MODEL VERSIONING IN CONTEXT OF LIVING MODELS

Waqar Mehmood and Arshad Ali

Department of Computer Science, COMSATS Institute of Information Technology, Wah Campus.

ABSTRACT

In this paper we present an approach of Model Versioning and Model Repository in context of Living Models view. The idea of Living Models is a step forward from Model Based Software Development (MBSD) in a sense that there is tight coupling between various artifacts of software development process. These artifacts include System Models, Test Models, Executable artifacts etc. We explore the issues of storage (import/export) of model elements into repository, inputs of cross link information, version management and system analysis. The modeling environment in which these issues will be discussed is a heterogeneous modeling environment, where different models types and different modeling tools are used in the development process. An overview of the tool architecture is also presented..

KEYWORDS

Model-based software development, living models, model repository

1. INTRODUCTION

Software Configuration Management (SCM) is an indispensable part of high-quality software development life cycle. It deals with managing the evolution of software system. The main activity in SCM is versioning of the software artifacts. Modern software development techniques such as MDA and MDSD deal models as first class artifacts. To ensure the quality of model and model transformation they must by designed, analyzed, maintained and subject to configuration management. There exist SCM systems (such as SVN [5], CVS [6] etc) for the later phases of software development, notably during implementation where the main artifact is source code in the form of text files. However these systems are not well suited for performing versioning task for models because models are not text file[7,8,9]. Models are represented by set of diagrams which has logical structure such as UML diagrams. Representing models with structured files like XMI or XML documents and performing versioning tasks on tree-based representation is an inadequate solution since it requires operation on models at a low level of abstraction. Models are structurally represented as graph and often rendered in a graphical notation therefore an alternate solution to represent models as graph structures at fine grained level rather than text files. Thus automated differentiation algorithms and supporting tools are required for the graphical structure of models. This problem is handled in our previous work [4].

Majority of the existing approaches in model comparison deal with single modeling language i.e. UML and focuses on detecting and reporting differences between the consecutive versions of UML class diagrams [10]. Nevertheless, in the context of MBSD and with the advent of approaches based on Domain Specific Languages (DSL), it is essential that models of different

DOI : 10.5121/ijsea.2015.6404

languages and technologies should be compared as well i.e Modeling in a heterogeneous environment. The task of version management becomes even more complex when we are working in a heterogeneous environment. In a heterogeneous environment we not only have different types of models but also have different types of modeling tools. For instance, at modeling level we may have System Analysis and Design models in the form of UML use case, class and collaboration diagrams as well as Test Models in the form of test cases, test report and test data etc. Similarly we may have Analysis and Design tools for analysis and design models and a testing tool for testing models. The output of working in such an environment is that we get a set of interrelated models. These interrelated models shares a common set of interrelated model elements. These models elements may belong to a common metamodel or may belong to different metamodels. To perform version management in such a modeling scenario one needs a centralized model repository as a first requirement.

Keeping in mind the difference between textual vs graphical representations on one hand and homogeneous vs heterogeneous environment on other, there is a need for new approaches to model repositories and version management which not only provide a suitable solution to the above stated problems but is also be tool-independent. In this paper we present an approach of Model Versioning and Model Repository in context of Living Models view. The idea of Living Models is a step forward from Model Based Software Development (MBSD) in a sense that there is tight coupling between various artifacts of software development process. These artifacts include System Models, Test Models, Executable artifacts etc. The remainder of this paper discuss these issues in more detail. We explore the issues of storage (import/export) of model elements into repository, inputs of cross link information, version management and system analysis. The modeling environment in which these issues will be discussed is a heterogeneous modeling environment, where different models types and different modeling tools are used in the development process.

The organization of paper is as follows. Next section describe model repository and cross-link information. In section 3 we define previous work on version management. Section 4 described the proposed solution to cross link or Mapping information and System Analysis. Finally a short conclusion and future work are presented in last section.

2. MODEL REPOSITORY AND CROSS-LINK INFORMATION

Central to all configuration management activities are a centralized Model Repository. A centralized Model Repository provides a facility to store and access the models independently from the modeling tools used to present and manipulate them. In a centralized repository one has two kinds of links between model elements i.e. inter and intra models links [3]. Intra-model links connect model elements within one model, such as a use case model. In a use case model a link from a use case to a participating actor is an intra-model link. Inter-model links connect model elements of different models. A link from a use case in the use case model to an open issue in the issue model is an inter-model link. Integrating different tools also implies integrating their models. Intra-model links are already part of the models, while inter-model links are added during integration. Only by adding inter-model links models from different tools can be set into relation and additional value is generated compared with having separate models.

Inter-model links may be added for numerous reasons but a major motivation for adding them is traceability. Only by adding inter-model links elements from the previously isolated models can be traced to model elements from different models. As intra-model links are defined in the context of the respective tool they can be supported by the tools' SCM capabilities if there are any. However this will create a media break when viewing models from different tools. In

contrast the inter-model links are unknown to the tools, subject to integration, and are not supported by their SCM capabilities. So in summary for supporting the management of change in an integrated environment we need an integrated approach for SCM that is not limited to a single tool and intra-model links.

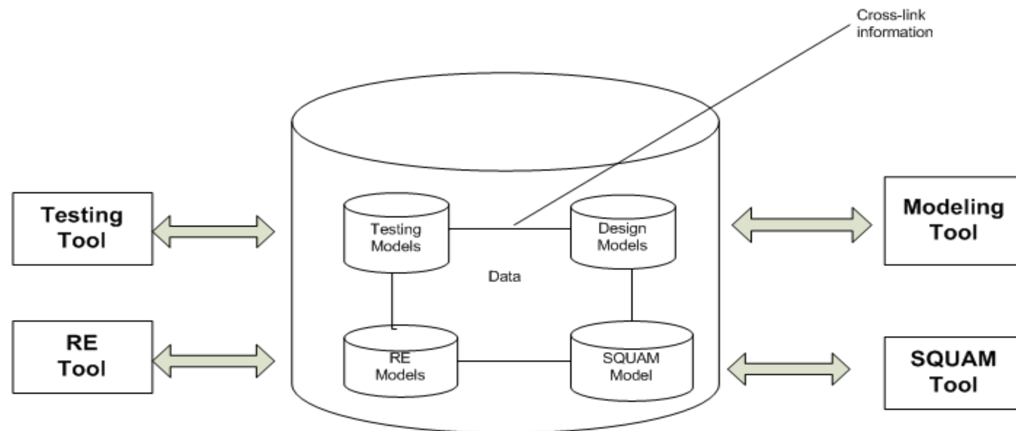

Figure 1. Model Repository

3 MAINTAINING CROSS-LINK INFORMATION

In this section we will describe how to maintain the cross-link or mapping information between different models in repository. Mapping information deals with inter-model links. It can be achieved by defining a Mapping function M between different model types.

Mapping Function A Mapping function M:S⟶T, where S is the source model and T is the target model. M(s) = t, for a source model entity s 2 S, t 2 T represent all the target model entities which are linked with s. Consider the scenario of Figure 3 where, S represent System Model and T represent Test Model and the element 'A' of System Model has a mapping relationship with elements 'X', 'Z' of Test Model. Then mapping function M(A) = X,Z represent
'A' in System Model relationship with the 'X', 'Z' in Test Model. By this way we can construct cross links between different models and store it into an XMI file. In [1] XMI _les is used for storing mapping information between two version of domain specific language. Table 1.0 shows example XMI file.

```
<mapping>
<domain class>
<classmap sourceclass =a targetclass =x>
....
</domain class>
<properties>
<propertymap sourceclass=a propertyname =name
targetclass=x propertyname=id>
...
</properties>
<relationship>
<relationmap sourceclass = c relationname =dependency
targetclass=y>
...
</relationship>
</mapping>
```

Table 1.0 Mapping information

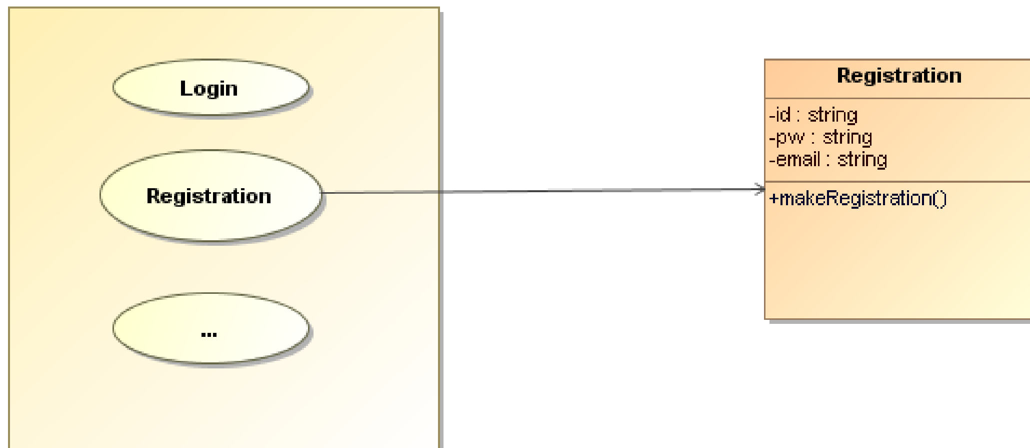

Figure 2. Use case to Class Mapping

While maintaining such mapping information we have to consider the possibility of automation or tool support. There are two kinds of mappings:

1. Mapping between same types of models e.g. a class diagram at model level and at implementation level.

2. Mapping between different types of models such as between use case model and class diagram or a class diagram and a test case model etc.

In former the automation is possible due to same types of model elements while in later automation becomes complicated and one rely on manual effort. The next step after maintaining the mapping information is performing System Analysis. For different types of models which share some common element we can perform System Analysis. System Analysis can be done based on

1. Difference Detection in model versions
2. Cross link information

For instance, consider a Use Case Registration and a class diagram Registration given in Figure 3. The use case Registration has information about participating actors, inputs to the use case and output. The class Registration has a set of attributes. It can be seen that inputs of the use case and attributes of the class are same information. This provides cross-link or mapping information between two entities. Now combining this mapping information with the difference report (obtained by model comparison) one can perform system analysis. Difference report gives information that Registration use case is modified in its second version, while from mapping information we know that Registration use case is linked with Registration class. Since Registration use case is modified we need to consider Registration class for possible modification.

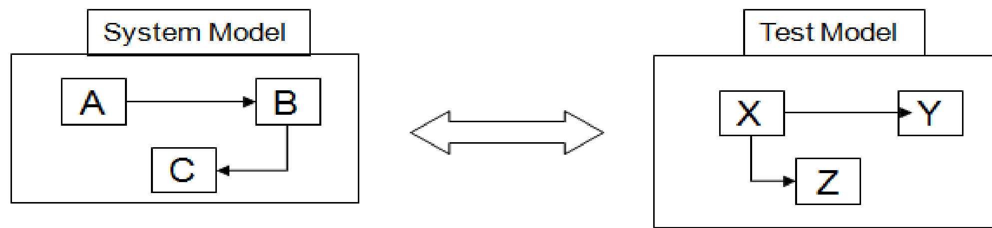

Figure 3. System and Test Model

4 RELATED WORK ON MODEL VERSIONING

Previously we presented a tool-independent approach for difference detection between versions of domain specific models as well as UML models. The basic architecture of our approach is given in Figure 4. For a given domain specific metamodel, an instance (data) model and its revised version will be created. These instance models will then be transformed into graph structure at fine-grained level. Then a comparison algorithm will be applied on them to detect
the mapping and differences. We address the problem of computing the mapping and differences between models by exploring the following issues:

1. What properties of models need to be compared?
2. How to represent instance models as graph structure at fine-grained level.
3. What algorithms can be used to discover the mappings and differences between models?

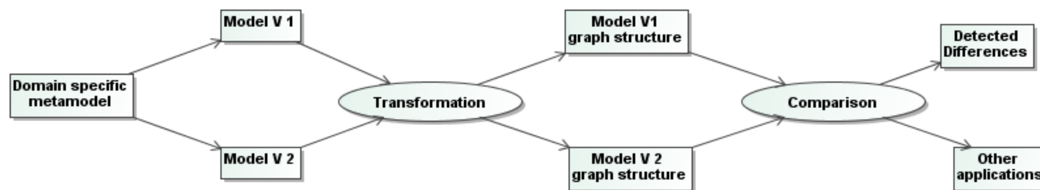

Figure 4. Basic Architecture

4.1 MODEL COMPARISON CRITERIA

Model comparison is the process of comparing two models for the purpose of identifying Mapping and Differences between them [8]. It is an essential activity in many model development and management practices such as Model-centric version control, Model Consistency, Model Merging, Transformation Testing etc. What do we mean by Model Mappings and Differences and what is the basis for this is the first question needed to be answered. When comparing two models Model Mappings define those model entities that represent a single conceptual entity in the compared model, while the unmatched entities represent model differences.

The basis for identifying Mapping and Differences is called correspondence criteria. Correspondence Criteria is a metric which defines what information needs to be considered for mapping and differences when comparing two models. Different approaches of model comparison define different correspondence criteria. They can be categorized into:

- Persistent Identifier
- Similarity based Matching

PERSISTENT IDENTIFIER

The assumption is that each element has a universally unique identifier (UUID), which is assigned to newly created element by the model repository. Model comparison is performed based on these persistent identifiers. The problem with such a criteria is that it can only be applied to two models that are subsequent versions and created in a same development environment. Such a solution can't be applied when two models are not subsequent versions or created by different development tools.

SIMILARITY BASED MATCHING

The other criteria for model comparison is based on similarity of the syntactical information of the compared elements. This syntactical information mainly includes Name, Type and Attributes. Different approaches use a variant of this criterion according to their requirement. The idea is that a pair of corresponding model elements need to share a set of properties which can be a subset of their syntactical information. It may also include the context or structure similarity, in which the structure of model entity is also considered. The structure includes the number of edges connected to an element and the end elements of a relationship.

In our approach a variant of the similarity based criteria is used. For syntactic information matching between two elements we use Node signature match, where node signature consist of Type-Name-Attributes information. Type is a correspondence metamodeling element, Name is the identifier or domain specific role of the entity and Attributes are predefined by the metamodel. For structural properties we use the comparison of connected elements in the graph structures of corresponding element.

4.2 GRAPH REPRESENTATION OF DATA MODELS

After defining the metamodel and its instance models, the next question need to be addressed is: At which level to compare the models. Normally a model can be stored at fine-grained level by structure data like XMI and XML but this kind of representation is not well suited by model differentiation purposes. An alternate storage is a graph structure. Our approach is based on graph representation of the model, thus applying the graph algorithms to model comparison.

4.3 COMPARISON ALGORITHM

In order to compare two models their syntactic and structural properties need to be compared. If there exist more than two elements of the same syntactical properties then structural properties are compared, which includes relationships between entities. Model comparison algorithm results in two sets Mapping Set (MS) and Difference Set (DS). The Mapping Set contains all pairs of model elements that are mapped to each other between two models. The Difference Set contains all detected discrepancies between the two models. The Difference Set contains New elements (N), Deleted elements (D) and Changed elements(C) i.e DS= {N,D,C}. Our algorithm differentiate between add/delete element and shift operation.

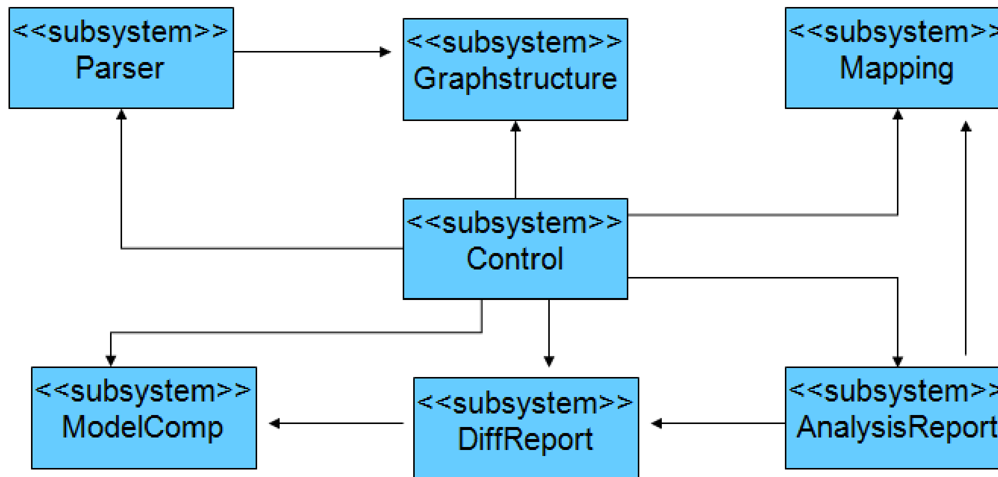

Figure 5. Tool Architecture

5 TOOL ARCHITECTURE

Our tool architecture is given in Figure 5.0. The tool reads two versions of UML model and produces Difference Report. Difference report is then combined with mapping information for generating system analysis report. There are seven main packages in the system, namely: parser, Control, Graphstructure, Mapping,ModelComp, Di_Report and AnalysisReport. The Parser subsystem parses the XMI files that describe the UML models. The parsed models are then converted into graph structures by Graphstructure subsystem. The control subsystem is responsible for overall control flow of the application. The ModelComp subsystem is responsible comparing the two versions of the model stored in the graph structures. The Diff Report subsystem generates the report of differences between two versions. Mapping subsystem is responsible for maintaining the mapping information. AnalysisReport subsystem generates the analysis report based on difference report and mapping information.

7 CONCLUSION AND FUTURE WORK

In this paper we present an approach of Model Versioning and Model Repository in context of Living Models view. The idea of Living Models is a step forward from Model Based Software Development (MBSD) in a sense that there is tight coupling between various artifacts of software development process. These artifacts includes System Models, Test Models, Executable artifacts etc. We explore the issues of storage (import/export) of model elements into repository, inputs of cross link information, version management and system analysis. The modeling environment in which these issues will be discussed is a heterogeneous modeling
environment, where different models types and different modeling tools are used in the development process. We showed how mapping information can be maintained and how it will used for system analysis purposes. An overview of the previous work on Model versioning and tool architecture is also presented. As a future work we will extend our work on the issues explored. We will focus on the implementation of the concepts provided.

REFERENCES

[1] Gerardo de Geest et al. Building a framework to support domain-specific language evolution using microsoft dsl tools.

[2] Frank Innerhofer{Oberperer Bernd Tilg Joanna Chimiak Opoka, Gunnar Giesinger. Tool{supported systematic model assessment. 2005.

[3] Maximilian K ogel. Time - tracking intra- and inter-model evolution. In Softwarev Engineering (Workshops), pages 157-164, 2008.

[4] Waqar Mehmood. Model versioning. In Master Seminar , University of Innsbruck, 2008.

[5] Michael Pilato, "Version Control With Subversion," O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004, ISBN 0596004486.

[6] CVS project, cvs web site, http://www.nongnu.org/cvs.

[7] D.Ohst, "A fine-grained version and configuration model in analysis and design," In ICSM'02, Proceedings of the International Conference on Software Maintenance, Washington, DC, USA, 2002, IEEE Computer Society, page 521, ISBN 0-7695-1819-2.

[8] Dirk Ohst, Michael Welle, Udo 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, , New York, NY, USA, 2003, ACM, pages 227–236, ISBN 1-58113-743-5, doi:http://doi.acm.org/10.1145/940071.940102.

[9] U.Ohst, D., M.Welle, Kelter, "Merging uml documents," Technical report, University Siegen, 2004.

[10] M.Girschick, "Difference detection and visualization in uml class diagrams," In Technical University of Darmstadt, Technical Report TUD- CS-2006-5, pages 37–51, 2006.

AUTHORS

Waqar Mehmood is Assistant Professor at the Department of Computer Science in COMSATS Institute of Information Technology, Pakistan. He received his PhD in computer science from the University of Innsbruck, Austria, in 2011. His prime research area is model-driven development focusing model-based software configuration management, model transformation, model versioning, graph transformation, domain-specific languages and model evolution.