Automated Model Integration and Release Mechanism for Multi-core Microprocessors

Major Project Report

Part - II

By

Sandip U Rajput

09MEC016

Department OF Electronincs and Communication Engineering

Ahmedabad-382481

May - 2011

Automated Model Integration and Release Mechanism for Multi-core Microprocessors

Major Project

Part -II

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Electronics and Communication

(VLSI Design)

By

Sandip U Rajput

09MEC016

External Guide

Mr. Narasimhan J Iyengar

Component Design Engineer,

Intel Technology India Pvt. Ltd.

Bangalore

Internal Guide

Dr. N. M. Devashrayee

PG-Coordinator(VLSI Design),

Elect. and Comm. Engineering,

Institute of Technology,

Nirma University, Ahmedabad

Department OF Electronincs and Communication Engineering

Ahmedabad-382481

May - 2011

Declaration

This is to certify that

i) This thesis details my work at Intel towards the degree of Master of Technology in Electronics and Communication (VLSI Design) at Nirma University and has not been submitted elsewhere for a degree.

ii) This work is not done by any other student of Nirma University

iii) The work described in Section 1.3 of this thesis is my original work and is based upon earlier work done at Intel

iv) Due acknowledgement has been made in the text to all other material used.

v) Opinions and/or material in this thesis are that of the student and in no way reflects the views/opinions/advice of Intel.

Sandip U Rajput

Certificate

This is to certify that the Major Project entitled **"Automated Model Integration and Release Mechanism for Multi-core Microprocessors"** submitted by **Sandip U Rajput (09MEC016)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Electronincs and Communication Engineering (VLSI Design) of Nirma University, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree.

Mr. Narasimhan J Iyengar
External Guide
Component Design Engineer,
Intel Technology India Pvt. Ltd.
Bangalore

Dr. N. M. Devashrayee
Internal Guide
PG-Coordinator(VLSI Design),
Elect. and Comm. Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Prof. A. S. Ranade
Head of Department,
Dept. of Electrical Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. K Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Abstract

The design development, processing and validation in today's Model Building (compiling, verifying) methodology is very manual, complex and time consuming. A large number of designers work on a single multi-core microprocessor development project which contains huge number of design files. Integrating all the changes made to the design files by the designers to a single design model, manually is very hectic and error prone job. The Automated model integration and Release Mechanism system when integrated with the legacy model build methodology can solve a lot of problems and can speed up the design development and validation.

Introducing new automated model integration methodology in any project is not that straight forward. This process had number of bottle-necks and difficulties. The inter design files dependency, SCM file locking problem, SCM server overloading (due to large number of tags and workspaces users and automated systems create) and huge model integration, compilation and elaboration time, were a few of them which were found to be very critical and important while testing AMI in earlier projects. These problems also needed to be solved before we actually implement AMI on the new coming multi-core project designs. One of the best solutions which were found, implemented and tested were makefile system for fast compilation, Dependency resolver system to solve the dependency problem, Automated Branching and Merging system for SCM which can solve SCM file locking and help prevent dependency conflicts, and SCM tag/client archival system which can reduce the load on SCM server for SCM and speed enhancement.

Acknowledgements

I express my gratitude and appreciation for all those with whom I worked and interacted at Institute of Technology, Nirma University, Ahmedabad and INTEL Technology (INDIA) Pvt Ltd, Bangalore thanks all of them for their help and co-operation. It has been great pleasure for me in doing a Major Project work "Automated Model Integration and Release Mechanism for Multi-core Microprocessors " under guidance of Narasimhan J Iyengar. I am very grateful to him who assigned me a project under his expert guidance, without his invaluable guidance the work would have not been possible. He always inspire to put best efforts to achieve the goal.I also personally thank Sushant Madan for his great support and technical help.

I am especially thankful to Manish Singh and Sambit Sahu from Intel Corporation for their guidance, encouragement, support and confidence in through the internship as a part of curriculum. I am especially grateful to them for giving me the opportunity to work on such an exciting project.

I greatly appreciate the generosity of Dr. N. M. Devashrayee, PG Coordinator (VLSI Design), Institute of Technology, Nirma University for his valuable and inspiring guidance, patience and support at every moment, and for giving me permission and providing the facilities for completing this project.

I would also like to thank my colleagues and friends for the things that they have taught me. My greatest thanks are to all who wished me success especially my parents.

- Sandip U Rajput

09MEC016

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Intel co-founder Gordon Moore is a visionary. His prediction, popularly known as Moore's Law, states that the number of transistors on a chip will double about every two years. Intel has kept that pace for over 40 years, providing more functions on a chip at significantly lower cost per function. The technology has evolved and the designing and manufacturing of these processors have become very much complex and complicated today. This kind of a complex task cannot be handled by any single or a group of people manually. We need a lot of amount of automation in each and every step of the processor designing.

The main goal of the automation work is to "Make Easy Things Easier and Difficult Things Possible" and to reduce the efforts needed to do the repetitive work so that the designer can apply more time and more efforts in design development. The main aim of this thesis is enhancing the Model Building (RTL development and Validation) techniques and Methodologies. For any multi-core microprocessors the biggest challenges are integrating, maintaining and validating the complex design files. Each project contains a huge number of design files which need to be stored in a secured and centralized place. Designers take the files from the main server also called mainline, edit the design files and submit the files back to the server. When embarking on a change, a developer takes a copy of the current code base on which to work. As other developers submit changed code to the code repository, this copy gradually

ceases to reflect the repository code. When developers submit code to the repository they must first update their code to reflect the changes in the repository since they took their copy. The more changes the repository contains, the more work developers must do before submitting their own changes. Eventually, the repository may become so different from the developers' baselines that they enter what is sometimes called "integration hell", where the time it takes to integrate exceeds the time it took to make their original changes. In a worst-case scenario, developers may have to discard their changes and completely redo the work.

The total timeline for any project to be completed is always very short so to complete this in time more than one designer need to work on one Design code and these designers may be on different site or different place. The basic concept of this thesis is to make a tool which can provide simultaneous or parallel access of the designs to the designers making the editing easier and faster and also automating the integration process of the changed codes and also provide a user friendly release mechanism. This is to be done initially for the unit level partition of the project and then implementing it on different partition hierarchy levels.

1.1 Continuous Integration system

Continuous integration involves integrating early and often, so as to avoid the pitfalls of "integration hell". The practice aims to reduce timely rework and thus reduce cost and time. Continuous integration - as the practice of frequently integrating one's new or changed code with the existing code repository - should occur frequently enough so that no intervening window remains between commit and build, and such that no errors can arise without developers noticing them and correcting them immediately. Normal practice is to trigger these builds by every commit to a repository, rather than a nightly (or weekly!) scheduled build. The practicalities of doing this in a multi-developer environment of rapid commits are such that its usual to trigger a short time after each commit, then to start a build when either this timer expires, or

after a rather longer interval since the last build.

1.2 Automated Model Integration System

AMI is continuous integration tool developed at Intel. This new methodology called AMI is an automated system for integrating code changes and releasing models. AMI solves the problems in traditional model builds methodology and improves productivity.

- The AMI stops bad codes reaching from the mainline and filters the model so even some designer has submitted bad codes the AMI rejects them and gives the error to designer for debugging.

- AMI is a continuous integration system so the designer doesn't need to wait for the integrator to pass his code. He can do the submission any time.

- AMI completely atomizes the work done by the integrator so enabling faster integrating and debugging capability.

- AMI quickly identifies bad submissions in about an hour or average and rejects these submissions back to the designers. By improving the response time on bad submissions, AMI improves the productivity of the designers for debugging and fixing bad submissions.

- Also by rejecting bad submissions, AMI guarantees that it can successfully release a model every night. As a result, AMI can prevent the complete project from incurring a get healthy effort loss.

- In addition AMI reduces headcount by eliminating cluster model builders. Lastly AMI allows designers to make changes to Design code at a greater rate. With the new realities of smaller design reams and shorter project schedules than in previous projects, enabling higher rates of code changes is critical.

1.3 Objectives of Thesis

The main objectives of the thesis are:

1) To assist in enabling AMI on the current project by developing modules that check the effectiveness of AMI in the current project

2) To resolve some of the bottlenecks of AMI by assisting in DRH modules

3) To test the AMI system and help users debug rejections from AMI

4) To come up with automation to enable seamless SCM branching and merging

5) To increase the performance of user SCM by archiving and cleaning-up tags and workspaces

1.4 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2, *Model Build Methodology*, describes Legacy model build methodology and its flow. It also explains the software configuration management system and complex project design hierarchy. It explains the basic differences between the Centralized SCM and Distributed SCM.

Chapter 3, *AMI (Automated Model Integration) System* , describes the basic need and application of AMI in the legacy model build methodology. It explains the basic difference between the model build methodology and AMI enabled model building. Also explains the basic bottlenecks in using AMI for the current project.

In **chapter 4**, *DRH (Dependency Resolver Helper)*, describes how we can use the DRH to solve the dependency problem AMI is facing. It also explains the DRH operation.

In **chapter 5**, *Branching and Merging in SCM*, presents the chapter Branching and merging in SCM describes the concept of branching and merging and how it is done in SCM. It also describes what integrating files means in SCM and what are the basic rules for that. It mentions what is the branching and merging behind the scenes and the vary advantages of it. Now it explains the manual steps for branching and merging in SCM and the automated system which can do this with just only one command.

In **chapter 6**, *Archival of the SCM tags and workspaces*, describes what basically archival means and why it is needed. It also describes the archival system which uses a sql database. It also mentions the archival implementation results in SCM mainline server.

Finally, in **chapter 7**, *Results, Achievements and Conclusion* mentions some of the results got while implementation, some of the achievements and final overall conclusion of the thesis.

Chapter 2

Model Build Methodology

The legacy model builds and release mechanisms use the basic checkout and check in process of SCM. In this the designer checks out the code from the mainline, makes some modifications in the code and then submits his changes to the main line. There is an Integrator at different levels of hierarchy in the project who takes the changes and integrates and sanitizes them. The integrator uses various model building scripts to integrate, analyze and test the model.

2.1 SCM (Software Configuration Management System)

SCM is a methodology to control and manage a software development project. SCM consists of several components including version/revision management, SCM tools, change management, release management, configuration management and the Software Configuration Management Plan (SCMP). Software configuration management (SCM) is the task of tracking and controlling changes in the software. Configuration management includes the revision controlling and the establishment of baselines.
Revision control, also known as version control or source control is the management of changes to documents, programs, and other information stored as computer files. It

is most commonly used in software development, where a team of people may change the same files. Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored and updated.

2.2 Complex Model Hierarchy

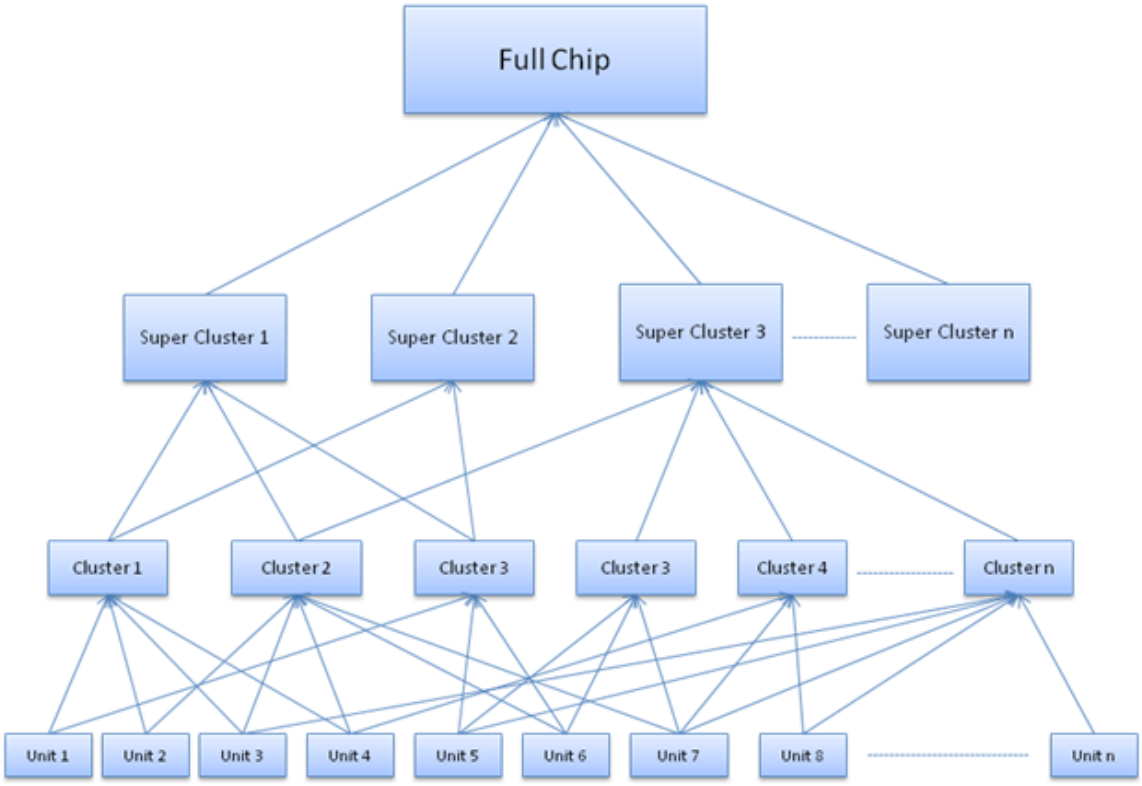

Figure 2.1: Complex Model Hierarchy

In the above block diagram we can see the complete hierarchy of any microprocessor project. It contains around 5 hierarchy levels each lower level feeding to the

upper level with very complex dependencies between them. Each unit has around 50 design files. The design files may have instantiations of other design files which may or may not be in the same unit. This creates inter unit dependency. The units are fed to the clusters. There are units which are fed into more than one cluster which creates the inter cluster dependencies. Same way the clusters fed to the super cluster and all the super clusters combine to make the full chip model for the processor. This model structure can be very complex making the compilation and validation a very challenging job at each level of hierarchy.

2.3 Model Build and Release Operation

Getting the whole picture of the complex model hierarchy we can now understand the model building and release mechanisms. At each hierarchy level the design need to be processed, verified and debugged for any errors. This processing of the design files is called model building. After processing the design need to be fed to the upper hierarchy level, this is called release. This Block diagram shows the legacy model build and release mechanism.

Cluster is a term used within Intel to describe a component of the RTL model of a processor. A cluster is smaller than a full chip, but larger than a unit. Typically, a processor model has several clusters, each responsible for a function like instruction-cache and pre-fetch, integer-instruction execution, floating-point instruction execution, memory-instruction execution, or first-level cache. Design file is the logic design using Hardware Description Language (HDL) also called RTL (register transfer level), and its name derives from the design abstraction level known as the Register Transfer Level (also, RTL). It is important to note that the coding language used is a HDL, the degree of detail that the design is described is RTL (Level), and the resulting design object is the RTL (Logic). Two main HDLs have been used to write RTL, either VHDL or Verilog. System Verilog is the latest enhancement to Verilog.

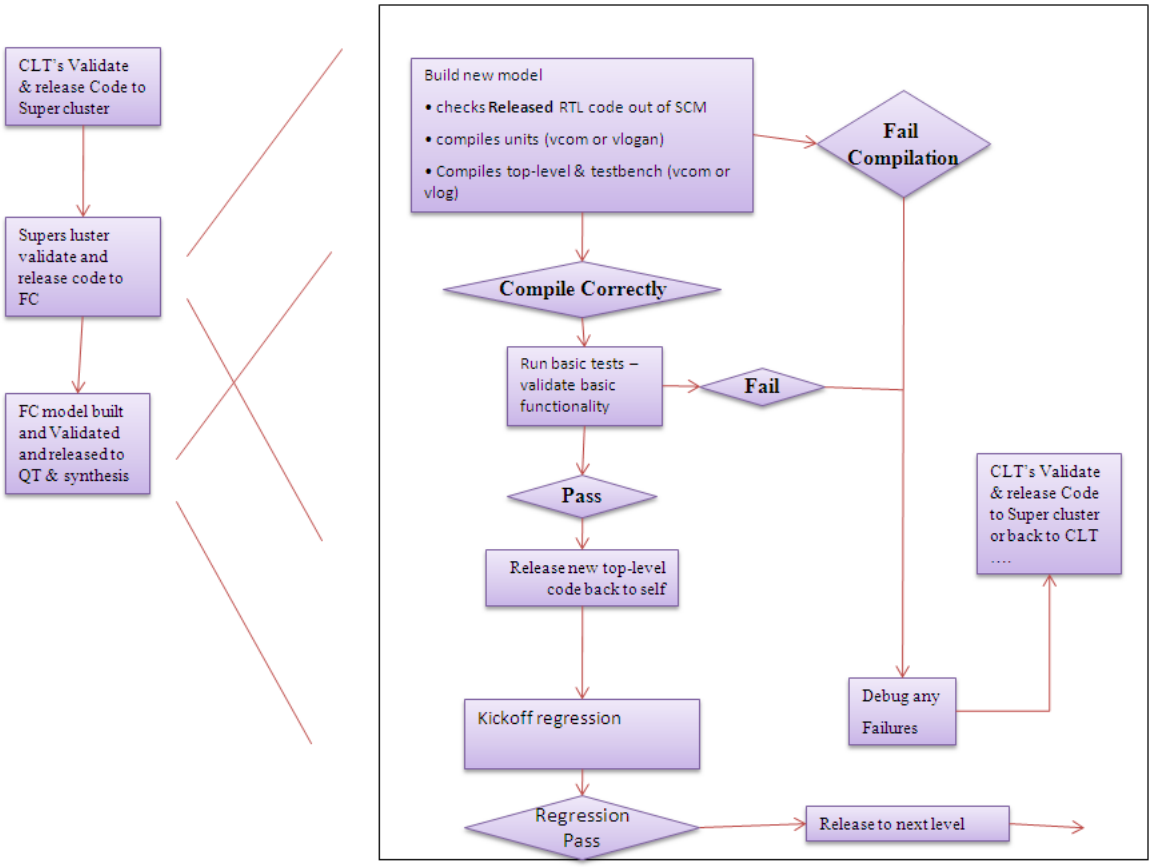

Figure 2.2: Model Build and Release Operation

As we can see from the flowchart given above, there is a main line at the server side which contains the complete fullchip model of the project and it is divided into several big super clusters. It is again divided into several clusters. Each cluster contains around 30 to 40 units and each unit contains some function specific Design files. This way the complete fullchip is partitioned to the lower level units and to the Designs.

In this mechanism the user first takes precompiled and stored files from the SCM mainline server database. The user makes some desired changes to the design files. Now to check if the made changes are valid or not the user need to build a model. In this system we have several legacy model builder and release tools which are used

to build the model and release it to the upper hierarchy levels. This model is built from the code stored in the SCM server, this can be the changed code submitted by the user or a complete clean code which may be used for verification and release only. It can be built from scratch or we can clone some older or other user's model also. After transferring the files from SCM mailine server to the local space in a particular model directory structure, the model is compiled for errors using the vcs compiler. If compilation fails then the designer debugs the error and builds the model again. If the compilation passes than some basic tests are run on the model which test the basic functionality of the model. If this thing also passes then the main part comes where this model is integrated with the top level of module. As mentioned earlier at this level the integration becomes very complex because each project contains more than 200 engineers and a lot of changes are submitted per week. After integrating the changes the exhaustive testing is started on the model. If this regression passes than the code is released to upper level and if any test fails than the designer need to debug the code and resubmit these to upper level. The process of finding the bug in the full chip model is very complex and very time consuming and also the debugging cycle is also very long so it causes a lot of time wastage.

2.4 Distributed vs Centralized Revision Control system

Distributed revision control (DRCS) takes a peer-to-peer approach, as opposed to the client-server approach of centralized systems. Rather than a single, central repository on which workspaces synchronize, each peer's working copy of the code base is a bona-fide repository. Distributed revision control conducts synchronization by exchanging patches (change-sets) from peer to peer. This results in some important differences from a centralized system.

- No canonical, reference copy of the code base exists by default; only working

copies.

- Common operations (such as commits, viewing history, and reverting changes) are fast, because there is no need to communicate with a central server.

Rather, communication is only necessary when pushing or pulling changes to or from other peers.

- Each working copy effectively functions as a remote backup of the code base and of its change-history, providing natural protection against data loss.

Other differences are as follows.

- There may be many "central" repositories.
- Codes from disparate repositories are merged based on a web of trust, i.e., historical merit or quality of changes.
- Lieutenants are project members who have the power to dynamically decide which branches to merge.
- Network is not involved in most operations.
- A separate set of "sync" operations are available for committing or receiving changes with remote repositories.

DVCS proponents point to several advantages of distributed version control systems over the traditional centralized model.

- Allows users to work productively even when not connected to a network
- Makes most operations much faster since no network is involved
- Allows participation in projects without requiring permissions from project authorities, and thus arguably better fosters culture of meritocracy[citation needed] instead of requiring "committer" status

- Allows private work, so users can use their revision control system even for early drafts they do not want to publish
- Avoids relying on a single physical machine as a single point of failure.
- Still permits centralized control of the "release version" of the project

As a disadvantage of DVCS, one could note that initial cloning of a repository is slower compared to centralized checkout, because all branches and revision history are copied. This may be relevant if access speed is low and the project is large enough. For instance, the size of the cloned git repository (all history, branches, tags, etc.) for the Linux kernel is approximately the size of the checked-out uncompressed HEAD, whereas the equivalent checkout of a single branch in a centralized checkout would be the compressed size of the contents of HEAD (except without any history, branches, tags, etc.). Another problem with DVCS is the lack of locking mechanisms that is part of most centralized Version Control System and still plays an important role when it comes to non-mergable binary files such as graphic assets.

Chapter 3

AMI (Automated Model Integration) System

3.1 Why and What

In previous projects we have used legacy model build and release methodology to integrate and release design changes which was ad-hoc, unpredictable, labor intensive and a project bottleneck. In the manual process, the model builder's job was to integrate code changes submitted by various coders on the project into a single model, compile the model, test the model against pre-defined tests for functional correctness, and then release the model (with all the changes) for

a. Integration at next hierarchies/stages

b. Validation of the current changes where more rigorous tests are run to find corner case bugs

This process of manually integrating models is very tedious, labor intensive and requires a lot of coordination among the different coders who submitted code and the integrator. Also, since the process in manual, there is high possibility of creating mistakes and/or oversight. Sometimes the integrator can integrate the model within

a day, but this process can take as long as 2-3 days. Due the manual intensive nature of the above process, model builds are typically done once a week.

In contrast the AMI (Automated Model Integration) methodology is completely automated. It eliminates the need for having specific people (integrators) to just do the manual model build process. AMI automates the process of submitting changes from various coders, ensuring that the changes can cleanly merge with the mainline, build the merged model and then test the final model that is built against a set of tests, push the sanitized code changes to mainline and notify users. AMI also provides immediate feedback to users submitting code if their code is not following the guidelines of functionality, coding guidelines, merge issues, failing tests etc. With such a feedback system and an automated continuous model build system there is a significant productivity boost for code developers.

AMI was in use with other projects at Intel however the project that I was working on was still using the manual process. My involvement in the project was to assist in bringing the goodness of AMI into the project and help in creating specific modules and testing the overall system. I was also involved in helping users to understand the system and debugging issues.

3.2 Difference between AMI flow and Model build flow

As discussed earlier, AMI is an automation that helps in continuous integration of code changes and releasing models. The below 2 figures try to depict the scenario in a manual model build process and an automated model build/release process (AMI)

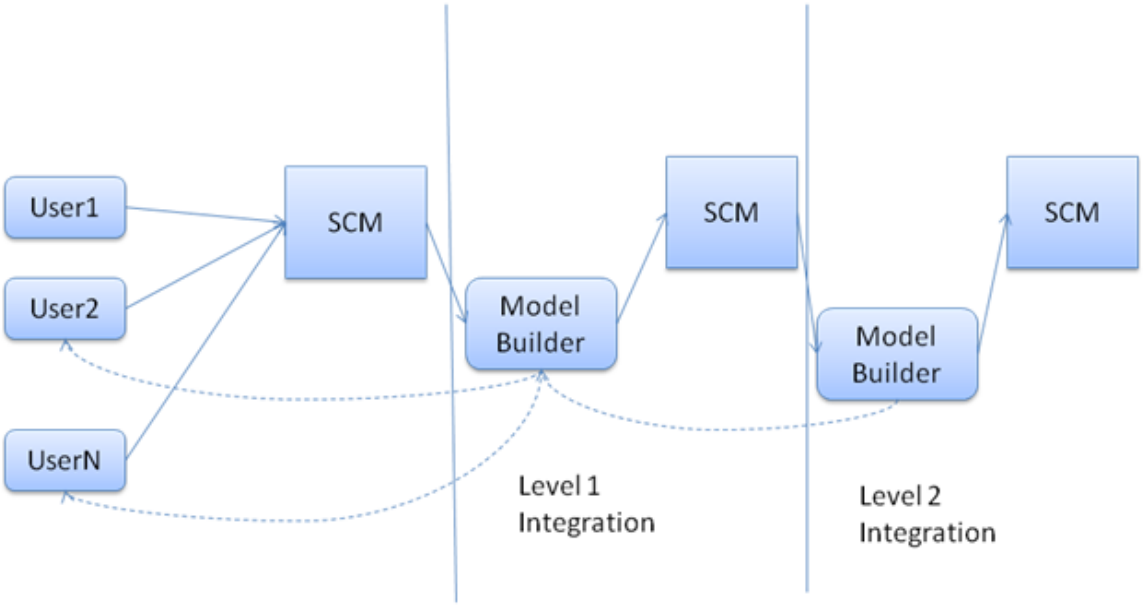

Figure 3.1: Model Build flow

Here in the above figure we can see that with the help of AMI we have replace the model builder with a server and an automated tool. In the case of AMI as and when the design engineer submits the code the server automatically integrates the model and does the regression tests as opposed to the case in manual model build where model builder integrates the models and then manually applies the regressions and finds the bugs. As evident in the above figures, the manual effort of all the integrators is avoided in AMI and hence saves resources and time for the project.

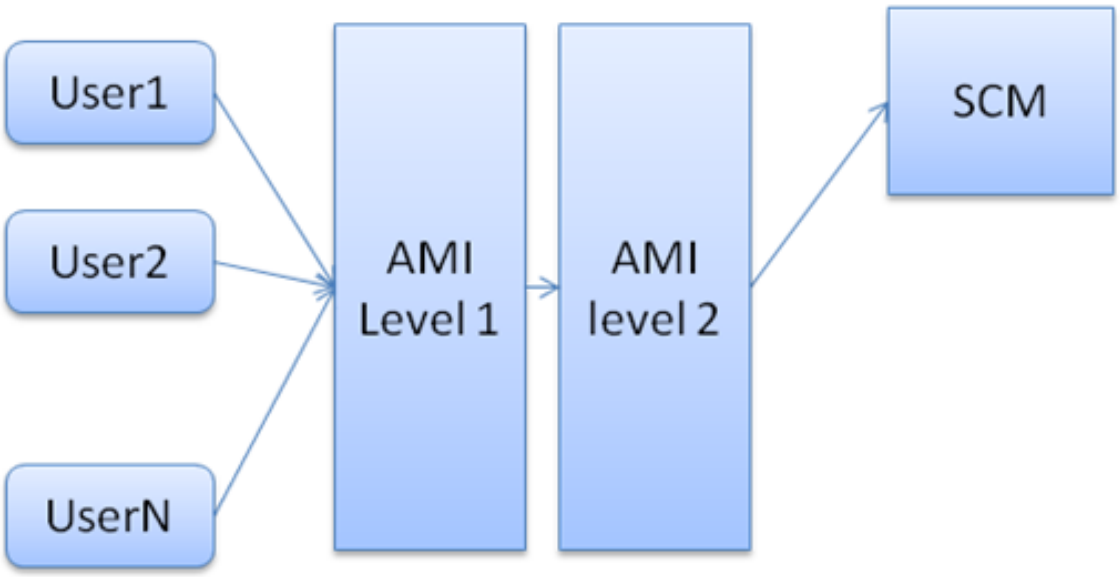

Figure 3.2: AMI flow

3.3 AMI Implementation bottleneck

AMI, which was developed for other projects, could not be used directly for the current project. There were some bottlenecks and my assignment was to assist in creating modules that helped alleviate some of these bottlenecks.

- Inter Design dependency

  The complete project design is divided into big hierarchy which contains several levels like units – cluster – super cluster – full model and each unit is owned by individual owners. As like in any design units share interfaces with other units across the design (cluster, super cluster, full chip) and hence any changes in one unit needs to be in sync with changes in other unit. This dependency of one unit with the other is hard to resolve since many individuals are involved.

- SCM file locking problem

  The project used SCM such that a file is locked when one user is using it. When any unit owner checks out the files and opens the file for editing the files are locked in his name and at the same time the files cannot be opened by any

other user for editing. This may cause a serious problem if some files remain unintentionally by any user or if there are conflicts in changes which again lock the files.

- SCM server overloading

  There are a lot of systems working on the SCM server simultaneously. These systems create a lot of tags in SCM and the users working on design editing create a lot of workspaces in SCM server. The AMI system also works on the SCM server and if the SCM server is overloaded; it can cause the failure of the model building or checking out of files.

Chapter 4

DRH(Dpendency Resolution Helper)

4.1 DRH

The dependency problem in the model building operation is one of the biggest problems which can cause the failure of the submission even if the change made to the design code is valid change. These problems are harder to debug. Today in the case of dependencies the unit owners co-ordinate with each other and try and intimate each other to select to submit their changes, manually (through emails). This is very time consuming process. We augmented the AMI system with a wrapper called DRH (Dependency Resolution Helper). This system tracks the dependency between the units or submissions and stops the submission if there is any dependency which is not yet fulfilled. This system also notify the owner of the mast unit on which the submission was dependent so that the master unit owner can make the changes in his unit quickly and submit his code first to AMI and fulfill the dependency requirements.

4.2 DRH operation

The dependencies can be of three types

- Cross Unit dependencies
- Unit to library dependencies
- Unit to software model dependencies

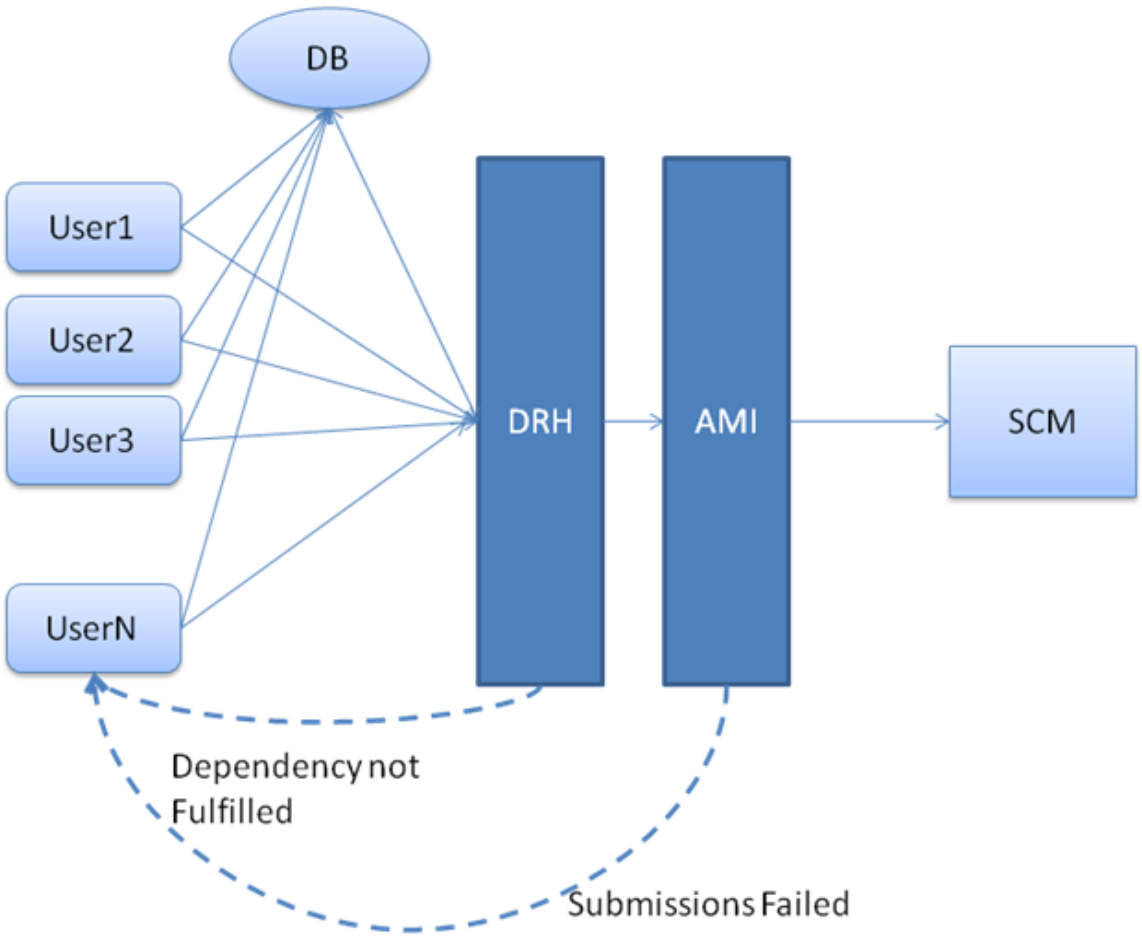

Figure 4.1: DRH (Dependency Resolver Helper)

In DRH system we used a database to track the dependencies between units. The unit owner (U1) having dependencies makes an entry in the database. It is called a bug. The owner U2 creates a master bug which has all the bugs which are dependent on his changes. DRH actually accesses this database and finds if there is any master bug

present on the child bug submission. If it is then it rejects the submission without sending anything to the AMI and notifies the master bug owner that he needs to submit the changes.

The features of DRH are given below.

- DRH enables users to add dependency information in the database
- Automatically tracks completion of dependencies and notifies users
- Users can then submit code to AMI as a single package
  AMI will build and regress changes with the latest released code and on success, integrate and release the new code as latest.

Chapter 5

Branching and Merging in SCM

Branching is a method of maintaining the relationship between sets of related files. Branches can evolve separately from their ancestors and descendants, and you can propagate (integrate) changes from one branch to another as desired. SCMs branching mechanisms preserves the relationship between files and their ancestors while consuming minimal server resources. SCMs branching mechanism enables us to copy any set of files to a new location in the repository by allowing changes made to one set of files to be copied, or integrated, to the other.[6]

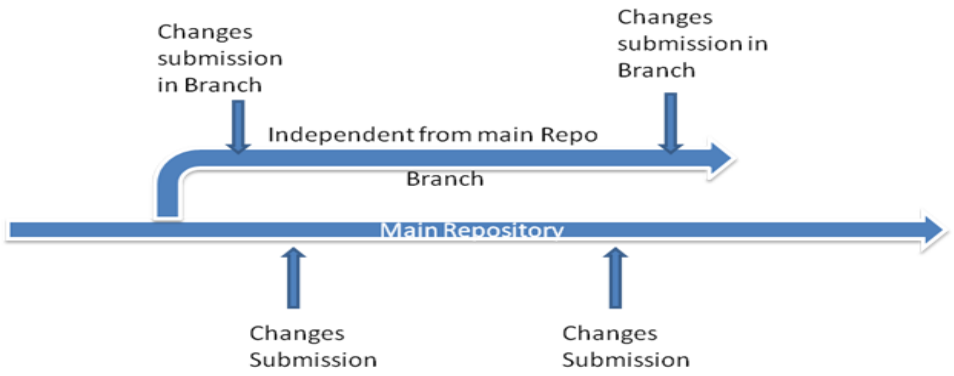

Figure 5.1: Merging the Branch changes to mainline

A file is uniquely identified by its complete filename, e.g., /repository/trunk/src/item.cpp. Any non-deleted revision of a file can be branched. Branching creates a new file with a new name. For example, my/index.php may be branched into your/index.php and each file may then evolve independently. Repository paths are typically designated as containers for branched sets of files. For example, files in the /repository/trunk path may be branched as a set into a new /repository/rel1.0 path, resulting in two sets of files evolving independently and between which changes can be merged.[4]
In SCM the operation that merges changes from one branch to another is called integration. Integration propagates changes from a set of donor files into a set of corresponding target files; optional branch views can store customized donor-target mappings. By default, integration propagates all outstanding donor changes. Donor changes can be limited or cherry-picked by changelist, date, label, filename, or filename pattern-matching, The system records all integrations, uses them to select common ancestors for file merging, and does not by default perform redundant or unnecessary integrations.[4]
Merging is actually only one of three possible outcomes of the integration. The others are ignoring ("blocking") and copying ("promoting"). Merging is used to keep one set of files up to date with another. For example, a development branch may be kept up to date with its trunk through repeated merging. Ignoring disqualifies changes in one set of files from future integration into another. It is often used when a development branch must be up to date with, and yet divergent from, its trunk. Copying is typically used to promote the content of an up-to-date development branch into a trunk.

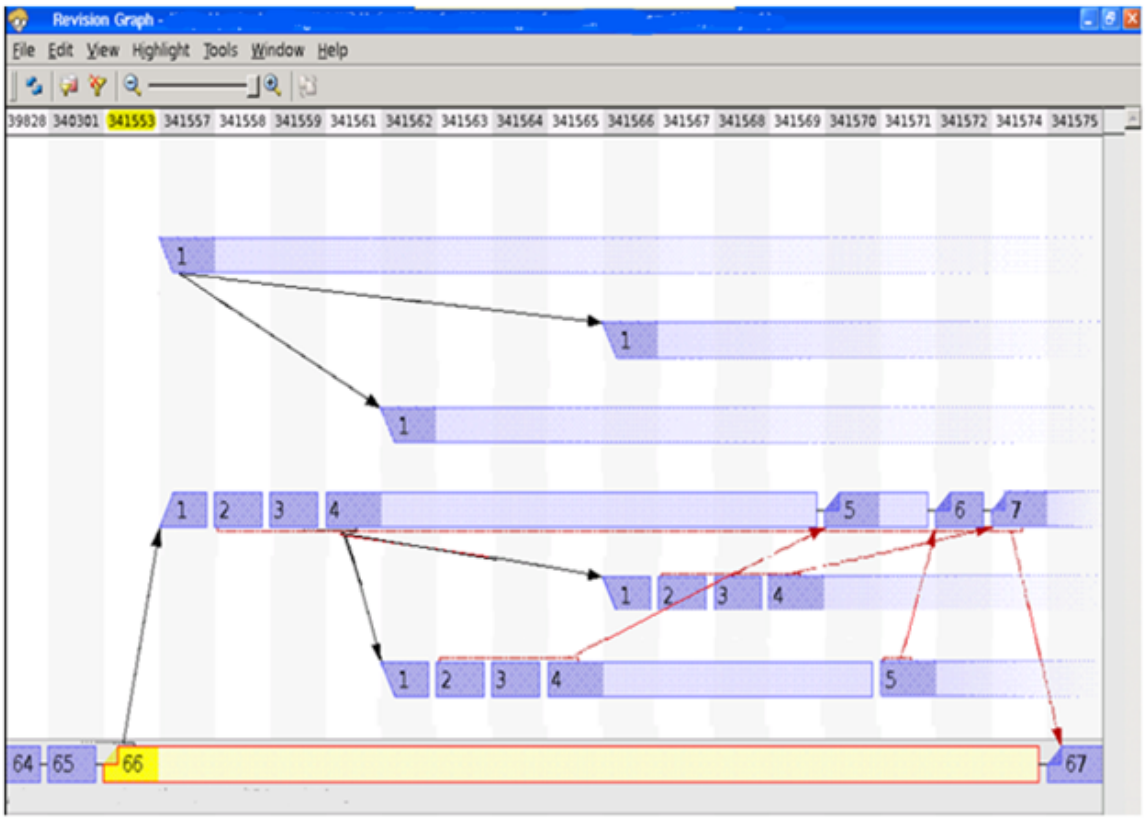

Figure 5.2: Revision Graph

5.1 Integrating Files

When you've made changes to a file that need to be propagated to another file, start the process with "scm integrate". The simplest form of this command is scm integrate from File toFile; this lets the SCM server know that changes in fromFile need to be propagated to toFile, and has the following effects:[9]

- If toFile doesn't yet exist, fromFile is copied to toFile, and then toFile is opened for branch in the workspace.
- If toFile exists, and was originally branched from fromfile as above, then toFile is opened for integrate. You'll then use scm resolve to propagate all of, portions of, or none of the changes in fromFile to toFile. The scm resolve command will

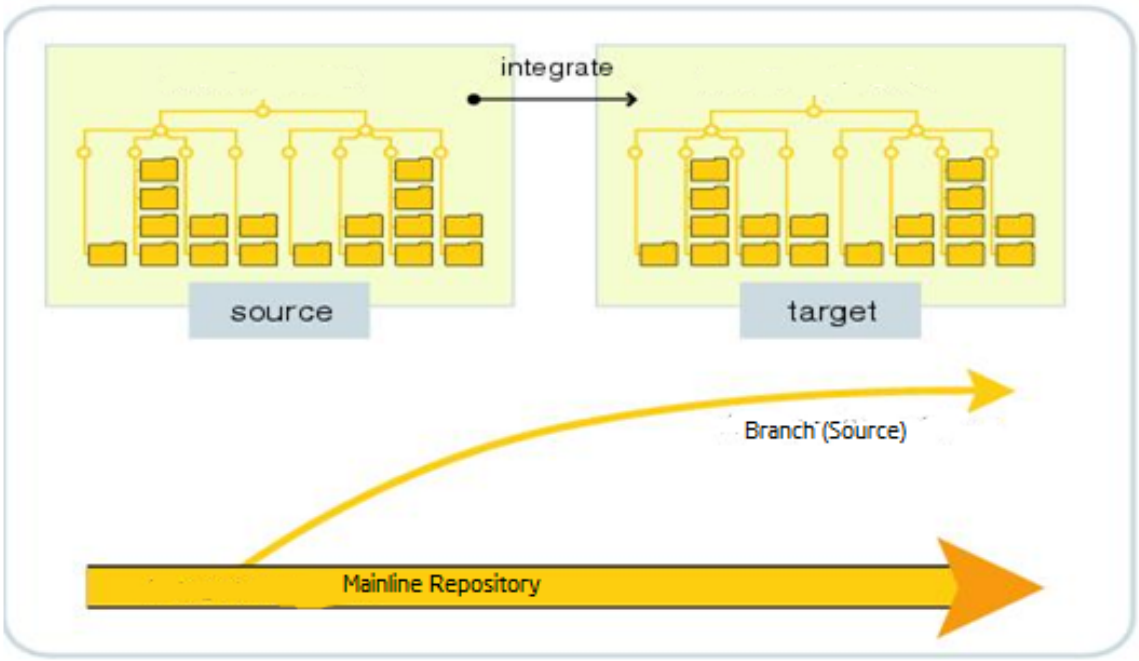

Figure 5.3: Integrating branches

use fromFile as theirs, toFile as yours, and the previously integrated revision of fromFile as base.

- If both toFile and fromFile exist, but toFile was not originally branched from fromFile, the integration will be rejected.
- If fromFile was deleted at its last revision, toFile will be opened for delete in the workspace.

5.2 Branching Behind the scene

When we branch a set of files, although it appears that SCM is copying all the files, (e.g. there are 1,000 files in /repository/main/... and it appears as if there are now another 1,000 files in /repository/r1.0/..., giving 2,000 in total), that the newly branched files are just virtual copies. SCM has duplicated the metadata in its database to say that all the /repository/r1.0/... files now exist, but they are just

pointers to specific version of the real files in /repository/main/....
Behind the scenes, the /repository/main/... files actually have RCS format archives (for text files) to store all the details about the contents of each revision (binary files are normally stored as a compressed version of each revision - see "scm help filetypes" for more information). The files in /repository/r1.0/... only acquire their own archive file on the server when they are modified (scm edit and then submit). Thus for typical situations where you branch 1,000 files and yet change less than 100, you only get less than 100 new archive files on the server.[1]

5.3 Advantages of Branching and Merging

- Resolves the problem of file locking in SCM by providing separate branch to each user.
- Enables Simultaneous editing of the design files.
- User can have complete cluster level model at local space and more than one user can combine their changes in one main.
- The changes can be checked in the main branch before sending to mainline.
- Selective submission of the changes to the mainline. So no manual editing if the dependent changes are made, therefore branching can also be used to resolve the dependency problem.
- The changelist generated are independent of the main line changelist so they are not reflected in the main line repository change number. So the changes made in the branch don't affect the repository files till we submit them to the main line.
- It can make the Design files editing and submitting more efficient and can save a lot of time of the designer.

5.4 Steps for Branch and merge

- Step 1: Creating a branch

  To create a new branch we use the SCM command scm branch (branch_name) Example: scm branch surajput_branch_1 It opens the branch specification window in which we need to set the branch view corresponding to the depot directory. Example: depot/rel1.0/unit1/ depot/rel1.0/sandip_branch_1/ Now save this file it will save the branch specifications.

- Step 2 : Set the SCM workspace for the branch

  Now we need to set the SCM workspace as we do in general case but in this we will set the workspace view to the branch directory not the depot main file directory. Example: scm workspace surajput_workspace_1 Now in the workspace view we set //depot/rel1.0/sandip_branch_1/ /sandip_workspace_1/

- Step 3: Integrating a branch

  To integrate a branch we use the SCM command scm integrate -b (branch_name) Example: scm integrate -b sandip_branch_1

- Step 4 : Submit the change number in the branch

  This thing is same as in the general case. We use scm edit command to open a file than we edit and make some changes and using the scm submit command we submit the changes to the branch.

- Step 5: Sync the files in one new workspace

  Set a new workspace using the scm workspace command and using scm sync command sync the depot files into the workspace. In this workspace the branch files will be merged.

- Step 6: Merging the branch with the workspace

  Now we can integrate our branch's latest change to the depot file using the scm integrate command. We can also specify the changelist we want to sub-

mit if we don't want the latest as a default to merge scm integrate //depot/rel1.0/sandip_branch_1/ //depot/rel1.0/unit1/ It opens all the files which have the changes in the branch for the merging.

- Step 7 : Resolve the conflicts

  We need to resolve all the conflicts using the SCM scm resolve command. It gives several options to accept reject merge or edit the changes in the branch file and the workspace file. Example: scm resolve -v /root/home/sandip/unit1/unit1.v - merging //depot/rel1.0/unit1/unit1.v#51 Diff chunks: 1 yours + 1 theirs + 0 both + 0 conflicting Accept(a) Edit(e) Diff(d) Merge (m) Skip(s) Help(?) e:

- Step 8 : Submitting the merged files Now using the scm submit command we can submit the merged files which generates a changelist which is in the main line.

5.5 Branching and Merging System

The Branching and Merging system is a Perl script which uses the SCM Perl API, Perl TK module (for the conflict resolver GUI), The Branching and merging system contains main four operations 1. Clone files 2. Edit files 3. Commit files 4. Push files to the mainline

- Clone

  Command: scm -clone In this operation the user can integrate the files from the depot to the workspace in a branch. It transfers all the files of the model to the current directory. This branch is separate from the mainline so the changes done in this branch are separate and does not affect the mainline files. It takes the unit name from the environment variable. We get SCM depot paths and all units which need to be integrated in this cluster level model. It saves the workspace name and the branch name into a file .scm workspace in the user's home area. It also submits the initial branch files to the branch all with the

version 1. It prints the change number generated by SCM. This change is the initial change of the initial Branch (Branch creation change). This changelist generated is not reflected in the mainline. This command works at unit and cluster levels.

We can also provide extra switches like -f (from) with the clone command. When we want to take the cluster level model of any other user or we want to make changes in any other Branch model, we can use this -f switch. Using this we can provide a branch name in the command line and the system will make a sub branch of the main branch. It will integrate all the units of the cluster from the given branch to a new branch. This way we can clone any other user's model also.

- Edit

  Command: scm edit (filename)

  After executing the clone command we have got the complete cluster level model. By default all the files which are transferred from the server to the local space are in read only mode so the user need to first open them for edit. The designer can use scm -edit command to edit the branched files.

- Commit

  Command: scm commit

  This command submits the changes done in the branched files. It generates one changelist and prints last 5 change number generated in the branch. These change number generated are separate from the mainline change number.

- Push

  Command: scm -push

  This command is used to merge the changes made in the branch into the mainline. This command syncs the complete model in the current directory. It now integrates the branch to the synced files and opens up the conflict resolver GUI.

This GUI contains the list of files with all the resolve options (at, ay, am, d, e) as radio buttons. After selecting the file in the list and selecting the resolve option it does the resolve for that file. The diff option opens the files in the scmmerge tool. In this tool we can see the diff between our changes and the auto merge file. The edit option opens the auto merge generated fie with the merge markers and all the changes in gvim editor. The user can edit the changes which to take and which to remove from the file. The user also needs to remove all the merge markers before he saves the file exits the file editor. This submits the edited file. When the file list is empty in the CR GUI, one exit button comes up and user can exit the GUI. After this the resolved files are submitted to the mainline and last 5 change number generated by the user is printed as output. While pushing the files the designer need to resolve the conflicts between the branch files (which he edited in the branch) and the mainline files. It is possible that the mainline files may have got updated or changed during the time when he made this branch from the mainline and the time when he is pushing back his changes to the mainline.

If the designer try to do this manually it takes a lot of time to manually go and resolve each file. I have automated this in the Branching and merging system using the Conflict Resolver. The Conflict Resolver is a GUI which I made using the Perl TK module.

- Conflict Resolver

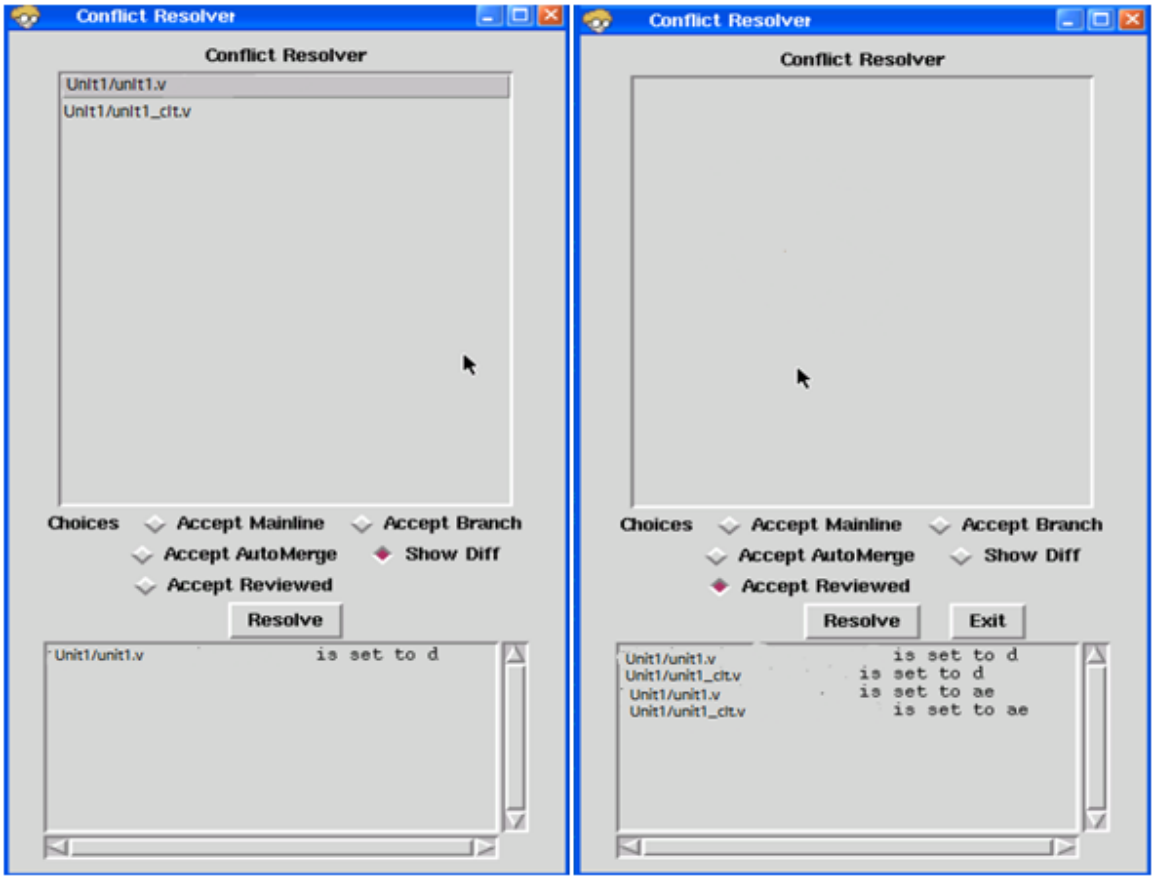

Figure 5.4: Conflict Resolver

As we can see in the snapshots of the conflict resolver tool, there is one list box which contains the list files which were edited in the branch by the designer. These are the files which are to be merged with the mainline files. Below the list there are 5 radio buttons which are the resolve options the designer can choose. These options are similar to the command line options the designer can provide to resolve any file. The accept mainline button ignores the branch changes, accept branch button simply copies the branch file to the mainline synced files, accept automerge button will accept the SCM generated auto merge file (this file

contains both files changes and all the line by line conflicts are overwritten by the branch files). The show diff button opens both the files and the automerged file in the SCMMERGE tool. This tool is shown in the snapshots given below. This tool shows the diff of the mainline and the branch files. Using different switches given in the tool we can take any files changes and save the reviewed file. There is one more radio button accept reviewed. Clicking this button resolves the conflict by submitting the review file which we have saved from the SCMMERGE tool. After the tool has resolved any file it removes those files from the files list. When the resolve files list is empty, one new switch with the 'exit' label comes up. The designer can press this exit switch and can exit the conflict resolver. This automatically submits all the resolved files to the mainline and generates a change number. This way the user can merge his Branch changes to the mainline files.

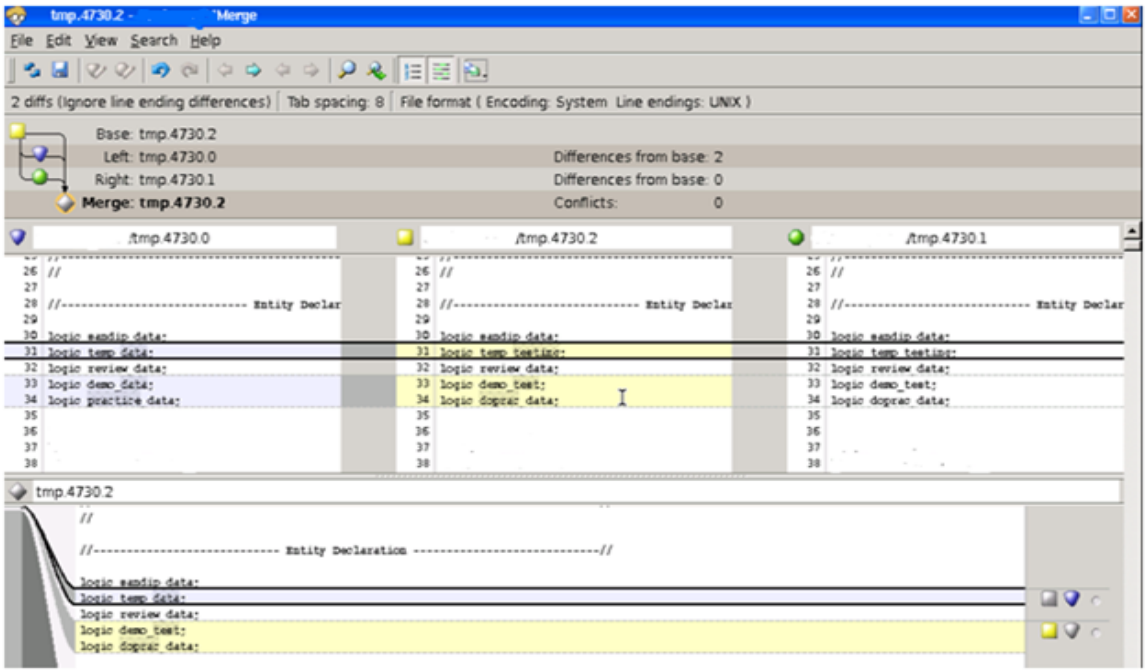

Figure 5.5: scmmerge tool

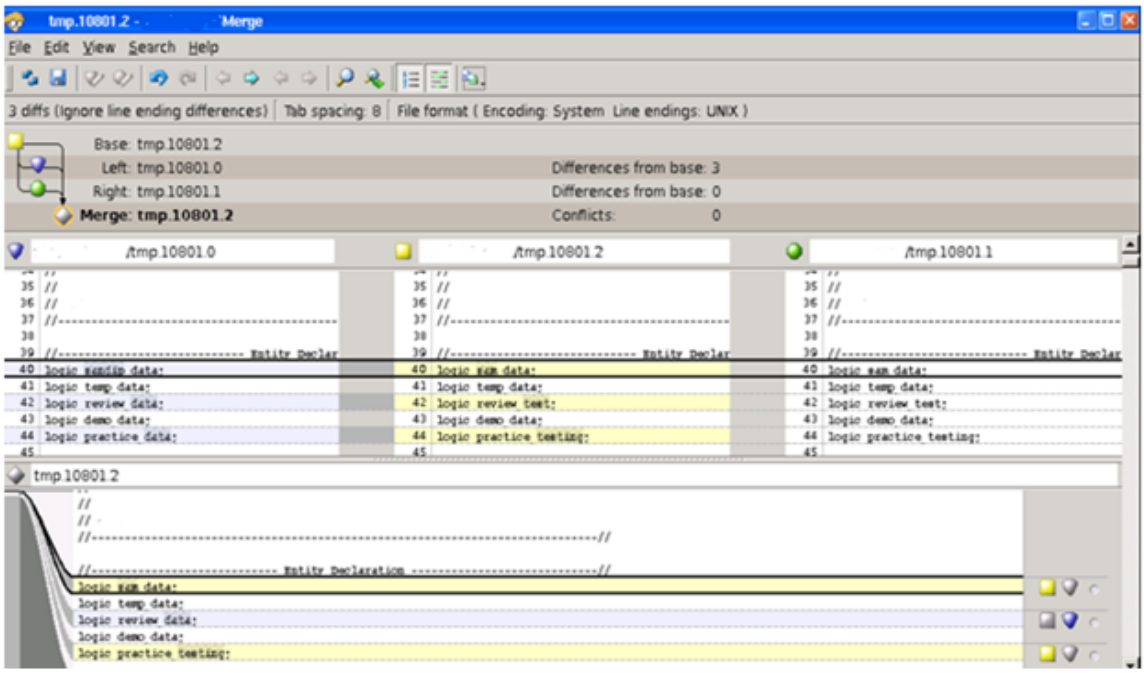

Figure 5.6: Merging changes in scmmerge tool

5.6 Adding Distributed revision control system like operation

Using the branching and merging system any user can create his branch at cluster level from the mainline files or from any other user's branches. He can simply give option -f (from branch) with the scm clone command, it will clone the given user's branch in a new branch with all the files with the latest changes made in the branch. The user can edit his branched files and submit it to the branch itself. When the changes are checked in the branch and satisfactory he can merge his branch changes to any other user's branch or to the mainline. To do this the user needs to provide command line switch -t (To Branch) with the scm push command. It will open the conflict resolver GUI to merge the changes in the branch to the given branch. If no option is provided with the push command, by default the user's branch is merged

to the mainline files.

In Branching and Merging system the user also has the flexibility to provide -t (To Branch) -f (From Branch) with the scm push command. Using these options the user can submit any of his branches to any of the other user's branch irrespective of the current branch he is working on. This branching and merging operation when introduced in SCM adds a distributed revision control system like feature in which user's have their own repository branch and they can transfer their model to any other model and can take a new model from any other model.

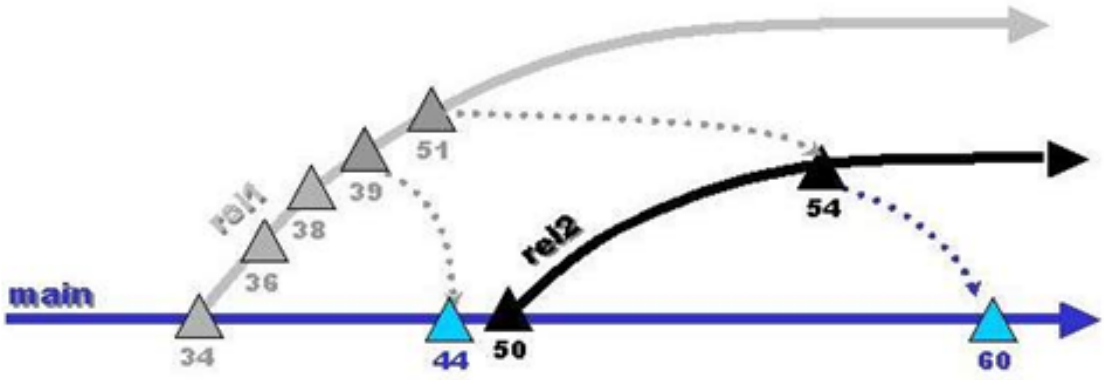

Figure 5.7: Distributed RCS like Operation

Chapter 6

Archival of the SCM tags and workspaces

SCM system supports tagging mechanism which user can use to label any of the SCM database files or users edited files so that he can have a benchmark of the files. The user can anytime checkout a particular tag and can get the exact version of the files which he tagged. The SCM also has the client workspace mechanism in which the user need to create a client before he checks out the SCM files and start editing or submitting them.

In a multi-core microprocessor development there is huge number of tools and systems used like nightly model build, verification tools, debugging tools, etc, which work with the SCM files. These systems and even SCM users generate a large number of tags and workspaces which get stored in SCM server. These workspaces and tags are generally left behind and not deleted from SCM which can decrease the performance and efficiency of the SCM server. The more number of tags and workspaces the slower the response of the SCM server. The AMI system also depends on the SCM server to check out the files, tag them as appropriate and then submit them back to the SCM server as needed. If the SCM server is loaded then it may cause the failure of the submissions which we have seen while testing on the earlier projects.

I worked on the SCM tags and client workspaces archival system which reduce the load from the SCM server by reducing the number of workspaces and tags from the mainline server.

6.1 SCM tag/ workspace archival system operation

The administrator in SCM server can delete anyone's workspaces or tags and can reduce the load from the server directly but the problem with this operation is, we may also lose some very important and critical tags or workspaces. The designers wanted a system which can give them their tags as they were after getting deleted. So the exact requirements of this task were:

- The tags and the workspaces the archival system deletes must be older then a particular date given by the management. So I needed the access and update times of the tags and workspaces from which I can get which to select for deletion.

- Develop a system which not only deletes the tags and workspaces but also store the important data which the client and tags were having. There should be some secured data storage mechanism which can take input the data and also when called can output the particular data so that we can restore the tag/client.

I developed a system which is using SCM Perl API and it is parsing the tag form info, from this information, according to the access and update dates I am listing down the tags of workspaces which I want to delete. I also extracted the list of files which were tagged by that particular tag. The system uses Perl modules to communicate to the database and transfers the complete data which I want to store, in a table. After storing the data the system deletes that particular tag.
At the time of restoration the user need to execute the script and give the name of

the tag as a command line input to the system. The system checks if this entered tag was deleted by it, if yes it communicates with the database and executes the DB queries. After getting all the important information the system creates a new tag with everything same as the deleted tag.

The archiving system generally has 4 steps:

- Listing down the complete tags/clients
- Sorting out the tags/clients for deletion
- Archiving the sorted tags/clients
- Restoring the entered tag/client

6.2 Archival Implementation

- Tag Archival Using the Archival system I archived around 70000 tags which were dated 2008 or older. The archival process queries the SCM database for all the tags and their access and update dates which loads the SCM server heavily. To avoid any overload failures I did the archival process in the weekends when there is low load on SCM server.

Data	Listing all tags	Sorting tags for deletion	Remove tags	Restore tags
Start time	Sat Mar 12 10:45:03 IST 2011	Sat Mar 12 10:49:33 IST 2011	Sun Mar 13 18:23:19 IST 2011	Wed Mar 16 15:38:48 IST 2011
End time	Sat Mar 12 10:45:28 IST 2011	Sun Mar 13 17:44:20 IST 2011	Tue Mar 15 12:24:14 IST 2011	Wed Mar 16 15:40:16 IST 2011
Time elapsed	25secs	30 hrs 55 min	42 hrs	~2 min
Number of tags	**483345**	**69126**	**69126**	

Table I: Tag archival

- Client Archival

Data	Listing of clients	Sorting the clients for deletion	Archiving clients
Starting time	Sun Mar 27 13:57:58 IST 2011	Sun Mar 27 13:58:39 IST 2011	Sun Mar 27 15:53:57 IST 2011
Ending time	Sun Mar 27 13:58:03 IST 2011	Sun Mar 27 14:49:39 IST 2011	Sun Mar 27 16:22:46 IST 2011
Time consumed	5sec	51 min	28 min
Number of clients	**12248**	**2280**	**2280**

Table II: Client Archival

Chapter 7

Results, Achievements and Conclusion

I was involved in DRH module enhancements and testing. My involvement in enabling AMI for the current project was in the areas of testing the pilot and helping users in debugging issues when AMI rejected submissions. I also worked on the automation that was used to detect the effectiveness of AMI on the current project pilot. This was needed since AMI was not deployed and users were free to use the manual system or the AMI. SCM branching and merging wrappers I created was one of my other key contributions that would alleviate the issues with the SCM locking mechanism and thus indirectly aid in the success of AMI on the current project. I also created automation to clean-up and archive SCM tags/workspaces thus increasing the performance of SCM on the project. Key results are discussed below

7.1 Results

When implementing the combination of AMI and DRH in a current multi-core microprocessor project for a week and found these results.

	Submissions to the AMI
Accepted Submissions	29
Rejected Submissions	8 (Dependency detected by AMI and resolving the dependency, submission passed)

Table I: AMI implementation results

The average time taken for the accepted submissions was very less (in order of hours) when compared to the manual process where users would have to wait for a week before they got to know if their code submissions were good or not. Also, the above result shows that by AMI rejected around 8 submissions that depended on other submissions. We were able to catch these in DRH.

SCM Tag archival results:

The results of the SCM Tags and workspaces archival are also very good. I have done the comparison of model building time before and after the tag archival.

Tests	Starting time	Completion time	Time difference	Results
Model build Before Archival	Wed Mar 9 11:02:38	Wed Mar 9 11:16:43	14 min 5 sec	
Model Build After Archival	Thu Apr 21 14:48:18	Thu Apr 21 14:58:25	10 min 7 sec	**3 min 58 sec FASTER (28% less time)**
SCM DB query for Label Before Archival			25 sec	
SCM DB query for Label after Archival			18 sec	**~28% time saving with Archiving**

Table II: Tag Archival Results

7.2 Achievements

- Received an award for my work on AMI and DRH
- Received an Award for the good work in "SCM branching and merging" and "SCM tag/workspace Archival".

7.3 Conclusion

The best known methodology to ensure a better and higher quality design is 1) Giving more time and flexibility to the designers so that he can concentrate more and more on the design development, 2) Removing all other hurdles, bottlenecks and extra manual operations which come in the design development process and can be automated using tools.

The legacy model builds and release mechanism which we are using is not changed since last several years therefore it has a great scope of enhancements and improvements. Introducing the AMI with the model build methodology can automate the integration process and increase the speed with accuracy. The added solutions like dependency management system, makefile system, SCM branching and merging and SCM tags/workspaces archival can revolutionize and the new model building methodology.

References

1. http://www.vaccaperna.co.uk/scm/branching.html
2. http://en.wikipedia.org/wiki/Build_automation
3. http://book.git-scm.com/3_basic_branching_and_merging.html
4. http://en.wikipedia.org/wiki/Perforce#Branching_and_merging
5. http://en.wikipedia.org/wiki/Continuous_integration
6. http://www.perforce.com/perforce/doc.current/manuals/p4guide/06_codemgmt.html
7. http://www.vance.com/steve/perforce/Branching_Strategies.html
8. http://en.wikipedia.org/wiki/Software_configuration_management
9. http://www.perforce.com/perforce/doc.current/manuals/cmdref/integrate.html
10. http://en.wikipedia.org/wiki/Distributed_revision_control
11. http://clearcase.weintraubworld.net/cc.branch2.html
12. http://ioctl.org/unix/cvs/branches/