work with others'. You are prompted for a repository location automatically when you try to share a project to CVS, or you can right-click the empty CVS Repository view and connect first. The CVS capability is not selected when a new workspace is in use, so before you can begin, you will have to go into Window > Preferences, then Workbench > Capabilities and select Team > CVS Support as shown in Figure 3.

**Figure 3. CVS capability**

Once you've enabled CVS Support, you can launch the CVS Repository Exploring perspective, as shown in Figure 4.

**Figure 4. Selecting the CVS Repository perspective**

Finally, Figures 5 and 6 show you how to set up the connection to a repository.

**Figure 5. Context menu in Repository view for new Repository Location**
Your CVS administrator can provide the repository location, as well as your username and password. If your machine and passwords are secure, then it should be safe to save your password for convenience during later operations.

Finally, if your project is new, then you will want to share it with other team members; sharing a project to a team provider is really saying that you want to deposit it in a repository somewhere. The Team > Share Project context menu item on the project resource will present you with the wizard shown in Figure 7.
Figure 7. Team Share Project dialog

The nature of this article suggests that you select CVS and click Next. This produces the dialog shown in Figure 8.
Figure 8. CVS Repository Location dialog

Since you have already connected once to your repository location, you can leave the default setting (Use existing repository location) and click Next. Figure 9 shows the next step, choosing a module name.
Figure 9. CVS choose Module Name dialog

You need to choose a module name for this project. In CVS, a module is a sub-section of the repository that can hold an individual Eclipse project, or a component consisting of multiple Eclipse projects. Your CVS administrator will have chosen a module strategy, which you will apply in this step. For this example, let the default stand again, using the project name as the module name. Figure 10 shows the next step of the wizard, where resources are committed to the repository:
You now commit the project to the repository and click **Finish**. A few more dialogs ask you to commit changes and add a comment, after which you'll have a newly shared project in CVS, as shown in Figure 11.

**Figure 11. Shared project in workspace**

Now that you have shared projects in your repository, you can begin working with the classic parallel development workflow, which is shown in Figure 12.

**Classic parallel development workflow**

Now that you have shared projects in your repository, you can begin working with the classic parallel development workflow, which is shown in Figure 12.
Figure 12 shows two users (A and B) checking out the same version (1) of an artifact. The users make changes to the separate artifacts, with user A committing a changed artifact first, creating version 2. When user B checks in further changes, CVS parallel change detection marks the file as a conflict. (In ClearCase, the merge would proceed automatically with the user then prompted to check in the merged artifact.) User B completes the merge and commits the file to the repository, creating version 3.

**Best practice -- frequent synchronization**

If you work with a workspace for a long time between synchronization with the repository, you run the risk that you will modify a fairly old version of an artifact, which may force you to re-merge a large number of previously merged changes. This can happen when an artifact has a burst of activity, with several change sets submitted over a short time frame. If you fail to synchronize, you will end up modifying a version that has none of these changes. When you then go to commit the file, CVS will see your version as parallel with all of the modified versions between your last synchronization and the latest one in the repository. When you launch the merge, CVS will load the ancestor (the version that was in your workspace the last time you synchronized), the remote file containing all of the previous changes, and your file. You will be forced to deal with all of the previous changes as competition with your changes, a potentially complex process. Had you simply synchronized your workspace before modifying the artifact, you may have completely avoided any merging at all. In a nutshell ... your baseline must advance frequently in order to minimize the scope of or entirely eliminate merges.

This is the simplest form of parallel development, basically a checkout and checkin by multiple people on a single artifact starting at the same ancestor version. It easily scales to handle as many parallel versions as necessary; as each new version in checked in, there is always a latest (remote) version and there is always a common ancestor. Each parallel version will therefore be the result of a merge of all previous changes with the new change set. This workflow scales to handle arbitrary artifact change sets, because each artifact's history is processed independently of the others. That is, the common ancestor is selected each time and can change, depending on where each version in the merge descended from. The SCM tool handles these interleaved change sets automatically, with no extra thought or coordination required by team members.
To extend the above example -- each additional work flow looks exactly like that for user B so I won't add another figure -- suppose that versions 1c and 1d are checked out and changed in parallel with the 1a and 1b working versions, giving four parallel versions. When version 1c is checked in, a second merge is launched automatically with version 1 as the common ancestor and version 3 as the remote contributor (containing the previously merged changes in working versions 1a and 1b.) When completed, the final version is checked in as version 4. Then, when version 1d is checked-in, yet another merge is launched with version 1 as the common ancestor and now version 4 as the latest (remote) version. The merge result, now containing all changes from versions 1a, 1b, 1c, and 1d is checked in as version 5.

The point here is that there is no limit to the number of parallel versions that this cascading work flow can handle, because each checkin creates the next remote version to be used as input to the next parallel merge session. The tool has therefore taken on the entire artifact and work flow management burden, which is a huge productivity booster in large teams.

To continue with this example, simulate two parallel users by checking out another copy of the project. To do this, go into the repository view and use the context menu on our shared project to select the **Check Out As** command, and then select a name for the second copy of the project, as shown in Figures 13 and 14.

**Figure 13. Check Out As command**
Switching back to the modeling perspective, you now have two copies of the project to manipulate in parallel. The model artifact is shown as version 1.1 in both, which is what you would expect for a new project. Figure 15 shows your two parallel projects:

**Figure 15. Parallel projects in the same workspace -- a neat CVS trick**

In order to work on these in parallel, you can simply start changing them. Add one new class to each (see Figure 16) just to keep things simple.

**Figure 16. Parallel added classes**

Commit the first user’s changes to the repository. If there were a parallel version already committed, this would fail. Of course, it does not, as shown in Figures 17 and 18.
You now have version 1.2 in the repository. Now try to commit the simulated second user's changes. There is a parallel version now, and the result is shown in Figure 19:

The error text is pretty nondescript, but you can infer that a parallel version has already been committed. You can prevent this ambiguous situation by always synchronizing your resources with the repository and letting CVS be more explicit about the state of your workspace versus the state of the repository. Do that now, choosing the obvious (but incorrect) way by using the Synchronize with Repository command on the project (see Figure 20.) This is actually a mistake, but continue a bit longer to demonstrate the effect of this choice.
Figure 20. Synchronize with Repository command on the second user's project

Issuing the Synchronize command directly on the artifact displays a merge session (see Figure 21) immediately.

Figure 21. Synchronize Results in a Merge

Note that this is a three-way merge, with user 2's local file on the left and the remote file on the right. There are no conflicts, which you can see immediately because you were not put into the conflicts tab in the structural viewer. You can accept the delta in the left differences tab, then switch to the right differences tab and accept that delta, and you end up with both new classes in the final result. Save that over the left contributor. See Figure 22 for the result. In CVS the left contributor is always the local workspace file, which is always the merge target, which makes it the file you will check in shortly. CVS merge flows are very similar to those for the Eclipse compare with each other merge techniques described in detail in Part 2 of this series.
Figure 22. Merge result is that both classes are added

![Merged result](image)

Closing the merge session leaves the merged artifact sitting in the workspace, but committing that artifact brings up the same error! CVS has not yet been told that the artifact conflict has been removed. Mark the artifact as merged so that CVS understands that the latest repository version has now been successfully integrated into the local version. This is normally done in the team synchronization view, but that view does not automatically show up when a single project is synchronized; remember that the merge displayed immediately. **Synchronizing successfully requires that you launch the team synchronization perspective first.**

Open that perspective (see Figure 23) in order to be able to finish this task.

Figure 23. Open Perspective dialog

![Select Perspective](image)

With the merge still in progress, the state of the perspective does not get set up correctly, and the left pane is blank (see Figure 24.)
Figure 24. Team Synchronize command

To correct this, close the merge session and rerun the synchronization from its own perspective (see figures 25 through 27.) Having already saved over the local artifact during the previous merge attempt in this example, you'll need to remember that the merge has been completed.

This is a fundamental difference between ClearCase and CVS -- CVS uses a target-based merge approach where the local file is the merge target and is altered permanently by the merge. ClearCase uses a fourth file to receive the merge result, so cancellation or failure of the merge process leaves the workspace as it was before the merge started. Mistakes in the merge process with CVS can easily be corrected by using local history, a fact that is obvious to long-time CVS and Eclipse users. Long-time ClearCase users should be aware of this difference in work flows.

Figure 25. Choose CVS synchronization type

Clicking Next brings up the Select Resources dialog (Figure 26) so you can choose which projects to synchronize.
When you click **Finish**, the result (see Figure 27) displays very quickly, which seems kind of anticlimactic.

**Figure 27. User 2 synchronization result**

The double arrow decorators indicate the presence of a change conflict, which in CVS really means the presence of parallel versions of an artifact. CVS Team synchronization is a manual process that requires that the user resolve each difference before committing to the repository. Since you already know that there are no conflicts in the models' content (because you have already completed this merge previously), you will choose *not* to double-click on the artifact conflict entry to launch the merge. Instead, you will immediately mark the artifact as merged (see Figure 28) to tell CVS that this version contains all the changes from all parallel repository versions. In CVS, these steps are separate. You can run the merge as often as you want (with the caveat that your local file will always contain the result of the previous merge session so each merge session will be different.) Once you have decided that the artifact is completely done, you issue the **Mark as Merged** command separately.
The decorator changes from a conflict arrow to an outgoing arrow, indicating that you may now commit the artifact to the repository.

At this point, click **Commit** on the artifact’s context menu in the Synchronize view, and the result is a successfully committed merge result. The repository view changes its state to indicate that the project and repository are now synchronized, as shown in Figure 30.

Switching back to the Modeling perspective, you can see now that each user’s workspace contains a different version of the artifact. The merged version was committed by user 2 as version 1.3, but user 1’s previously committed artifact is still sitting at version 1.2.

User 1’s project must now be synchronized with the repository to pick up any recent changes. The result is shown in Figure 31.
User 1 is reported as being unsynchronized. The incoming arrow indicates that a newer version exists in the repository. Since you have not changed your version lately, there is no conflict and you can simply update to the newer version. Right-clicking the incoming version gives you a context menu with the **Update** command, which you select. Version 1.3 appears in user 1's workspace, and both users are now synchronized.

**Final word**

A few notes on IRSA and CVS:

- **Do not** change modeling artifact types to *text* in the CVS preference pages; text merges will **ensue**, *causing almost certain corruption of your artifacts*. Modeling artifacts are to be treated as binary by CVS (which is the default); this means that they cannot default to text merges and thus cannot be destroyed accidentally.
- CVS will clearly indicate the artifact's state relative to the repository. A left pointing arrow indicates an incoming change, a right pointing arrow indicates an outgoing change, and a double ended arrow indicates a conflicting change, which requires a merge. The state is updated in real time as the artifact is operated on. Use the Team Synchronization perspective to keep your workspace up to date.
- CVS uses optimistic locking of artifacts. That is, the lock is obtained at the last possible moment and a commit failure is a valid result. Checking out a resource in CVS is really just an automatic import from the repository to the Eclipse workspace. Once you have the project in your workspace, you have carte blanche with its artifacts. CVS will iron out changes and conflicts during synchronization.
- You can override an artifact in either direction at any time, updating it from or committing it to the repository.

Parallel development in CVS always boils down to variations on the themes explored in this article. Once you have checked out the files you want to work on, you can modify them with no further communication with the CVS repository. You can synchronize with the repository at any time in order to update the state of your files or deliver your changes to the master sources. CVS will perform all of the necessary parallel change detection during the synchronization step. When merges are necessary, CVS will calculate the correct remote and ancestor artifact versions. Parallel versions can therefore go to any depth as CVS will see parallel changes right up to the last possible moment.

© Copyright IBM Corporation 2005
**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)