DevOps Model RealTime
and
RSAD EGit Integration

Logical and closure merge from command line

Author: Sergey Eroshkin *HCL*

Revision history

Version	Revision Date	Summary of Changes	Who
1.0 Draft	March 2015	Initial version	Sergey Eroshkin
1.1	May 2016	Updated for 10.0	Mattias Mohlin
1.2	September 2016	Updated EGit version to 4.4.1	Elena Strabykina
1.3	August 2018	Updated for 10.2	Mattias Mohlin

Contents

1 Introduction....4
1.1 Definitions....4
1.2 Git customization limitations....4
1.3 Overcoming EGit limitations for modelling tools....4
2 Installation....5
3 Common principles....6
3.1 Eclipse Team API....6
3.2 Workspace configuration....6
4 Merge operations....7
4.1 Logical merge from command line....7
4.2 Closure merge....11
5 Compare operations....14
6 Automatic project import....17
6.1 Custom script for launching....17
6.2 Project import filters....17
6.3 Using extension point to define custom project importer....17
6.4 Commit analysis....18

1 Introduction

This document describes the command line API of DevOps Model RealTime for invoking logical or closure merge for models stored in a Git repository. The implementation is based on Eclipse EGit/JGit plugins and the Model RealTime Compare/Merge command line API (see the document “Compare/Merge command line tool for Model RealTime and RSAD”).

1.1 Definitions

We will use the following terms to define artifacts managed by Model RealTime:

- **File**: model root (*.emx) or fragment (*.efx) or some other file (e.g. transformation configuration file)
- **Logical model**: a set of files with containment relationship having single root file. In Model RealTime this corresponds to model file (*.emx) with set of zero to many contained fragments (*.efx)
- **Closure model or just closure**: an arbitrary set of logical models.

These artifacts define several possible compare/merge scenarios:

- **file-by-file** compare/merge: files are processed independently of each other, one by one
- **logical model compare/merge:** files are grouped into logical models and each logical model is processed as a single entity from compare/merge point of view
- **closure compare/merge**: files are grouped into one or multiple closures and each closure is processed as a single entity from compare/merge point of view

Some EGit/Git terms:

- **Merge Tool –** graphical application to resolve conflicts that have resulted from a previous merge operation. In the context of EGit this is the command which starts the Merge editor on conflicting files.

1.2 Git customization limitations

The plain console Git allows to define a custom diff/merge tool. Unfortunately there is no way to define a diff/merge tool that would process a set of files as a single entity. It is therefore not possible to implement an extension to Git to perform logical or closure compare/merge operations as part of **regular** git commands (like merge, pull, rebase, etc).

Thus the support for logical and closure compare/merge requires customized commands that can be aware about models.

The approach in Model RealTime is to implement a solution based on the Eclipse EGit/JGit projects, taking into account what is specific about Model RealTime models.

1.3 Overcoming EGit limitations for modelling tools

EGit is an Eclipse Team Provider for the Git version control system which is based on a Java implementation of Git (JGit). The current latest public EGit release does not fully support modelling tools. It always uses plain text merge for files and hence could easily cause corruption of models when merging them.

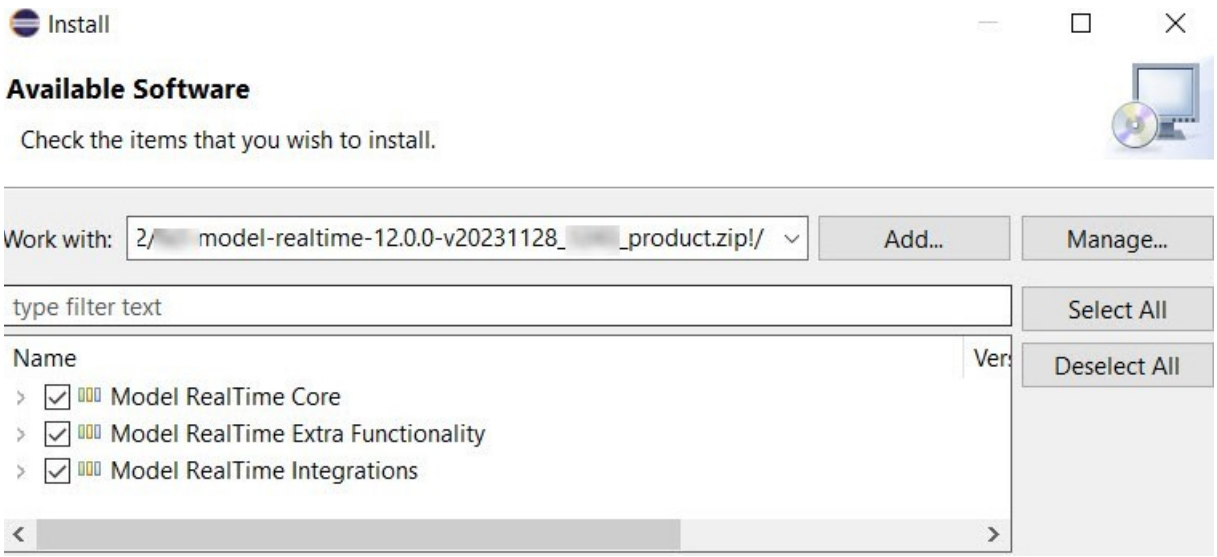
Work was initiated to make EGit aware about modelling tools. However, at the time of writing the result of this work has not yet been fully incorporated into the standard EGit. Also, the proposed additions still don’t cover all required scenarios from Model RealTime point of view. The main limitations are missing support for closures, lack of extension points and inappropriate handling of addition/deletion of logical model fragments in some cases.

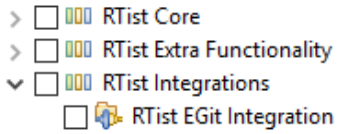
The Model RealTime EGit integration is therefore based on the Compare/Merge command line API, the Eclipse Team API, the latest public EGit release (with some ideas taken from the ongoing work mentioned above), the previous integration with ClearCase TeamExplorer and it also takes into account some specific Model RealTime cases.

The Model RealTime EGit integration provides a command line interface which allows to perform both non-visual and visual logical/closure merge and visual logical/closure compare (non-visual logical/closure compare is not yet supported). It also provides possibility to re-use Eclipse mergers that are associated with files.

2 Installation

The Model RealTime EGit integration can be installed as an optional component when installing Model RealTime:





Note that in order to install it you first need to have EGit version 4.8.1 installed. The version of EGit that is already included in your Eclipse installation may be different (either newer or older), and this can cause problems when installing the Model RealTime EGit integration. In this case you must first uninstall EGit from Eclipse, and then proceed with the installation. Refer to the Model RealTime installation instructions for how to do this.

The command line interface for the Model RealTime EGit integration is based on the Model RealTime command line Compare/Merge tool ***cmcmdline.jar***. For more information about this tool see the document “Compare/Merge command line tool for Model RealTime and RSAD”. In this document we will focus on logical and closure compare/merge operations only.

3 Common principles

3.1 Eclipse Team API

The support for logical and closure merge is based on the Eclipse Team API which imposes some constraints on possible scenarios. The Eclipse Team API is intended for files in an Eclipse workspace. All its functionality is available only if files are present in a workspace in some way. Thus it becomes impossible to use logical or closure merge functionality on files that are not imported into a workspace.

The Eclipse Team API does not have explicit support for closures, it supports only logical models. In Model RealTime a closure is represented as an implicitly generated hidden *.ecx file representing a root of a logical model where fragments of this logical model are *.emx files included in the closure. The closure file is generated automatically based on the options you provide when the compare/merge operation is started.

3.2 Workspace configuration

The Model RealTime command line Compare/Merge tool requires access to an Model RealTime instance with a properly configured workspace. You can either use a pre-configured workspace or use options for automatic population of a workspace with projects from a Git repository.

The Model RealTime EGit integration does not require to set any special preferences in the workspace in order to use compare or merge operations from the command line. It only needs relevant resources to be imported into the workspace and the Git repository should be in the right state when performing compare/merge operations.

There are two main recommendations for workspace content when using command line logical or closure compare/merge

- do not have projects from different Git repositories (this may cause problems with resolving branch names since different repositories may have branches with the same name)
- do not have non-Git projects in the workspace (this decreases performance of compare/merge operations because Model RealTime needs to filter out unrelated projects)

4 Merge operations

The generic behaviour of merging from commit/branch/ref '*S*' to branch '*T*' on a Git repository using the Model RealTime command line Compare/Merge tool is the following:

1. Connect to a running Model RealTime instance (or launch a new instance if specified by command line options)
2. [*Optionally*] Create workspace and import relevant projects from Git repository.
3. Check that workspace contains content related to the target branch 'T'
4. Group files in workspace into logical or closure models
5. Perform non-visual logical or closure merge for each logical/closure model
6. Perform non-visual file-by-file merge for Model RealTime files not included in logical models or closures (e.g. transformation configurations)
7. Perform default non-visual EGit text merge for non Model RealTime files (text files, C/C++ source files, etc)
8. [*if xmerge command was used*] Start Merge Tool on each conflicting logical or closure model and other Model RealTime files (e.g. transformation configurations).
9. [*optionally (if the command-line option -CPFG="commit:true" is used) and only if no conflicts were found*] Perform commit of the result to the Git repository.

4.1 Logical merge from command line

The common format for starting command line logical merge is:

```
java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool merge|xmerge -kind=logical -source "source" -target "target" [<options>]
```

Option	Description
-kind=logical	Enables logical model merge
merge	Runs non-visual logical merge. When conflicts are detected they will be printed to the console.
xmerge	Runs visual logical merge. This operation performs non-visual merge first and then starts the Merge Tool on each conflicting logical model. The merge tools are started sequentially. Information about currently processed logical model is printed to the console (see example below). When you resolve conflicts in the Merge editor and click on the 'Commit merge session' button, the affected files are automatically added to the Index and the red conflict decorator is removed in the Project Explorer
-source "source"	Defines the source context for the merge. The name within quotes should specify one of the following: • commit SHA • complete reference name (refs/...) • branch name • short reference name under refs/heads, refs/tags, refs/remotes namespace • HEAD
-target "target"	Defines the target context for the merge. The name within quotes should

	specify one of the following: • branch name • complete reference name (refs/...) • short reference name under refs/heads, refs/tags, refs/remotes namespace • HEAD The workspace of the running Model RealTime instance should contain projects from the target context.
-CPCFG=”values”	Defines additional options for the merge operation. Currently supported options are: • -CPFG=”commit:true|false”. This option enables/disables auto-commit of the merge result. Default value is ‘false’. Autocommit can be performed only if no conflicts were detected. • -CPFG=”ffmode:ff|no-ff|ff-only”. This option sets the fast-forward method for the merge. Default value is “ff” (fast-forward). • -CPFG=”squash:true|false”. This option enables or disables squash mode for the merge. Default value is ‘false’. • -CPFG=”fcheckout”. This option forces a checkout of the target branch. To use more than one of these options, separate them with comma. For example: -CPCFG=commit:true,ffmode:no-ff
-CPMSG=”value”	Defines the commit message for the merge commit

Example:

<u>Non-visual merge:</u>

Command:

java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool merge -kind=logical -source “branch-01” -target “main-01”

Output:

```
CONNECT  : Connecting to TeamServer...

CONNECT  : Connected to TeamServer at port 60001. Protocol: 9.2

Performing Logical Non-Visual Merge on contributors:

    -source=branch-01

    -target=main-01

Calling RSx EGit Command Line Logical and Closure Merger to perform a LOGICAL MODEL merge

Preparing compare/merge operation

Running compare/merge operation

Operation completed with status 'Conflicting'

Merge status message:

Merge branch 'branch-01' into main-01
```

```
Conflicts:
      LMM1/ModelFragment_1.efx
      LMM1/ModelFragment_2.efx
      LMM1/Root1.emx
      LMM2/Root2.emx
      LMM2/ModelFragment_2.efx
      LMM2/ModelFragment_1.efx



Command line merge COMPLETED
```

Visual merge

Command:

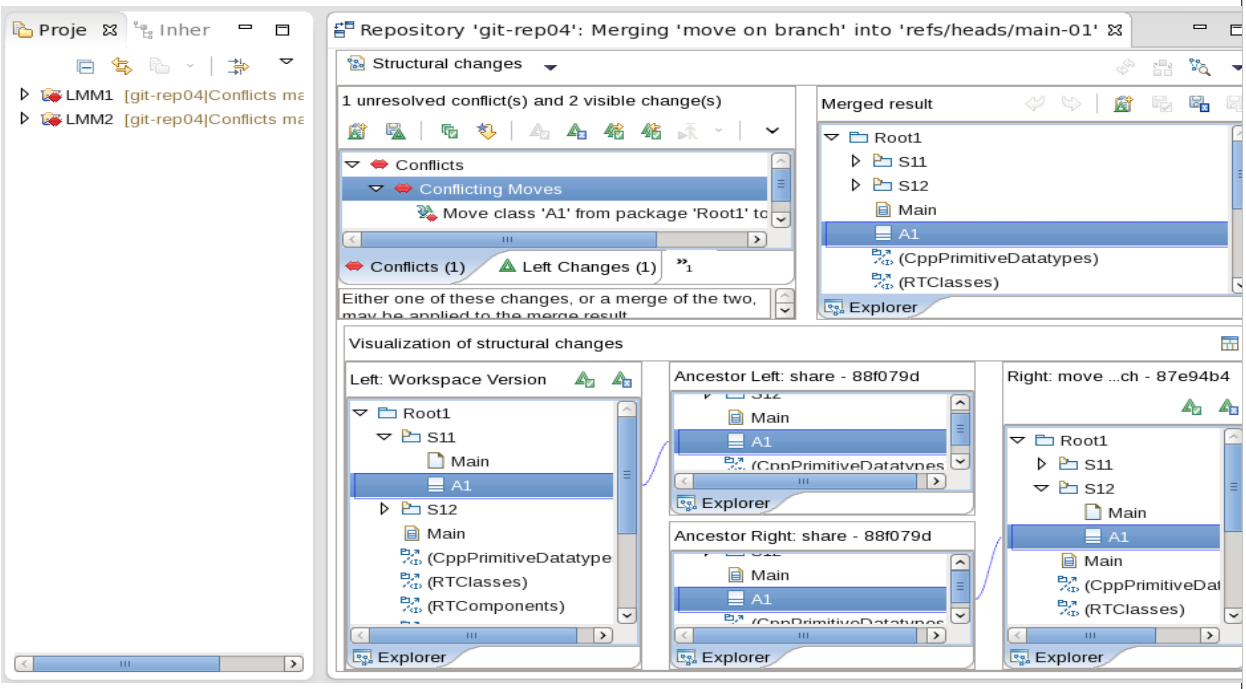
java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool xmerge -kind=logical -source “branch-01” -target “main-01”

Output:

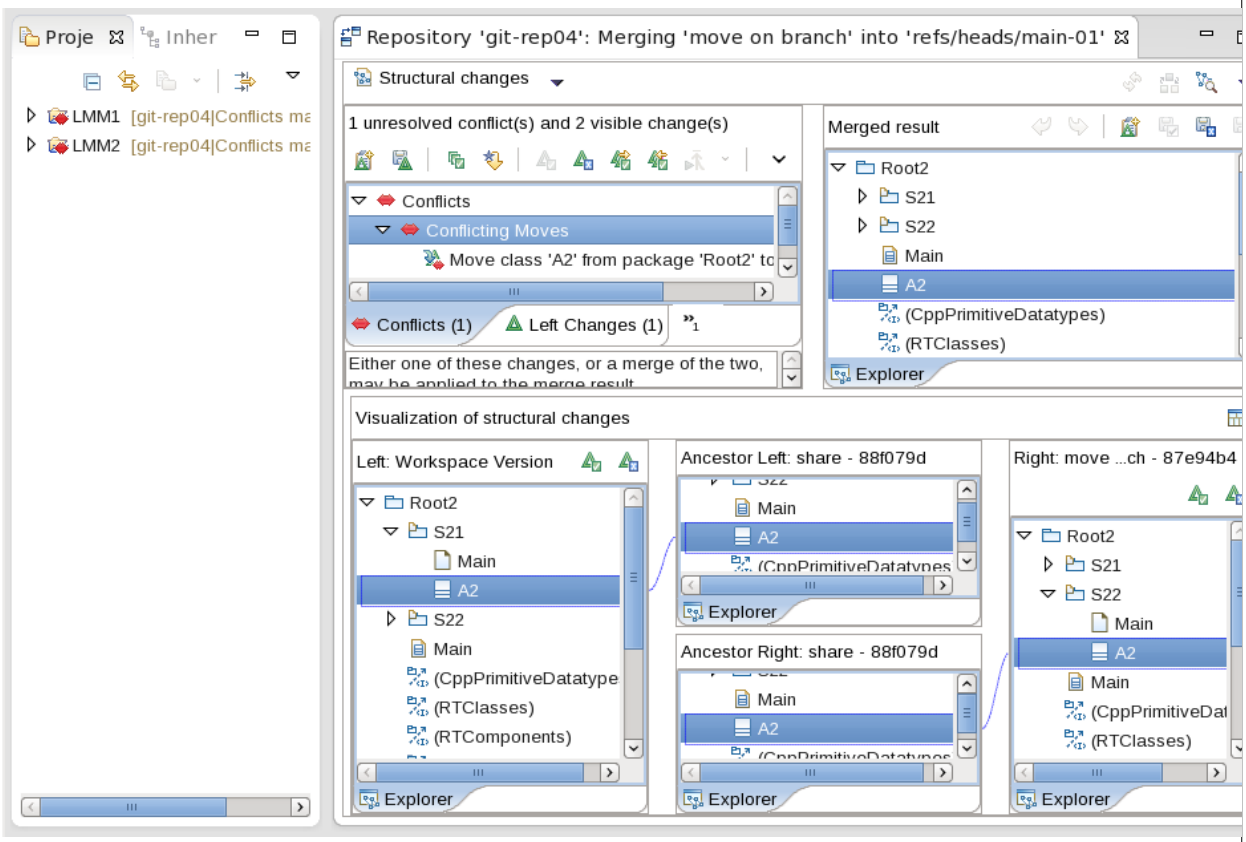
```
CONNECT  : Connecting to TeamServer...
CONNECT  : Connected to TeamServer at port 60001. Protocol: 9.2
Performing Logical Visual Merge on contributors:
    -source=branch-01
    -target=main-01
Calling RSx EGit Command Line Logical and Closure Merger to perform a LOGICAL MODEL merge
Preparing compare/merge operation
Running compare/merge operation
Operation completed with status 'Conflicting'
Merge status message:
Merge branch 'branch-01' into main-01

Conflicts:
      LMM1/ModelFragment_1.efx
      LMM1/ModelFragment_2.efx
      LMM1/Root1.emx
      LMM2/Root2.emx
      LMM2/ModelFragment_2.efx
      LMM2/ModelFragment_1.efx

Starting Merge Tool to resolve conflicts
Merging logical or closure model [1/2] [/LMM1/Root1.emx]
User cancelled merge for this logical model
Merging logical or closure model [2/2] [/LMM2/Root2.emx]
```

```
User cancelled merge for this logical model
Command line merge COMPLETED
```

First merge session (for the first logical model):



Second merge session (for the second logical model)



4.2 Closure merge

The common format for starting command line closure merge is:

```
java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool merge|xmerge -kind=closure -source “source” -target “target” -manifest “path” [<options>]
```

Option	Description
-kind=closure	Enables closure model merge
merge	Runs non-visual closure merge. When conflicts are detected they will be printed to the console.
xmerge	Runs visual closure merge. This operation performs non-visual merge first and then starts the Merge Tool on each conflicting closure model. The merge tools are started sequentially. Information about currently processed logical model is printed to the console (see example below).
-source “source”	The same as for logical merge
-target “target”	The same as for logical merge
-CPCFG=”values”	The same as for logical merge
-CPMSG=”value”	The same as for logical merge
-manifest “path”	Defines path to a closure manifest file. **If this option is omitted, then an implicit closure manifest file is generated which includes all logical models in the workspace**. The closure manifest is a text file containing one or more sections with the below structure: [closure Name] <workspace-relative path to *.emx file> <workspace-relative path to *.emx file> Where [closure Name] defines closure with given ‘Name’ containing one or more logical models with specified root files.

Example:

Non-visual merge:

Command:

java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool merge -kind=closure -source “branch-01” -target “main-01”

Output:

```
CONNECT  : Connecting to TeamServer...
CONNECT  : Connected to TeamServer at port 60001. Protocol: 9.2
Performing Closure Non-Visual Merge on contributors:
    -source=branch-01
    -target=main-01
Calling RSx EGit Command Line Logical and Closure Merger to perform a CLOSURE merge
```

```
Preparing compare/merge operation
Running compare/merge operation
Operation completed with status 'Conflicting'
Merge status message:
Merge branch 'branch-01' into main-01

Conflicts:
      LMM1/ModelFragment_1.efx
      LMM1/ModelFragment_2.efx
      LMM1/Root1.emx
      LMM2/Root2.emx
      LMM2/ModelFragment_2.efx
      LMM2/ModelFragment_1.efx

Command line merge COMPLETED
```

Visual merge

Command:

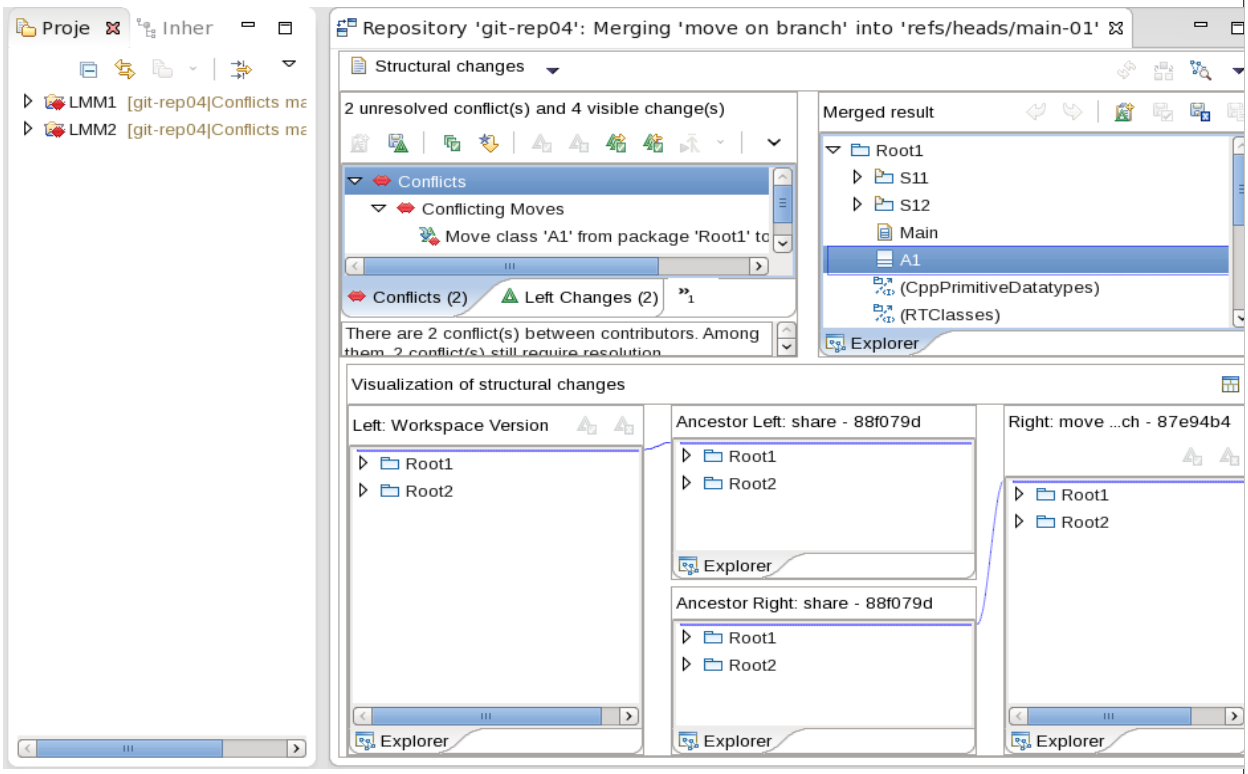
java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool xmerge -kind=closure -source “branch-01” -target “main-01”

Output:

```
CONNECT  : Connecting to TeamServer...
CONNECT  : Connected to TeamServer at port 60001. Protocol: 9.2
Performing Closure Visual Merge on contributors:
    -source=branch-01
    -target=main-01
Calling RSx EGit Command Line Logical and Closure Merger to perform a CLOSURE merge
Preparing compare/merge operation
Running compare/merge operation
Operation completed with status 'Conflicting'
Merge status message:
Merge branch 'branch-01' into main-01

Conflicts:
      LMM1/ModelFragment_1.efx
      LMM1/ModelFragment_2.efx
      LMM1/Root1.emx
      LMM2/Root2.emx
      LMM2/ModelFragment_2.efx
```

```
      LMM2/ModelFragment_1.efx

Starting Merge Tool to resolve conflicts
Merging logical or closure model  [1/1] [/LMM1/Root1.emx, /LMM2/Root2.emx]
User canceled merge for this logical model
Command line merge COMPLETED
```

Single merge session (closure includes two logical models)



5 Compare operations

Currently only visual logical or closure compare is supported. The compare operation works similar to merge - it compares the source context with the target context which is in the workspace.

The common format of launching a logical or closure compare operation is

```
java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool xcompare -kind=logical|closure -source “source” -filter=”filter” [-target “target”] [<options>]
```

Option	Description
-kind=logical|closure	Enables logical or closure compare
xcompare	Runs visual compare
-source “source”	Defines the source context for the compare operation. The name within quotes should specify one of the following: • commit SHA • complete reference name (refs/...) • branch name • short reference name under refs/heads, refs/tags, refs/remotes namespace • HEAD
-target “target”	This defines the target context for the compare operation. **If this option is omitted, then compare will be performed against the current configuration of the workspace.** Possible values are the same as for the ‘-source’ option.
-filter=”pattern,pattern,...”	Defines a pattern for file selection for the compare operation. The value is a comma-separated list of file name patterns (which may contain * and ? wildcard symbols). The patterns are matched against the workspace-relative path of *.emx files in the workspace. When using logical compare, the pattern must match just one logical model. When using closure compare, the pattern can match multiple *.emx files. All matched files will be grouped into the closure.

Example

Closure compare:

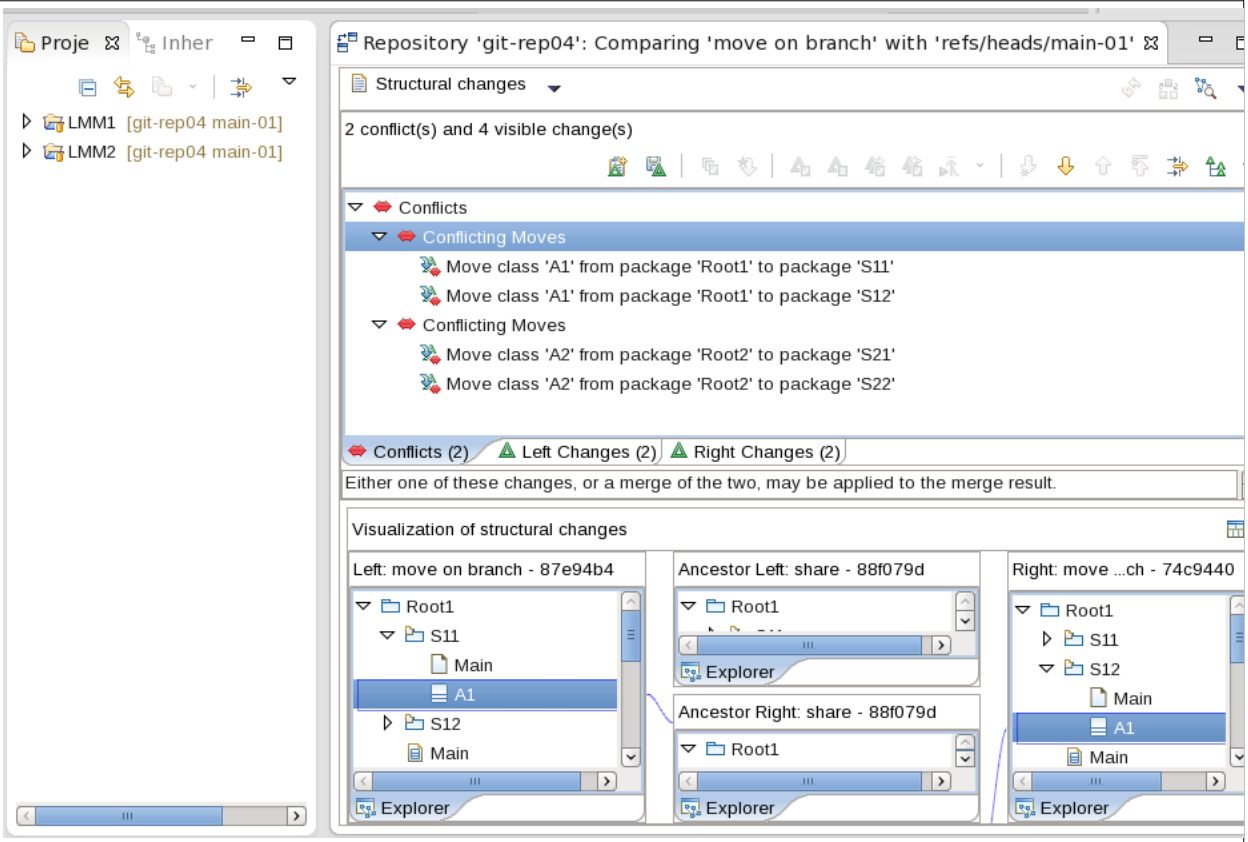
Command:

```
java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool xcompare -kind=closure -source “branch-01” -target “HEAD” -filter “*LMM*”
```

Output:

```
CONNECT  : Connecting to TeamServer...
CONNECT  : Connected to TeamServer at port 60001. Protocol: 9.2
Performing Closure Visual Compare on contributors:
```

```
    -source=branch-01

    -target=HEAD

Calling RSx EGit Command Line Logical and Closure Merger to perform a CLOSURE compare

Preparing compare/merge operation

Running compare/merge operation

Openning Compare session.

Compare Session completed.

Command line merge COMPLETED
```



Logical Compare

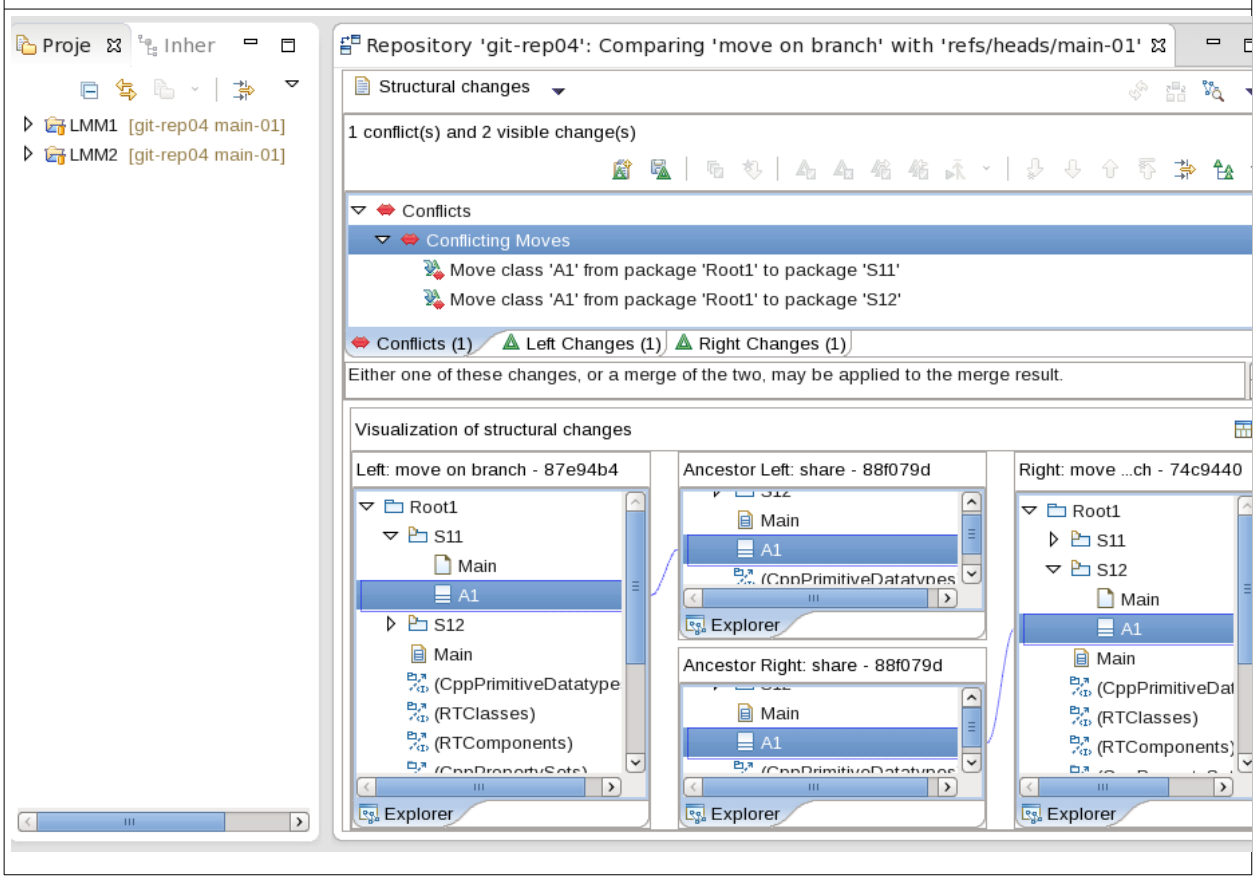
Command:

java -cp cmcmdline.jar com.ibm.xtools.comparemerge.cmcmdline.CMTool compare -kind=logical -source “branch-01” -target “HEAD” -filter=”*LMM1*”

Output:

```
CONNECT  : Connecting to TeamServer...

CONNECT  : Connected to TeamServer at port 60001. Protocol: 9.2

Performing Logical Visual Compare on contributors:

    -source=branch-01

    -target=HEAD

Calling RSx EGit Command Line Logical and Closure Merger to perform a LOGICAL MODEL compare

Preparing compare/merge operation

Running compare/merge operation
```

```
Openning Compare session.
Compare Session completed.
Command line merge COMPLETED
```



6 Automatic project import

When working with command line logical or closure compare/merge it is important to have a properly configured workspace. Model RealTime provides several solutions for automatic workspace configuration and population.

6.1 Custom script for launching

The workspace population can be done as part of a custom script for launching the Model RealTime instance to be used in the compare/merge operation.

The custom script can be passed to the Model RealTime Compare/Merge command line tool via one of the following options:

-autoLaunch|-al <argument>	Runs the specified Model RealTime instance for the Compare/Merge operation. When this option is used, the tool will use **only** the auto-launched version. The argument should be a command line to launch an Model RealTime instance.
-autoLaunchFile|-alf <file path>	The same as '-autoLaunch' but takes the launch command from the specified file.

6.2 Project import filters

You may specify additional options on the command line for compare or merge operations to ensure automatic project import from a Git repository:

Option	Description
-CPimport="path"	Defines absolute path to the Git repository in the local file system

-CPimportManifest="path"	Defines absolute path to an import manifest file. An import manifest is a text file with a list of filters describing which projects should be imported or ignored. If the manifest is omitted, then all projects will be imported.
	The import manifest consists of a list of rules:
	• Inclusion (project whose path matches condition will be imported): +wildcardpattern or +[regex]
	• Exclusion (project whose path matches condition will be ignored): - wildcardpattern or -[regex]
	where "wildcardpattern" is a string that may contain wild card symbols (* or ?) and "regex" is a Java regular expression.
	Example:
	`+*MyProject*`
	`-*ObsoleteProject*`
	`-[\\d+\\w+]`

6.3 Using extension point to define custom project importer

You may also define a custom project importer in a separate plugin using a special extension point.

```
<extension
        point="com.ibm.xtools.comparemerge.egit.projectImporter">
     <projectImporter
           class="ProjectImporterFactory">
     </projectImporter>
</extension>
```

Sample Java implementation:

```
public class ProjectImporterFactory implements IExecutableExtensionFactory {
    public static class MyImporter {
        public IProject[] importProjects(String repository, String
sourceBranch ,String targetBranch, IProgressMonitor monitor) {
            // TODO: Import projects from the 'repository'
            return null;
        }
    }

    public ProjectImporterFactory() {
    }

    @Override
    public Object create() throws CoreException {
        return new MyImporter();
    }
}
```

6.4 Commit analysis

A possible future improvement would be to have an automatic project import that will import projects based on analysis of the commits that are involved in the compare/merge operation.