02/28/03 Version 1.2

Rational. software

Using IBM Rational XDE and IBM Rational ClearCase Together

Khawar Z. Ahmed

Table of Contents

Introduction

Over time, visual design and Software Configuration Management (SCM) have become increasingly critical to the success of modern software projects. As the provider of IBM Rational XDE and IBM Rational ClearCase, market-leading tools in the visual design and software configuration management categories respectively, IBM Rational offers seamless integration between these products, thereby simplifying the software development experience.

This document, intended for software developers using Rational XDE/Java Platform Edition and Rational ClearCase, outlines the details of the integration between Rational XDE and Rational ClearCase.

The primary purpose of this document is to familiarize developers with XDE and ClearCase concepts related to configuration management and identify approaches to using Rational XDE and ClearCase together. The document also outlines the most common configuration management related tasks from a software developer perspective. Important points to note appear highlighted in the document.

This document does not cover topics related to Rational ClearCase administration nor does it cover other editions of Rational XDE. If you are interested in those topics, please see the references section at the end of this document for sources of additional info.

Supported Software Versions and Configurations

Since software products and functionality changes over time, this discussion applies only to the following specific software releases of IBM Rational products:

- Rational XDE 2002 Release 2.1 Service Release
- Rational ClearCase 2002.05.00 with patch level 15 or higher
- Rational ClearCase LT 2002.05.00 with patch level 2 or higher

Additionally, to limit the discussion, we do not cover the following related to Rational ClearCase:

- MultiSite
- Triggers

What is Software Configuration Management?

Anyone who has worked in a team project has likely encountered and practiced software configuration management in one form or another. It may have been something simple such as ensuring that modified software is checked in to a version control system for safe keeping to something more elaborate requiring numerous scripts to setup the needed environment for development.

More formally, Software Configuration Management (SCM) is commonly used to refer to:

- SCM Tools
- Techniques

involved in managing change to software. SCM tools automate the techniques involved in software configuration management.

From a developer perspective, practicing SCM requires minimal additional work, however it offers significant benefits such as:

- *Security:* SCM ensures your artifacts are secure and available on demand via a repository
- *Versioning:* You can track changes to software as it evolved and revert to previous versions if required
- *Organization:* You can organize versioned artifacts into sets, projects and releases, as required

Modern SCM tools, such as Rational ClearCase, offer numerous additional benefits. Some of these are highlighted in the next section.

IBM Rational ClearCase Overview

Rational ClearCase is the leading SCM tool on the market. It offers a flexible, proven approach to SCM automation that can be employed in software projects of all types.

Like other SCM tools, Rational ClearCase provides all essential SCM functions such as the ability to safeguard and version software artifacts. Unlike other SCM tools, Rational ClearCase also offers several advanced capabilities:

- *Concurrent changes:* When working on a project, two or more developers can make changes to software artifacts concurrently. Rational ClearCase provides graphical merge and conflict resolution capabilities for bringing the changes together.
- *Environment and Workspace Management:* ClearCase allows you to recreate entire project development environments, including complete development workspaces, on demand
- *Parallel development:* ClearCase provides extensive capabilities that allow you to create and work on multiple project releases simultaneously.
- *Distributed development:* ClearCase enables teams to develop software in a geographically distributed team environment via replicated repositories
- *Unified Change Management (UCM):* With Rational ClearCase, you can work at a higher level of abstraction by organizing your changes along tasks, defects or enhancement requests. Such activity-based development streamlines your entire change/configuration management workflow.

IBM Rational ClearCase Terminology and Concepts

Rational ClearCase capabilities can be broadly classified into two basic categories:

- General purpose SCM capabilities, commonly referred to as *Base ClearCase*
- Activity-based SCM capabilities, referred to as *Unified Change Management* or *UCM*

Base ClearCase

Base ClearCase revolves around the notion of versioning software artifacts in a permanent data repository known as a *Versioned Object Base* or *VOB*. VOBs store *elements* that can be files and directories. You can also split and join VOBs together.

A ClearCase VOB is different from a typical CM repository in that it uses the file system notion to represent the stored elements. That is, ClearCase VOBs display their content as files residing in a file system. Not only that, you can also manipulate the ClearCase VOB contents as you would manipulate something in a file system.

Software development often requires a specific working environment. For example, as a Java developer, you may require certain source files to be located in a specific directory structure. Rational ClearCase facilitates this by supporting the concept of a workspace, known as a *view*. The main idea is to facilitate the setup and creation of a sandbox where developers have a properly configured and stable environment for developing software. You define a view with the help of rules in the form of a *configuration specification (config spec)*, which selects the versions of elements you want to work on.

In base ClearCase, you work by selecting a view, checking out desired elements, modifying them as needed and checking them in as required. In other words, you are responsible for knowing and managing the details of what elements need to be checked in/out in support of a specific task, and so on.

There are two types of views in ClearCase.

A *snapshot view* provides a static view of the versioned elements in the VOB. That is, when you create your workspace, you essentially make local copies of the elements you want to work with, as they existed at the time you created the view. You can then proceed with your changes and later "synchronize" with the VOB contents via an *update* operation.

Snapshot views offer the ability to work in a disconnected fashion. Since you have local copies of artifacts, you can work independently. Furthermore, some operations may be faster since you are dealing with local artifacts rather than performing operations over the network.

Like snapshot views, *dynamic views* also allow you to create a workspace, with the key difference that no local copies of the elements are maintained. Instead, the elements are directly accessed in the VOB via a virtual file system.

One of the advantages of dynamic views is that you are not restricted to accessing static (and possibly stale) contents that you copied over at the time the snapshot view was last updated. It is also faster to create a dynamic view since you are not required to copy elements locally. Dynamic views also allow you to reuse build artifacts that have been built by others via a process known as *wink in*.

Rational ClearCase also supports the notion of a *branch*. A branch allows you to perform parallel development as well as maintain multiple versions of an element. Each element has a *main* branch (its default branch). You can create additional branches from the main branch as required. For example, if you needed to do a custom product release for a customer, the development could be handled by creating a separate ClearCase branch for that effort.

Unified Change Management

Unified Change Management (UCM) builds upon base ClearCase concepts by wrapping SCM in an out-of-the-box best practices process. This best practices process typically involves the use of Rational ClearCase and Rational ClearQuest in an integrated manner, although it is possible to work in a ClearCase-only UCM environment.

The main idea behind UCM is to simplify the overall task of SCM by focusing on components and activities thereby making a vast majority of change management functions transparent to the user.

A ClearCase *component* groups files and directories that need to be developed, integrated and released together. Similar to the idea of a component in software, a ClearCase component typically implements a reusable piece of the system. A new version of a component, created when you modify the elements that make up a component, is called a *baseline.*

A developer in a team using ClearCase UCM must first join the ClearCase project. A ClearCase *project* is created by the project manager and establishes the environment required for the software development effort e.g. components included, a workspace configuration, shared area where changes will be integrated, and so on.

When a developer joins a project, a logical work area (based on the details specified in the project) is automatically created where the developer performs the development work. This logical work area consists of a development view and a development stream. The development *stream* contains the information needed to automatically generate a config spec for the view.

Each UCM project also has an integration stream that is part of a shared work area.

As a developer, you make changes to components based on activities that have been assigned to you. An *activity* essentially identifies a task, defect or feature that needs to be worked on in the context of a specific set of file versions. This allows you to focus on tasks you need to complete without needlessly spending time on determining which files to check in or check out, or worrying about which files line up with certain versions, and so on.

When you are done with your work, you deliver activities to the project integration stream. Each project's integration stream is isolated and changes delivered to an integration stream are not visible to another project until those changes are incorporated into the next project baseline.

Periodically, you may also need to *rebase* your development stream to update your view and access any recent changes that have been delivered and incorporated into the baseline. In general, it is a good idea to rebase your development prior to a delivery operation to ensure that changes being delivered are on the latest baseline.

IBM Rational XDE Terminology and Concepts

Rational XDE provides extensive integration with Rational ClearCase. While the integration between the two products is set up such that you can use them together out of the box, it is useful to understand the Rational XDE concepts presented in this section in order to customize the integration and associated behavior to your specific SCM needs.

Cross-model references

In Rational XDE, You can create references across models and projects. Such references are called c*ross-model references* and require XDE to maintain information about where the resources are located.

Cross-model references do not dynamically adjust to moving and copying workspaces, nor are they view-aware. In other words, the cross-model information uses absolute paths and if you copy or move the resource in any way, it needs to be managed carefully. For example, another user may be working on a different drive or in a different view and so the absolute path will fail to resolve when the cross-model element is encountered.

There are several XDE concepts related to cross-model references.

First is the *Location Registry*. Rational XDE uses a location registry to maintain information about specific locations where cross-model resources may be located and paths to those locations.

Each entry in the location registry is called a *registered location.* Such registered locations map a unique location identifier called a *component ID* to a path. These may be automatically created when you create a reference between models (for example, by dragging a class from one model onto a diagram in another) or you may create them manually.

Each machine has a Location Registry containing entries for all the registered locations available on that machine. The physical registered location (such as a directory containing some model elements) is itself marked by the presence of a *.ratl_comp_root* file that contains the unique location identifier.

A registered location is an entry in the location registry and consists of a component ID and the root directory where the component is located.

Let's walk through a situation where a new cross-model reference is created.

Assume that we have two registered locations for directories d1 and d2. Since a registered location is marked by the presence of a .ratl_comp_root file, each directory contains one such file that has the component ID of the registered location that it marks. Furthermore, directories *d1* and *d2* contain models *a* and *b*, respectively and each model contains a class. These classes are named C1 and C2, respectively. The directory structure is shown graphically in figure 1.

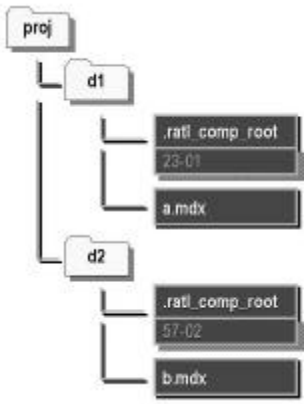

Figure 1: Registered locations

If you drag and drop class C1 from model A to model B, XDE will look for the .ratl_comp_root file in the containing directory d1. Once it finds the .ratl_comp_root file, it will create a new cross model reference using the unique ID from the .ratl_comp_root file for the component ID field and the path to the model to fill in the Model Path field.

Starting with the Rational XDE SR release, VOB root-level components are created automatically. All subsequent cross-model references will thus be relative to the root of the VOB in which the model resides.

Storage Units

When you use Rational XDE, you build a visual model of your UML-based design. Such models are stored in an .mdx file. For simple projects, it's generally sufficient to use a single model file to manage your model. In any realistic project, however, such a monolithic approach to model storage can cause issues. For example, a large model may cause model loading at startup to be slow and inefficient. For a team based development environment, a single model often leads to conflicts as multiple people make changes to a single model file.

To overcome the issues associated with the use of a single file to store the entire model, Rational XDE lets you partition a model into multiple, smaller files. Each of these files is generically referred to as a *storage unit.* In Rational XDE, a storage unit can be of varying granularity, all the way from a complete subsystem or package down to a single class or a diagram.

Rational XDE also provides a user preference for partitioning a model into storage units automatically. These settings and their pros and cons are discussed further in the section *Setting up Rational XDE for use with Rational ClearCase.*

Model Profile and upgrade

A *model profile* defines the set of data a Rational XDE model can hold. A model profile may need to be changed from one XDE release to the next to accommodate new product capability requirements.

When you upgrade from one release of Rational XDE to another, it may or may not entail model profile changes. If there are no model profile changes, the upgrade process is straightforward and no special user actions are required.

On the other hand, if the model profile has changed, all models using an older model profile must be upgraded to the new model profile before they can be used. This is necessary because you cannot compare or merge XDE models that are based on different model profiles. A side effect of model upgrade is that you may encounter a large number of conflicts when you try to merge models that have just been upgraded.

In the context of ClearCase, a specific process is recommended for upgrading models to a new profile. For full details, please see the Rational XDE Service Release documentation.

Setting Up Rational XDE for Use with Rational ClearCase

Several Rational XDE user preference settings can potentially affect its interaction with Rational ClearCase. Starting with the Rational XDE 2002 Rel 2.1 Service Release (SR), most such settings are preset appropriately by default.

The recommended settings, rationale for the recommendations and situations where it may be warranted to deviate from the recommendations are outlined below.

Checkout preferences

These checkout related user preferences are accessed via *Window>Preferences>Rational ClearCase>Advanced Options>Operations tab.* This is shown in figure 2.

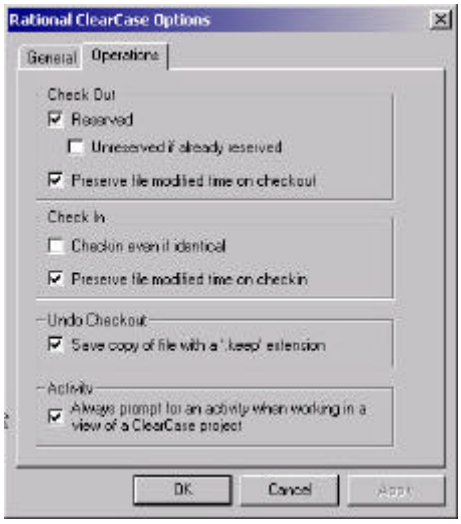

Figure 2: Checkout preferences dialog

There are three user settings on this dialog. Their purpose and recommended settings are discussed below.

- *Reserved:* Checking this box forces XDE to perform a reserved checkout when a checkout operation is carried out. A file is available for reserved checkout if another user hasn't already checked it out in reserved mode. This is the default setting because it reduces merging. If you understand how merging works, are comfortable with merge sessions and associated considerations involved, you can uncheck this option.
- *Unreserved if already reserved:* If the file is already checked out in reserved mode, a user can optionally checkout the file in unreserved mode. This will permit both users to edit copies of the file simultaneously, which can then be merged together when the users submit their respective changes. By default, this is left unchecked and it is best to leave it so. The primary reason is to reduce the overall merging, since chances of requiring merges increase with unreserved checkouts. If you are comfortable with merges and associated limitations, you can disregard this recommendation.
- *Preserve file modified time on checkout:* When you compare the time the file was last modified in the view to the modified time on checkout, it sometimes can trigger unnecessary reloads. To minimize such reloads, this setting is enabled by default at the start of each session. You can however uncheck it once the session is started. It is strongly recommended that the setting be left checked.

Checkin preferences

These checkin related user preferences are accessed via *Window>Preferences>Rational ClearCase>Advanced Options>Operations tab.* This is shown in figure 2.

There are two options related to checkin:

- *Checkin even if identical:* By default, when a new file is added, it is placed under source control and kept checked out. If no changes are made, subsequent checkin attempts will result in error for trying to checkin identical items. In general, users who pay attention to details and understand why an error is being reported need not set this option. If you don't want to get this error message and don't care about having an identical version of file being created, you can check this option
- *Preserve file modified time on checkin:* This is similar to the *preserve file modified time on checkout* setting discussed in the previous section. Checking this setting can sometimes trigger unnecessary reloads. To minimize such reloads, this setting is enabled by default at the start of each XDE session, however you can uncheck it after the session has started. It is strongly recommended that the setting be left checked.

Undo Checkout preferences

There is only one setting related to undo checkout. It is accessed via *Window>Preferences>Rational ClearCase>Advanced Options>Operations tab.* This is shown in figure 2.

- *Save copy of file with a '.keep' extension:* This is really a common sense setting. It is generally good practice to keep a copy of the local file on undoing a checkout command in case you need to perform recovery of some sort. If you are comfortable with the risks associated with not doing so, then you can safely uncheck it. This is checked by default

Activity preferences

There is only one setting related to activity preferences. Note that this only applies to those projects using ClearCase UCM. It is accessed via *Window>Preferences>Rational ClearCase>Advanced Options>Operations tab.* This is shown in figure 2.

- *Always prompt for an activity when working in a view of a ClearCase project:* You should check this option to ensure you are prompted regularly to set the correct activity. This is generally considered to be a good practice. But if you are using a single activity for an extended period of time, you can leave this option unchecked.

Model-Code synchronization Preferences

The Rational XDE model-code synchronization related XDE preferences are accessed via *Window>Preferences>Rational XDE>Code-Model Synchronization.* The preference of interest is *Autosync*. This is shown in figure 3.

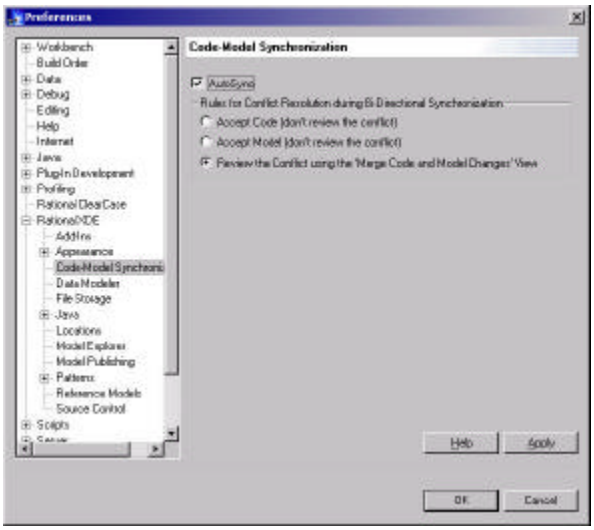

Figure 3: Model-Code synchronization Preferences

From a developer perspective, auto-synchronization between model and code changes is very appealing because it eliminates the potential for human error associated with keeping the model and code in sync manually (e.g. overlooking to synchronize changes made in code thereby over writing the code on next model to code synchronization).

It is recommended that you turn off autosync during periods of model churn. Specifically, during periods when you are actively adding new packages and other model elements, turn off autosync and turn it back on when the model has stabilized. This will avoid situations where you add a model element to source control only to realize that it needs to be renamed, etc. Such churn is problematic from a CM perspective, as artifacts under CM control require more effort and steps to rename.

File Storage preferences

The Rational XDE storage unit related XDE preferences are accessed via *Window>Preferences>Rational XDE>File Storage.* This setting determines how the model gets stored. Details of this concept were discussed in an earlier section titled *Storage Units.* This is shown in figure 4.

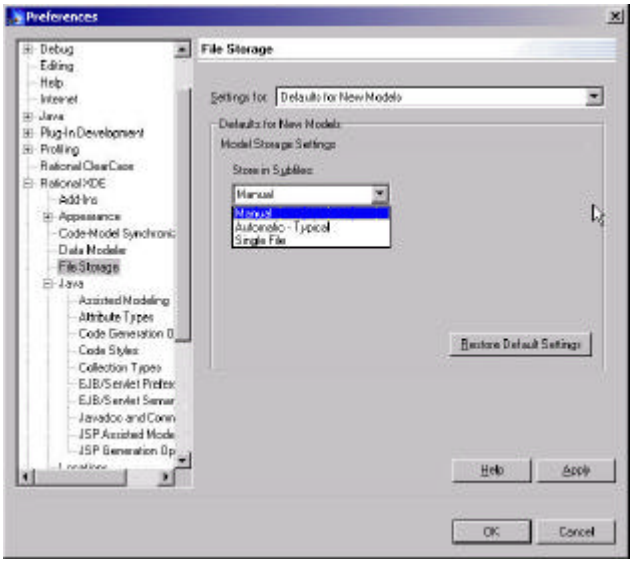

Figure 4: File Storage preferences

The user preference of interest here is the *Default for New Models – Model Storage Settings* option. This preference offers a drop-down list with the following options:

- *Manual:* With this option, the model is must be partitioned manually.
- *Automatic – Typical:* If this option is selected, packages and subsystems are automatically created as independent storage units. If you want to make something other than a package or subsystem into a storage unit, you must do so manually
- *Single File:* No subunits are created. Everything is placed into a single model file

It is up to individual teams to decide which of the three options are most appropriate for their project. As a general recommendation, you should create storage units after some thinking about the justification for creating a new unit. For example, creating them to facilitate ownership of model unit files and to reduce merging.

Generally speaking, since the architecture can change frequently and drastically during the early stages of analysis and design, modeling elements need to be created and/or moved often. This suggests the need to limit the number and granularity of model elements that should be made into storage units at this stage. Usually making only one or two levels of the package hierarchy will provide enough flexibility until the architecture stabilizes. Once areas of the architecture stabilize, then the elements in those areas can be made separate storage units down to a finer granularity before proceeding with detailed implementation.

Source Control Preferences

Rational XDE Source Control preferences are accessed via *Window>Preferences>Rational XDE>Source Control.* This is shown in figure 5.

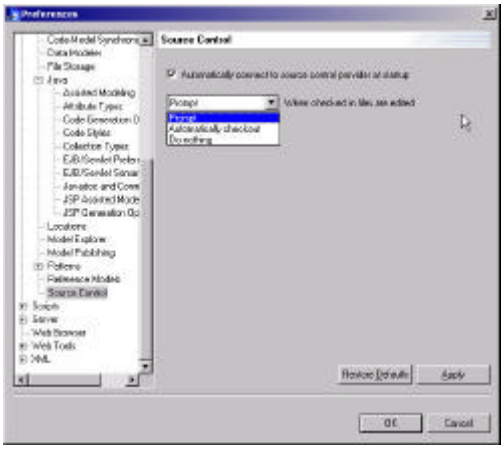

Figure 5: Source control preferences

There are two preference settings of interest:

- *Automatically connect to source control provider at startup:* If you leave this option unchecked and make changes that require any additions to source control, the changes will not be added to source control because you will not be connected to ClearCase. It is recommended that this option always be checked.
- *[action] when checked in files are edited:* This option allows you to specify the action that should be taken when a user attempts to edit a checked in file. Possible settings for this action are *prompt, automatically checkout* and *Do nothing.* It is recommended that you use either the *prompt* or the *automatically checkout* option. The *do nothing* option is strongly discouraged since using this option will permit editing a file without actually checking it out. As discussed, such in-memory editing can lead to loss of data on a subsequent view update or checkout operation.

Using Rational XDE with Base ClearCase

Setting up and using Rational XDE in a base ClearCase environment is straightforward. This section outlines the high level process for the various activities you, as a developer, will need to undertake to start using Rational XDE and base ClearCase for configuration management.

Administrative activities

Several set up steps are required before actual developer activities can take place:

- *Define VOBs*: Any VOBs that may be required for the project should be defined at this time
- *Create administrative view:* An administrative view is required so you can proceed to setup the required projects for use by the developer
- *Define projects and models:* Launch XDE and choose the administrative view. Any required projects and models should be defined at this time as a part of this administrative activity
- *Prepare for concurrent development:* Partitioning the model into subunits facilitates concurrent development.
- *Export project set file:* Once the models are set up, the project set file should be exported so the projects can be recreated in developer views.

The detailed steps for setting up VOBs, XDE models and project file sets are not discussed here. Please see the *Rational ClearCase User Manual* and the *Rational XDE Service Release Documentation* for more details on this topic.

Setting up your development environment

In order to use Rational XDE with base ClearCase, you will need to first

- *Create a snapshot view:* You can do this via the ClearCase View Creation Wizard (Start>Program>Rational ClearCase>Create View). Follow the instructions provided in Rational XDE SR release documentation for choosing the appropriate options and selecting the appropriate VOBs for the view.

- *XDE preferences:* By default, XDE sets up the appropriate preferences as outlined in *Setting up Rational XDE for use with Rational ClearCase* section. If you want to customize any settings, you should do so at this point.
- *Import project set file:* Start XDE and choose the snapshot view you just created from the list in the dialog box shown in figure 6 below. You will need to recreate the right projects and models in your view. This is done via the *ClearCase>Add Project Set to Workspace* menu option. You will need to use the project set file created by the administrator for this purpose.

Figure 6: View selection dialog

Creation of a new model element

Once you have set up the development environment, you can proceed with adding new model elements as you would normally.

- *Autosync preference off:* It is recommended that you turn autosync off during periods of extended model churn. This will help you avoid situations where you need to perform additional steps to rename an artifact that has been placed under configuration management.
- *Define a new model element:* When you define a new model element, XDE will either prompt you or automatically check out the parent of the model element from source control. This depends on your user preference settings as discussed in the *Setting up Rational XDE for use with Rational ClearCase* section.
- *Generate code:* If you do not have autosync on, you can proceed to code generation of the associated model element at anytime. When you do so, a source file will be created for the model element. For example, generating code for a class named *myclass* will result in a *myclass.java* file to be created and you will either be prompted to add the file to source control or it will be done automatically (again, depending on your user preferences). You can then proceed to add the operations and attributes to the newly created model elements as needed.
- *Autosync preference on:* If you have autosync on, XDE will generate the code for the newly created model element once you have renamed it. You will either be prompted to add the file to source control or it will be done automatically (again, depending on your user preferences). You can then proceed to add the operations and attributes to the newly created model elements as needed

Checkout of an existing model element

Working with elements already under source control is quite straightforward. You simply need to checkout the model elements and proceed to modify them as required.

- *Checkout the model element:* When you start working on a model element, XDE will either prompt you or automatically check out the model element from source control. This depends on your user preference settings as discussed in the *Setting up Rational XDE for use with Rational ClearCase* section. You can then proceed to modify the details (e.g. operations and attributes) of the checked out model elements as needed

Submitting work and resolving conflicts

Once you have completed the additions/modifications you need to make, the checked out model elements can be checked into ClearCase and a new version created.

- *Save work:* You should save all work to ensure that all your changes will be part of the submission.
- *Validate model:* This is done via *Modeling>Validate.*
- *Checkin modified artifacts:* All checked out model elements should be checked in together. This is important because checking in just some of the related artifacts while neglecting others can cause consistency issues. Rational XDE SR release automates some aspects of this.

Using XDE with UCM

This section outlines the high level process for the various activities you, as a developer, will need to undertake to start using Rational XDE and UCM ClearCase.

Administrative activities

Several set up steps are required before actual developer activities can take place:

- *Define Project VOB*: Each UCM project must have a project VOB. It needs to be defined before you can create UCM projects.
- *Define UCM Project:* To use UCM capabilities, you must create a UCM project.
- *Plan UCM components:* In UCM, components are the organizational units for your files and directories and typically represent a reusable piece of your system.
- *Define VOBs:* UCM components reside in a PVOB whereas files and directories that make up a component are stored in a VOB. Those should be created at this stage.
- *Create UCM work areas:* A user work area consists of a stream and a view. You should set up an integration stream that is part of a shared work area and development streams as required.
- *Setup project-specific details:* Once the basic UCM project setup is complete, you need to setup details such as which components are modifiable in your project, the starting point for your project in the form of a baseline (i.e. versions of each element that will be part of your project), recommending a baseline for your project and setting up additional policies as required.
- *Define projects and models:* Launch XDE and choose the administrative view. Any required projects and models should be defined at this time as a part of this administrative activity
- *Prepare for concurrent development:* Partitioning the model into subunits facilitates concurrent development.
- *Export project set file:* Once the models are set up, the project set file should be exported so the projects can be recreated in developer views.
- *Deliver to the integration stream:* To make your work visible to others, you will need to deliver to the integration stream. As part of this delivery, you should test the work in the integration stream and then complete the delivery. A new baseline should be created and recommended at this point.

Setting up your development environment

As developer about to start working on a UCM project, you need to take some steps to ensure your environment is set up appropriately before you can proceed with development activities.

- *Join a UCM project:* This is easily done by using the ClearCase Project Explorer, selecting the appropriate project and using the *Join Project* option. When you join a project, a view is automatically created for you.
- *Add project set file to workspace:* Start Rational XDE and choose the view created for you in the last step. Add the project set file to the workspace so you can access the proper projects required for your work.

Creation of a new model element

Once you have set up the development environment, you can proceed with adding new model elements as you would normally.

- *Autosync preference off:* During period of high model churn, it is recommended that you turn off autosync. This will avoid the need for additional manual steps required to rename an artifact that is under source control.
- *Define a new model element:* When you define a new model element, XDE will either prompt you or automatically check out the parent of the model element from source control. This depends on your user

preference settings as discussed in the *Setting up Rational XDE for use with Rational ClearCase* section. At this point, you can rename the model element as desired. You will also be prompted for an activity.

- *Generate code:* Once the item has been renamed, you can proceed by saving your work and doing code generation of the associated model element. When you do so, a source file will be created for the model element. For example, generating code for a class named *myclass* will result in a *myclass.java* file to be created and you will either be prompted to add the file to source control or it will be done automatically (again, depending on your user preferences).
- *Autosync preference on: Autosync preference on:* If you have autosync on, XDE will generate the code for the newly created model element once you have renamed it. You will either be prompted to add the file to source control or it will be done automatically (again, depending on your user preferences). You can then proceed to add the required details e.g. operations and attributes to the newly created model elements as needed

Checkout of an existing model element

Working with elements already under source control is quite straightforward. You simply need to checkout the model elements and proceed to modify them as required.

- *Checkout the model element:* When you start working on a model element, XDE will either prompt you or automatically check out the model element from source control. This depends on your user preference settings as discussed in the *Setting up Rational XDE for use with Rational ClearCase* section. You will also be prompted for an activity that should be associated with this work. You can then proceed to modify the details (e.g. operations and attributes) of the checked out model elements as needed

Delivering your work

In UCM, you need to deliver your work to the integration stream in order to make your work visible to other developers on the project. This is done via the *Deliver to stream* option.

Rebase

To pick up changes that others have recently delivered, a developer needs to perform a rebase operation.

Other Things to Know

There are numerous aspects of software configuration management that yield best results with a well thought out approach. This section provides and overview of some such aspects.

Model Ownership Strategies

Merging is a time consuming task and it is not always possible for a CM system to detect conflicting changes. Using specific approaches to model ownership can constrain merges required.

Generally speaking, model ownership strategies fall into the following areas:

- *Model ownership*: Simplest way to avoid merging. Since only one person owns the entire model, there is no potential for conflicts. However, this is usually not practical in a team development environment.
- *Package ownership:* Modifying multiple packages at the top level of a model leads to a root model contention situation. This is because modifying a package 'dirties' the root model and it must be checked out and updated along with the package. This approach aims to avoid the root model contention situation by setting up packages as individual units at the top level of the model and assigning individual ownership of each package. Since changes within a package's subunits generally don't affect other units, this effectively creates a sandbox work environment for each individual.
- *Unit ownership:* Since a storage unit in XDE can be finer grained than a package, the model can be partitioned into multiple storage units and each developer assigned ownership of specific units. If it turns out that many people need to access a unit, then it can be broken into further subunits to reduce contention.

One way to approach the model ownership is to consider the number and nature of people involved in specific activities. For example, very few people may be involved in analysis whereas a much larger team may be responsible for constructing the application. In such a situation, the ownership strategy may evolve from a model-based ownership during analysis to a package ownership for design and to unit ownership for construction.

Refactoring

Martin Fowler provides the following definition of refactoring: “The process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure”. It is a common activity from a developer perspective and generally revolves around either altering the design or the artifacts in such as way that the external behavior is not impacted.

Some common examples of refactoring are:

- Moving a field or method to decouple two classes
- Extracting a class out of an existing class to simplify responsibilities
- Moving or renaming a class
- Moving or renaming a package or directory

There are certain refactorings that are problematic in the context of visual development and SCM. For example, the following can cause issues while refactoring:

- Rename, delete, move of a file e.g. renaming Class1.java to Customer.java
- Rename, delete, move of a directory e.g. as a result of a package rename operation
- Moving a storage unit e.g. combining a unit with its parent

This is so because elements that are already versioned in ClearCase cannot simply be renamed by renaming them in XDE. It is recommended that you minimize refactorings if possible and follow some basic recommendations when you have to do refactorings. Please see the Rational XDE SR documentation for more guidance on refactoring with XDE and ClearCase.

Unresolved references

When a cross model reference cannot be resolved, such unresolved references are identified in the model via special icons. For example, this can happen when models that contain cross-model references are shared.

Some of the special icons for unresolved references are shown in figure 7 below. In this figure, C1is identified as an external reference via the arrow at the top left corner. The “X” within the circle in the top right corner indicates that C1 cannot be resolved. The “\” within the circle on the inheritance relationship identifies that the supplier of the relationship (in this case, C1) cannot be resolved. Had C2 been an external unresolved reference, the relationship would have been shown with an “X” to identify that neither side of the relationship is resolvable.

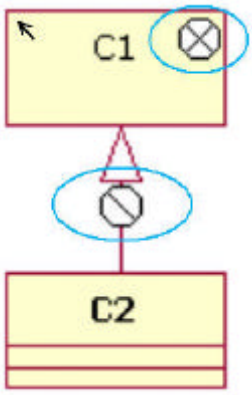

Figure 7: Unresolved references icons

Rational XDE provides built-in functionality for resolving unresolved references. To do so, you need to do the following:

- Select the Model
- Click Modeling > Check External References

- Click Resolve… in the resulting dialog box
- In the resulting dialog box, navigate to the target model. This will locate the registered location on this machine and update the location registry.

From now on, further updates to the model will resolve correctly.

Merging and conflict resolution

Rational XDE supports merging and conflict resolution to truly enable team development.

Merging requires that appropriate ClearCase Type Manager be configured to perform the merging. Merging of Rational XDE related artifacts requires the use of XDE Type Manager for proper merging. All ClearCase VOBs destined to hold Rational XDE artifacts must be configured to use the XDE Type Manager.

Beginning with the Rational XDE Service Release, the Type Manager configuration check is automated and tied to Rational XDE startup. When XDE is started, it checks and reports if the VOBs are not properly configured. You need to take corrective action to fix any issues that may be reported. Note that Rational XDE does not detects VOB Type Manager configuration issues for VOBs located on UNIX machines. This must be done manually. *See Rational XDE Service Release Documentation for more details.*

When different parties make changes to the same artifact and the changes are submitted, Rational XDE launches a ClearCase merge session to graphically show the differences between the submissions. If the differences are trivial, they are resolved automatically. For more involved differences, the user is provided the opportunity to resolve and merge the changes as desired.

A compare/merge session screenshot is shown in figure 8 below.

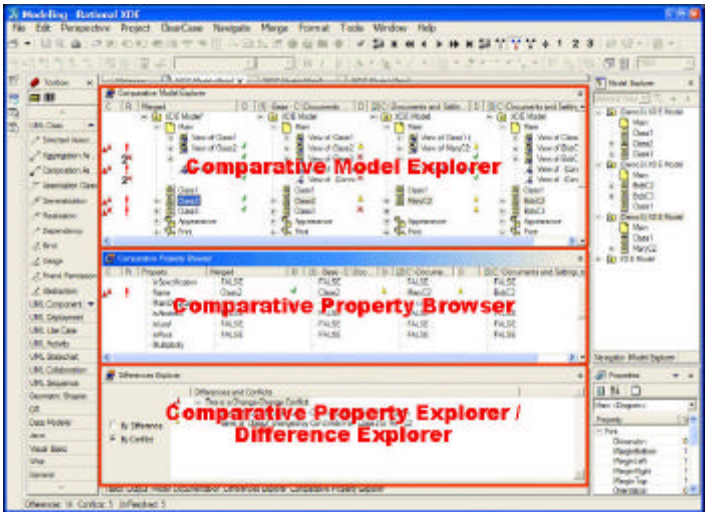

Figure 8: Compare/merge session

For more information on merging and conflict resolution, see the *Rational XDE Service Release Documentation*.

Summary

Rational XDE and Rational ClearCase provide an unparalleled integration that allows you to use these best in class tools together on a project.

You can use Rational XDE with either base ClearCase or UCM ClearCase. Specific settings are recommended for each combination to ensure an optimal work environment.

References

- White, Brian, *Software Configuration Management Strategies And Rational ClearCase*, Addison-Wesley, 2000.
- Plante, Frederic and Scott Darlington, *Guidelines for using Rational XDE and UCM*, IBM Rational white paper, 2002.
- *Guidelines for using Rational XDE and base ClearCase with IBM WSS AD*, Rational XDE Service Release Documentation, 2003
- *Guidelines for using Rational XDE and UCM with IBM WSS AD*, Rational XDE Service Release Documentation, 2003

About the author: *Khawar Ahmed is in the IBM Rational XDE Technical Marketing Team, focused on the Java Platform. He is the author of "Developing Enterprise Java Applications with the J2EE and UML", Addison-Wesley, 2001.*

IBM software integrated solutions
IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- *DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.*
- *Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.*
- *Tivoli® software helps you manage the technology that runs your e-business infrastructure.*
- *WebSphere® software helps you extend your existing business-critical processes to the Web.*
- *Rational® software helps you improve your software development capability with tools, services, and best practices.*

Rational software from IBM
Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Printed in the United States of America
01-03 All Rights Reserved.
Made in the U.S.A.

The IBM home page on the Internet can be found at **ibm.com**