

Model-based Version Management System Framework

W. Mehmood¹, A. Ali², A. Qayyum³, M. E. Quershi⁴

^{1,2,3,4}Computer Science Department, COMSATS Institute of Information Technology, Wah Cantt. Pakistan
¹drwaqar@ciitwah.edu.pk

Abstract-In this paper we present a model-based version management system. Version Management System (VMS) a branch of software configuration management (SCM) aims to provide a controlling mechanism for evolution of software artifacts created during software development process. Controlling the evolution requires many activities to perform, such as, construction and creation of versions, identification of differences between versions, conflict detection and merging. Traditional VMS systems are file-based and consider software systems as a set of text files. File-based VMS systems are not adequate for performing software configuration management activities such as, version control on software artifacts produced in earlier phases of the software life cycle. New challenges of model differencing, merge, and evolution control arise while using models as central artifact. The goal of this work is to present a generic framework model-based VMS which can be used to overcome the problem of tradition file-based VMS systems and provide model versioning services.

Keywords-Software Configuration Management, Version Management System, Model Diff, Model Merge, Evolution Control, Model-Based VMS

I. INTRODUCTION

To develop large software projects (in which more than one person participate), it essentially needs the efficient management of software artifacts created during software development. In the absence of controlled management, the software products that the industry has turned out can be delivered much later than scheduled, may cost more than anticipated and would have been poorly design and documented [xi]. Version management system (VMS) aims to provide a controlling mechanism to such problems. VMS deals with controlling the evolution of software systems. Controlling the evolution requires many activities to perform, such as, construction and creation of versions of the software artifacts, performing **model diff** activity (i.e., the identification of differences between versions), conflict detection, and merge activity (i.e., combining two or more versions) [i].

With the advent of modern software development

methodologies, such as model driven engineering (MDE), models become first-class artifacts. Therefore, performing VMS activities on models are essential; however, existing file-based VMS systems are not adequate for performing such activities on models. Fundamentally, the main reason of inadequacy of existing systems, such as Subversion [x], is due to the fact that these systems are file-based and consider software artifacts as a set of text files having no relations. In contrast, models are graphs with nodes being complex entities and arcs (relations) containing a large part of model semantics. File-based tools use textual or structured data to represent models at fine-grained level. This representation is not suitable for diff and merge operation of models due to several reasons. For instance, in MDE, software documents are not only text files, but also consist of diagrams such as different types of UML diagrams. These diagrams are often stored as XMI formats, such as a class diagram might be represented by a few lines of text in the file. The order of these sections of text is irrelevant in a file and the CASE tools can store the sections representing classes or other diagram elements in arbitrary order. To a large extent, the order of text lines and their layout is immaterial for diff and merge operations on models. Therefore, applying diff and merge operations at the level of plain text would hardly produce meaningful results. Therefore, the goal of this paper is to develop a generic fine-granular model diff solution for class and activity diagrams of the unified modeling language (UML). The model diff deals with comparing the two versions to detect the differences and matches between them. It is an important and challenging task in the MDE. The traditional VCS systems are text-based systems and are not designed to operate adequately on models. Therefore, in this paper we propose an approach that handles the model structures adequately.

The goal of this work is to develop a generic framework to deal with the issues of model diff, merge and evolution control activities in model-based VMS system. At a fine-grained level we represent our models as graph structures, which is an intermediate representation based on graph theory. The diff, merge and evolution control activities are performed at the level of graph structures, whereas versioning activities should remain at textual or structural representation,

such as XMI-files. By doing so, on one hand we are getting the advantages of reusing the traditional VMS systems for versioning purposes and on other hand we avoid the problems associated with textual or structured representation when performing the rest of the activities. Evolution control mechanisms are defined based on intra & interlink dependencies between models element. The innovative aspects of the approach are generality, traceability between models element through intra & interlink dependencies, definition of evolution control mechanism and reusability of existing VMS systems.

The organization of this paper is as follows: Section 2 describes the related work. Section 3 presents the main components of our framework. Section 4 describes reference architecture. Finally a short conclusion and future work is given in the last section.

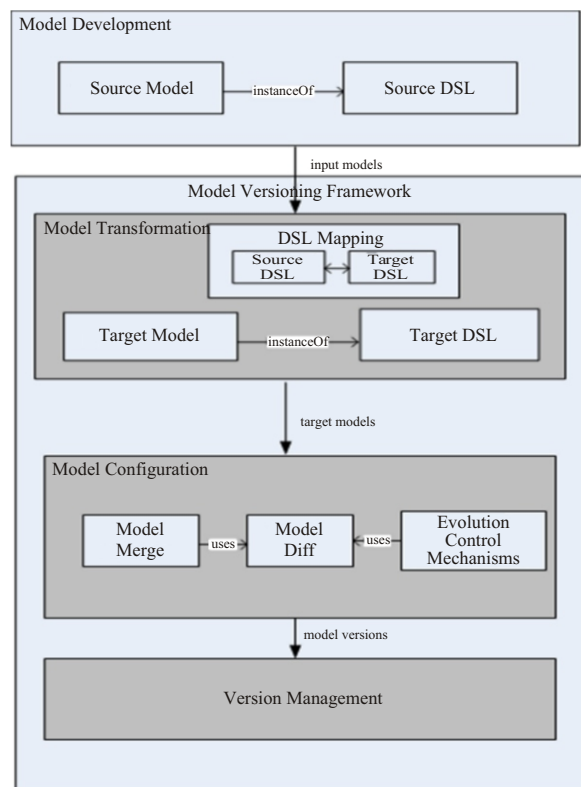


Fig. 1. Model-based SCM Framework

II. RELATED WORK

Many solutions to model-based SCM exist in literature. In this section we will describe the existing solutions. Alanen and Porres in [xiii] discuss the difference and union of models in the context of a version control system. Three meta-model-independent algorithms are given that calculate the difference between two models, merge, and calculate the union of two models. However, these algorithms crucially rely on the existence of a universally unique

identifier for each model element. The output produced by the approach is in form of a sequence of edit operation while in our approach the results are brought back into a model which is more comprehensible for understanding. The approach also does not detect shifting of elements between models and detect shift operation as delete-add operation. Ohst et al. [ix] address the problem of how to detect and visualize differences between versions of UML documents, such as, class or object diagrams. The approach assumes that each model element has a unique identifier which is used for model comparison. For showing the differences between two documents the unified document is used which contains the common and specific parts of both base documents; the specific parts are highlighted. EMF Compare [xiv] is an open source tool used EMF technology project to compare models in EMF. It is realized by a package of Eclipse plugins that overwrite Eclipse's standard comparing behavior. EMF Compare uses a generic algorithm for model comparison. The comparison is performed in two-phases: In the first phase the match engine tries to find similar elements and creates a match model. Based on this model the difference engine is used to generate detailed information about the differences of certain model elements. A difference model is the result of the second phase. Both match and difference model are EMF models and therefore can be treated like any other model. As compared to our approach the diff and match model produced by EMF Compare cannot be converted to graphical representation as done in our approach. Furthermore, EMF Compare also suffers from the sensitivity issue of layout or order changes. A detailed empirical comparison of our approach with EMF Compare is already given in Section 4 which shows the performance efficiency of our approach. Xing et al. [xv] presented an automated UML-aware structural-differencing algorithm, UML Diff. UML Diff is an algorithm for automatically detecting structural changes between the designs of subsequent versions of object-oriented software. It takes as input two class models of a java software system, reverse engineered from two corresponding code versions. The approach uses a language-based matching criterion and identifies corresponding entities based on their name and structure similarity. If two objects have same name, they are identified as equal, if not, their structural similarity is considered, computed from the similarity of names and other criteria specific of the considered entity type. Kelter et al. [xvi] presented a generic algorithm SiDiff which uses an internal data model comparable with a simplified UML meta-model. A diagram is extracted from an XMI file and is represented as a tree consisting of a composition structure. In this approach the model elements are characterized by the elements they consists of, the difference algorithm starts with a bottom-up traversal at the leaves of the composition tree. The approach uses

a signature-based matching criterion. The Pounamu approaches presented in [v] describes a generic approach for diff and merge via a set of plug-in components. Plug-ins is developed for the meta-CASE tool Pounamu which support version control, visual differencing and merging. The approach uses operation-based method for difference computation which results in the dependency of the tool in which diagrams are edited, contrary to our approach which uses State-based approach. The approach uses a universal ID (uid)-based matching criteria. Also the approach lacks detection of the shifts operation.

Existing approaches in the area of model-based VMS usually deal with only one specific kind of model e.g. workflow [viii] or class diagram [ix, vi], in contrast, our approach is generic and not dependent on any specific model. The approach presented in [vii] performs diff/merge on structured data, i.e., XMI. As stated earlier such representation is not suitable for these activities, we use graph structure for diff/merge operations to avoid problems of textual representation. The approach presented in [iv, iii], is based on operation-based conflict detection & resolution. All edit operations that are executed on the diagrams are logged by the editor tool and used for conflict resolution, thus the approach is dependent on editor tool. We presented a state-based approach which is independent of editor tools since logging of edit operations is not required. To the best of our knowledge the only approach that addresses the issue of evolution control is given in [ii], however the evolution control is based on the attributes' properties while in our approach it is based on interlink information. Furthermore, the existing approaches, with the exception of [v], don't reuse the traditional VMS tools, while we argue that existing VMS tools should be reused for versioning purpose.

III. MODEL-BASED VMS FRAMEWORK

Keeping the issues of file-based VMS systems this paper provides a generic model-based VMS framework, which aims to overcome the challenges faced by existing systems when dealing with models as central artifact. Following are the components of our proposed framework:

- a. Model development
- b. Model transformation
- c. Model configuration
 - i. Model diff
 - ii. Model merge
 - iii. Evolution control
- d. Version management
- e. Graph structure DSL

Below is the description of these components.

A. Model Development

The Model Development module deals with

modeling issues, such as the development of source models conforming to source DSL using a model editor. As a source models we are using MOF-compliant domain specific languages (DSLs), such as unified modeling language (UML), ECore. A source model conforming to source DSL is transformed into target model conforming to target DSL in Model Transformation module. A developer can load a source model from the repository rather than developing a new one.

B. Model Transformation

The Model Transformation module deals with model-to-model transformation. At a fine-grained level we represent our models as graph structures. In Model Transformation module the configuration manager first establishes mappings between the source and target DSLs.

The mappings are the transformation rules defined for the sake of transforming a source model into a target model. Transformation between a source model into the target model is based on the mapping rules defined by the configuration manager.

C. Model Configuration

The Model Configuration module deals with Model Diff, Model Merge, and Evolution Control Mechanisms. Below we give a brief description of these tasks.

1) Model Diff

The Model Diff deals with model differencing. Model differencing is the process of comparing two models for the purpose of identifying mapping (similarity) and differences between them. It is an essential activity in many model development and management practices [ix]. For example, model differentiation is needed in a model versioning system to trace the changes between different model versions to understand the evolution history of the models. Model comparison techniques and tools may help to maintain consistency between different views of a modeled system. Furthermore, model differentiation can also be applied to assist in testing the correctness of model transformations by comparing the expected model and the resulting model after applying a transformation rule set.

When comparing two models, model mappings define those entities that represent a single conceptual entity in the compared models, while the unmatched entities represent model differences [ix].

2) Model Merge

The Model Merge deals with model merging. Model merging denotes the process of combining n alternative versions a_1, \dots, a_n into a consolidated version a , usually, $n = 2$. Different variants developed more or less independently from some common

ancestor are sometimes needed to be combined into one common version. The merge process consists of the following three main steps:

Versions comparison: The process of comparing derived versions with the base version.

Conflict detection and resolution: The process of identifying the conflicted elements and resolving the conflicts either manually or automatically.

Merging: The process of combining two or more versions into a consolidated version.

The comparison process of versions are described in the previous subsection Model Diff. We reuse the results of Model Diff in merge activities. We identified different merge cases in order to differentiate the conflicted and non-conflicting cases. Based on merge cases we establish our merge policy. The result of diff comparison will be analyzed according to the merge policy and possible actions are categorized into add, delete, include changed etc. The desired action then will be performed. The process of merging cannot be completely automated [xix]. Manual interaction is required in case of conflict detection. A conflict usually occurs if the same element of a model is modified in parallel. In case of conflict the conflicted elements will be identified. A manual interaction will be required to resolve the conflict. Finally the merge operation will be performed and the merge diagram will be obtained.

3) Evolution Control

The goal of MDE is to perform Software Engineering (SE) activities only on models, however, in reality models and files coexist and will have to be managed together in a consistent way. As identified in [ii, xii] this situation requires the definition of new evolution paradigms for software projects that consist of a mixture of models and files. The Evolution Control deals with defining a policy for creating a new version, defining version granularity, defining intralink and interlink information. The assumption is that software development consists of a set of different kinds of models and the interlink information between these models. Such models include analysis and design models, test models, and implementation models. These models may possibly be created using different development tools in a heterogeneous environment. For traceability & synchronization between these models one needs to identify intra & interlink dependencies between different model elements. In our framework, we first define the concepts of intra & interlink dependencies between model elements. Based on intra & interlink information we define the concept of evolution control policy. In this regard this module addresses the issue of traceability and evolution control mechanisms in model-based VMS systems.

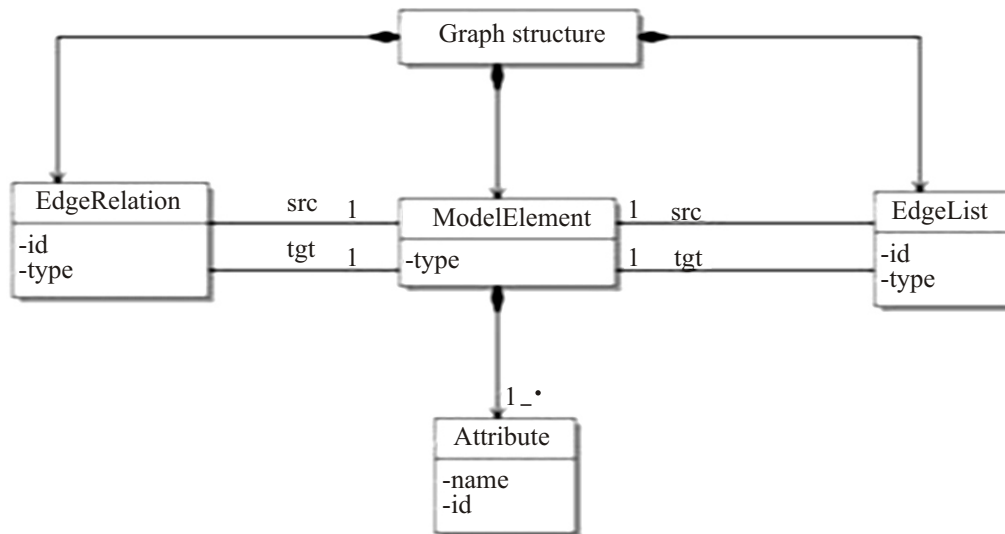


Fig. 2. DSL of Graph structure

D. Version Management

In traditional VMS approaches versioning, diff, and merge activities are performed on structured data such as XMI. In versioning, a version model defines the items to be versioned, the common properties shared by all versions of an item, and the deltas. It defines whether a version is characterized in terms of the state it presents or in terms of some changes relative to some

baseline. It selects a suitable representation for the version set (e.g. version graph), and it also provides operations for retrieving old versions and constructing new versions. Each version has relationship to the next and previous versions. Versions are queried or created by transactions. These include both read-only and read-write transactions, such as update, history, check in, checkout etc made by the user. Although traditional

VMS tools do not provide good support for model diff, merge, and evolution control activities however they do so for versioning. Therefore, in our approach we reused them for versioning purposes.

E. Graph Structure DSL

An abstract view of the proposed framework is given in figure 1 which depict the relation of input models and proposed framework. The inputs of the framework are graphical models such as UML models. We call the input model as source domain specific language (DSL). The source models are transformed into target models which are the instances of graph structure which is our target DSL. Below we describe our graph structure DSL.

A model can be represented in three different ways [vi], a) the graphical representation, i.e., the diagram itself, b) the persistence representation e.g. XMI, and c) intermediate representation e.g. syntax tree or graph structure. The graphical representation is the coarse-grained while the other two are fine-grained representations. In our approach, at a fine-grained level we represent models in an intermediate representation, i.e., graph structures. A metamodel of graph structure is given in Fig. 2. It consists of Model Element, Edge Relation, EdgeList and Attribute. Model Element represents the set of entities in the model, Edge Relation represents the relationships or associations within the model, EdgeList is a relationship used to connect all entities in the graph structure. Finally, Attribute represents all the possible features of model elements. With this metamodel, we can represent any kind of model at fine-grained level.

IV. REFERENCE ARCHITECTURE

Fig. 3 shows the reference architecture of our

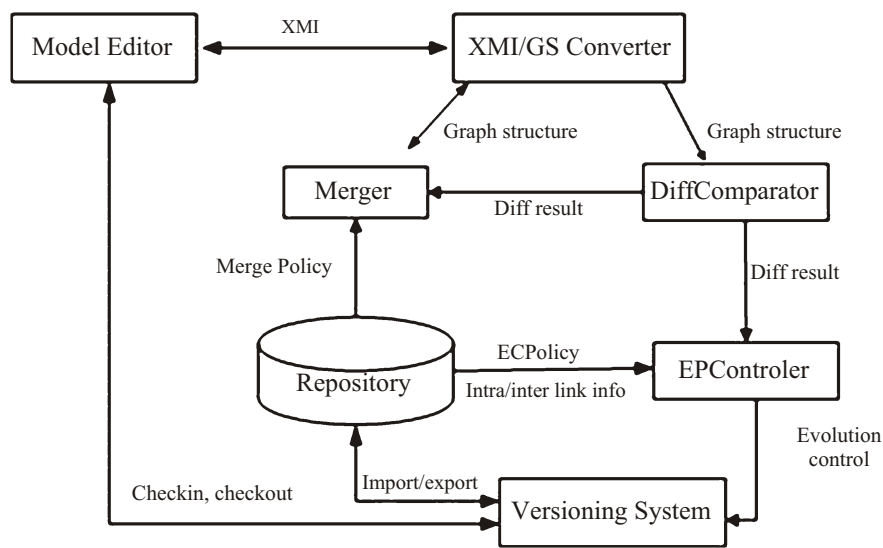


Fig. 3. Reference Architecture

proposed framework. There are six main components and a repository. The components are Model Editor, XMI/GS Converter, Merger, Diff Comparator, EPControler and Versioning System. XMI/GS transforms them into graph structures and vice versa. The graph structures of different versions are inputs to the DiffComparator. The Diff Comparator performs model diff by comparing the graph structures and produces diff result in the form of matched and unmatched elements. The output of the DiffComparator, i.e., The diff result is input to both the Merger and EPControler component. Merger analyzes diff result based on the merge policy and performs a three-way merge. The EPControler manages the evolution control based on diff result and intra & interlink information. Finally, the Versioning System is the reusable component of existing VMS systems and responsible for managing versions.

V. CONCLUSION

This paper presents a generic framework for model diff, merge and evolution control activities in model-based VMS systems. Graph structure can be used to represent any kind of model either domain specific or UML models. The presented framework is generic in a sense that it is neither dependent on any specific tool nor on any specific model type. Furthermore, these existing approaches do not consider reusability of existing file-based VMS systems and in most of the cases evolution control mechanisms are also missing. In this work at conceptual level, we proposed a model-based VMS framework that can be used to developed model-based VMS systems. As a future work, the implementation and evaluation of the architectural components, i.e., XMI/GS converter, Merger and EPControler will be performed.

REFERENCES

[i] Conradi and Westfechtel. "Version Models for Software Configuration Management". *CSURV: Computing Surveys*, 30, 1998.

[ii] J. Estublier, T. Leveque and G. Vega. "Evolution control in MDE projects: Controlling model and code co-evolution". *In Published at FSEN Int. Conf. On Fundamentals of Software Engineering Theory and Practice*, 2009.

[iii] M. Koegel, J. Helming and S. Seyboth. "Operation-based conflict detection and resolution". *In CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, 2009.

[iv] M. Kogel, "TIME - Tracking Intra- and Inter-Model Evolution". *In Software Engineering (Workshops)*, 2008.

[v] A. Mehra, J. Grundy and J. Hosking. "A generic approach to supporting diagram differencing and merging for collaborative design". *In ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005.

[vi] U. Ohst, D., M. Welle and U. Kelter. "Merging UML Documents". *Technical report, Universitat Siegen*, 2004.

[vii] H. Oliveira, L. Murta and C. Werner. "Odyssey-VCS: a flexible version control system for UML model elements". *In SCM '05: Proceedings of the 12th international workshop on Software configuration management*, 2005.

[viii] E. Ogasawara, P. Rangel, L. Murta, C. Werner and Marta Mattoso. "Comparison and versioning of scientific workflows". *In CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, 2009.

[ix] D. Ohst, M. Welle and U. Kelter. "Differences between versions of UML diagrams". *In ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, 2003.

[x] M. Pilato, *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[xi] E. H. Berso, V. D. Henderson and S. G. Siegel. "Software configuration management". *SIGSOFT Softw. Eng. Notes*, 3(5):9{17, 1978. ISSN 0163-5948. doi: <http://doi.acm.org/10.1145/953579.811093>.



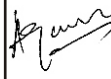
[xii] J. Estublier, T. Leveque and G. Vega. "Defining and supporting evolution strategies for model driven software projects". *LIG-IMAG*, 220, rue de la Chimie BP53, 38041 Grenoble Cedex 9, France.

[xiii] M. Alanen, I. Porres, "Difference and union of models," *In Proceedings of the UML Conference, Springer-Verlag LNCS 2863, San Francisco, California*, pages 217, Oct. 2003.

[xiv] Eclipse foundation, "emf compare," 2008, <http://www.eclipse.org/modeling/emft/?project=compare#compare>.

[xv] Eleni Xing, Zhenchang, Stroulia, "UmlDiff: An algorithm for object-oriented design differencing," *In Proc, IEEE/ACM International Conference on Automated Software Engineering (ASE'05), Nov. 2005, Long Beach, California, USA, ACM*, pp 54-65.

[xvi] U. Kelter, J. Wehrenand J. Niere, "A generic difference algorithm for uml models," *In Peter Liggesmeyer, Klaus Pohl, Michael Goedicke, editors, Software Engineering, volume 64 of LNI*, pp 105-116, GI, 2005, ISBN 3-88579-393-8.

Authorship and Contribution Declaration			
	Author-s Full Name	Contribution to Paper	
1	Dr. Waqar Mehmood (Main/principal Author)	Proposed topic, basic study Design, methodology and manuscript writing	
2	Mr. Arshad Ali (2nd Author)	Literature review	
3	Dr. Abdul Qayyum (3rd Author)	Data Collection, statistical analysis	
4	M. Ejaz Qureshi (4th Author)	Referencing and quality insurer	