MetaDiff - a Model Comparison Framework

Mark Kofman and Erik Perjons

Department of Computer and System Sciences
Stockholm University and Royal Institute of Technology
Stockholm, Sweden
{emis-mak,perjons}@dsv.su.se

Abstract. The increasing importance of models in software development raises a number of new concerns and challenges. This paper has concentrated on the issue of model comparison in the context of model driven development. The goal of the paper is to describe the requirements for, design, and implementation of a model comparison framework called MetaDiff. The framework is based on available implementations of Meta Object Facility (MOF) standard. The authors intend this framework to be useful to the following experimental research in the area of model management and when implementing specific model comparison and merge tools and algorithms.

1 Introduction

There are an increasing research interest and industry support for model driven approaches such as Model Driven Software Development [6], OMG Model Driven Architecture [20], Language Driven Development [7]. Number of open source projects such as Eclipse Generative Model Transformer (GMT) [12], Netbeans Metadata Repository (MDR) [18], Eclipse Modeling Framework (EMF) [8] and AndroMDA [3] and different CASE tool vendors work on implementation of different components for model driven development. The approaches are all based on the idea that modelling is a better foundation for developing and maintaining systems, then code centric development and maintenance [16]. With those approaches models are not side-effects of development anymore, but a central part of built software applications. However, the increasing importance of models raises also the number of concerns and new challenges. One category of problems is related to proper management of modelling artifacts.

If you take usual code centric development environment then you will find features like integrated version control, merge and comparison of version differences, and other features that make development in teams possible and convenient. However, most of those features are oriented to flat text files and are only suitable for conventional programming language based development. Transferring those features to model driven development tools is not straightforward and therefore a number of problems exist which yet have to be addressed by researchers and practitioners in this field.

According to [15], model comparison is key challenge towards best practices in model driven development. There are following aspects needed to be addressed while dealing with this issue:

- In some object oriented languages you split your system logically and physically into classes. Modelling languages, however, lack this standard way of physical decomposition. This often results in a huge amount of information stored in one model unit.
- Models are represented using different notations, often graphical. However, notation details easily sidetrack from identifying model logical differences.
- Models are not sequential text lines. Instead, models consist of entities that could be represented either as trees or graphs [15]. Therefore, other techniques for investigating differences must be used.

To deal with those aspects researchers and practitioners need to experiment extensively. For this reason they need a proper infrastructure solution. This should enable them to work on the aspect of their interest and see how other aspects influence or relate to each other. Furthermore, by bringing together input from different parties, this infrastructure could enable generation of new ideas and results.

The goal of the paper is to describe the requirements for, design, and implementation of a MOF-based model comparison framework prototype called MetaDiff. By "MOF-based" we mean that the models to be compared should all be based on the OMG's meta-meta-model standard, called Meta Object Facility (MOF).

The MetaDiff framework is an extensible comparison solution for modelling artifacts. Core of the framework is collection of interfaces or so called extension template. Main requirement for this extension template is to assure that the framework supports convenient way to add new implementations of matching and comparison algorithms. In addition, the framework should also provide practical way of working with different meta-models, i.e., different modelling languages. Main auditory for this framework are tool vendors and researchers in this field, who could use framework for the development of their specific components.

The framework is using open source Java implementations of MOF standard (Eclipse EMF [8] and NetBeans MDR [18]). Those implementations are able to load meta-model of MOF itself and enable user to instantiate it, that is creating their own meta-models. The implementations also provide a way to generate Java API to programmatically access meta-model level information.

This paper is organized as follows. In Section 2 we give introduction to concepts, ideas and related research results which are extensively used in this thesis work. In Section 3 we define requirements for the developed framework. Section 4 is concentrated on design and implementation. Application examples are given in Section 5. Finally conclusions are presented.

2 Concept and Related Work

2.1 Concepts and Basic Definitions

In this section basic concepts used by this thesis will be described shortly.

Model Driven Development. There are a number of different model driven approaches emerged during last years. Most well-known of them are Model Driven Architecture from OMG [20], Model Driven Software Development [6], and Software Factories [13]. All Model Driven Development approaches focus on model as primary artefact in the development process. Model transformations are considered primary operations on models, used to move information from one model to another. The idea of the software life cycle is being viewed as a chain of model transformations.

Models vs Diagrams. In this work similar view to OMG's Unified Modelling Language [22] is shared when defining diagrams. Diagram is considered to be a view or perspective on a full model or part of it. Usually diagrams are represented in graphical notation.

Meta-Object Facility. The Meta-Object Facility (MOF) [21] technology provides a model repository that can be used to specify and manipulate models. MOF is intended to be a a tool for designing and implementing new modelling languages.

Key modelling concepts in MOF are Classifier and Instance or Class and Object, and the ability to navigate from an instance to its classifier(meta-object). This key idea is sometimes used to organize modelling world into further levels of meta-layers. Each higher level meta-layer consists of classifiers for the lower level. The MOF standard gives the possibility to define as many layers as it is needed. However most typical is to limit yourself to four meta-layers.

Traditional architecture is based on four meta-layers. Those are :

- *M0*, which contains the data of the application (for example, the instances populating an object-oriented system at run time, or rows in relational database tables).
- *M1* contains the application: the classes of an object-oriented system, or the table definitions of a relational database. This is the level at which application modelling takes place (the type or model level).
- *M2* contains the meta-model that captures the language: for example, UML elements such as Class, Attribute, and Operation. This is the level at which tools operate (the meta-model or architectural level).
- *M3*, which is the meta-meta-model that describes the properties of all meta-models can exhibit. This is the level at which modelling languages are defined, providing interchange between tools.

2.2 Related Work

This paper focus is on model management. Similar issues as discussed here have already been addressed in the context of schema and database integration, e.g., [14], [23], [1]. Model management requires that operations can be applied on the models. Microsoft Research [5] has for several years been working on generic model management. They have developed a set of algebraic operators. Those operators are:

- *Match* - takes two models as input and returns a mapping between them. The mapping identifies relations (mapping) between the objects in the two input models. The relations are either equal or similar, based on some externally provided definition of equality or similarity.
- *Compose* - takes a mapping between models A and B and a mapping between models B and C, and returns a mapping between models A and C
- *Diff* - takes a model A and a mapping between A and some model B, and returns sub-model of A that does not participate in the mapping
- *ModelGen* - takes a model A and a mapping between A and B, and returns a new model B based on A and the mapping between A and B
- *Merge* - takes two models A and B and a mapping between them, and returns the union, called C, of A and B along with mappings between C and A, and C and B.

The implementation of operations presented above should be based on different algorithms. Several algorithms and solutions have been created for textual artefacts, but for model artefacts they are few. One such tree difference based comparison algorithm for MOF-based models [2] is used in this paper prototype implementation.

In [23] matching related algorithms are described. Distinctions are made between different schema matching approaches: schema-level and instance-level matches, element-level and structure-level matches and language-based and constrained-based matches. The goal of the framework described in this paper is to be able to manage all these different approaches.

When approaching the model difference problem from developers point of view it is important to consider layout information of model. Models are often represented using graphical notation such as UML diagrams. The importance to address visualization of diagram differences is stressed by [19]. The same paper suggests using colours as a solution to represent diagram differences. In addition to semantical differences authors identify modifications to the layout like shifting classes around in the diagram.

There are some CASE tool vendors that have come out with meta-model specific model comparison and merge solutions during late time. It is an important step forward; however it doesn't mean a complete solution to all model management problems.

3 Requirements Specification

This section describes the requirements for the MetaDiff framework, in form of use cases. The use-case model is a model of the system's intended functions and actors (roles) performing the functions. The use-case model is used as an essential input to activities in analysis and design. See Fig. 1 for UML Use Case diagram.

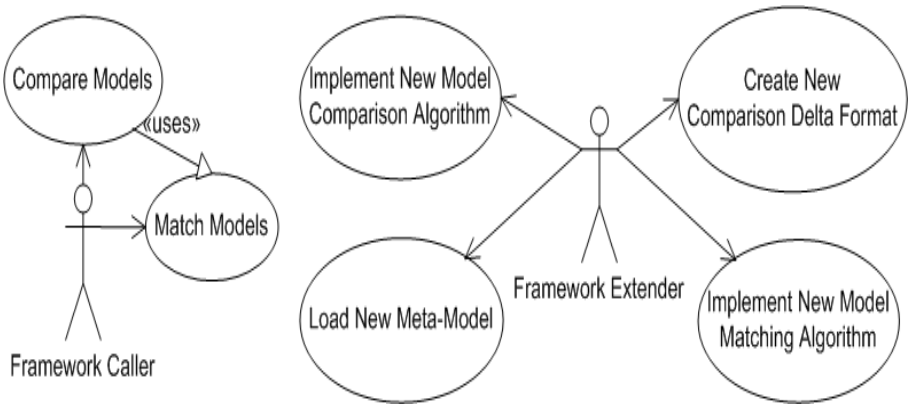

Fig. 1. MetaDiff Use Case Diagram

3.1 Actor Catalog

Actor catalog describes different actors related to framework, and gives brief description of actors' needs from the framework. See Fig. 2 for UML actors diagram.

The Framework Caller is interested in using basic framework functionality by calling its methods. Modeled system is a software framework so it does not provide any user interface. That means that *Framework Caller* is non-human actor who is using provided API. It is most likely software component or any other artefact being able to call framework methods. *The Framework Caller* should be able to work independently from specific implementation, e.g. changes made by *Framework Extender* should not affect significantly the way *Framework Caller* uses the framework.

Framework Extender is an actor who is interested in extending functionality of the framework and applying it in its own context. It is a human actor that most likely developing bigger system where model comparison framework could be used as a component.

The Tool Developer is a *Framework Extender*, who develops modelling tool which should provide model comparison functionality. *Tool Developer* usually has specific meta-model(s) in his hand. This actor's main purpose is to extend

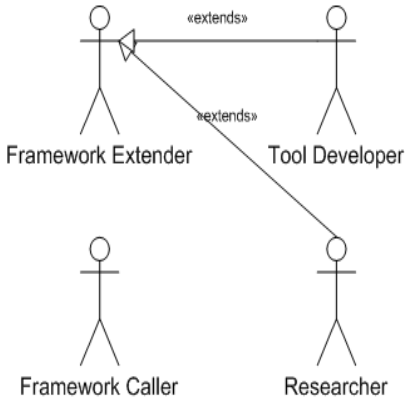

Fig. 2. Actors Diagram

framework functionality to make it suitable with modelling tool for later integration.

The Researcher is another type of *Framework Extender*. It is also a human actor whose purpose is to test new research ideas. This actor's main goal is to extend the framework for following study of related research problems by building different prototypes on top of the framework

3.2 Use Case : Match Models

Model matching is a function that takes two models as input and returns a mapping between those two models as output. Each mapping element of match result specifies that certain element of one model logically correspond to certain element of the other model [23].

The use case occurs when *Framework Caller* decides to find mapping between two models. First, *Framework Caller* specifies the algorithm to be used for model matching. Then he calls the model matching function. Framework executes chosen algorithm and returns resulting mapping. This mapping is likely to be used by following *Use Case : Compare Models* described in subsection 3.3. However, it could also be reasonable for *Framework Caller* to execute this use case independently.

Model matching could be a difficult problem in specific cases. In some cases it concerns semantic interpretation of models created by different human actors [14]. Therefore in some algorithms it is crucial to provide human/caller assistance. Framework design should be extensible for this type of matching algorithms.

3.3 Use Case : Compare Models

The use case enables *Framework Caller* to compare two models. First *Framework Caller* specifies the algorithm to be used for model comparison. Then he calls

the model comparison function. Optionally *Framework Caller* can also specify mapping between models, that is going to be used by comparison algorithm. As a result caller receives differences between two models based on the chosen algorithm execution. This result will further be referred as *comparison delta*.

In general case compared models should be instances of the same meta-level. However, in case of model transformation it could be useful to compare models that are instances of different meta-models. Framework itself should not have limitations on this feature. However, comparison algorithms can expose specific requirements on models and meta-models they accept.

Similar to matching algorithms, some comparison algorithms can rely on human/caller assistance. Framework design should be extensible for this type of comparison algorithms.

3.4 Use Case : Implement New Model Matching Algorithm

Framework Extender should be able to implement its own algorithms for model matching based on available meta-models. Algorithms could be meta-model specific and require a special type of models as an input. Algorithms can also be generic, in a sense that they use only MOF layer information.

3.5 Use Case : Implement New Model Comparison Algorithm

Framework Extender should be able to implement its own algorithms for model comparison. Those algorithms could be based on available meta-models, comparison delta formats and match algorithms if needed. Algorithms could be meta-model specific and require special type of models as an input. Algorithms can also be generic, in a sense that they use only MOF layer information.

3.6 Use Case : Implement New Comparison Delta Format

There is no strict way to represent model comparison result. Various comparison delta formats can have an influence on the way model comparison result is represented. Therefore, *Framework Extender* should be able to define its own comparison delta formats if needed.

3.7 Use Case : Load New Meta-Model

New meta-models should be possible to load by *Framework Extender*. This activity is specific to implementation of MOF standard used to load meta-model. It is important to be aware that definition of new meta-model can not be fully automated and requires human input in terms of code writing and integrating this code inside the framework.

3.8 Scenario: Create Model Comparison Tool

Actor *Tool Developer* could be interested in using MetaDiff framework to support development of model comparison tool. An example scenario describes activities the *Tool Developer* should carry out to succeed. Fig. 3 illustrates activity diagram for this scenario.

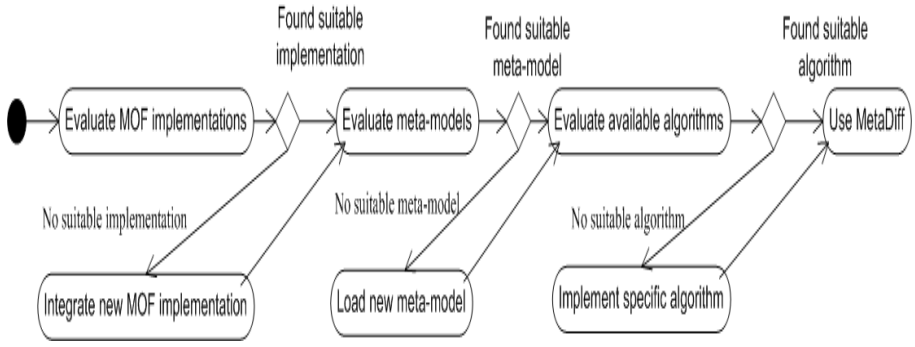

Fig. 3. Create Model Comparison Tool

The process consists of the following steps:

- Evaluate supported implementations of MOF standard. In general case it is the best choice to use one of already available implementations. But if none of them suits your purposes it is possible to integrate your own MOF implementation.
- Evaluate available meta-models. You should make sure that your model resource could be handled by framework. It means meta-model you are using should be loaded previously. You can also define new meta-model using one of available implementation of MOF standard.
- Evaluate available matching and comparison algorithms. It is likely you can use available algorithms or reuse parts of them. Otherwise create your own implementation.
- Now you can use a framework inside your model comparison tool.

4 Design and Implementation

In this section we presents design of the MetaDiff model comparison framework.

As it is shown on Fig. 4, on a higher level MetaDiff framework is split into three architectural parts:

- *Extension Template* - Collection of interface to assure different extensions of framework are possible

Fig. 4. MetaDiff Architecture

- *Infrastructure* - Integrated implementations of MOF standard and different meta-model definitions to enable loading of different model resources
- *Extensions* - Collection of generic and specific extensions distributed with a MetaDiff framework. Extensions are implemented on top of infrastructure provided and should implement *Extension Template*

4.1 Extension Template Design

Main requirement for Extension Template is to assure convenient way to add new match and comparison algorithm implementations and new comparison delta formats. Extension Template is implemented using a collection of interfaces and covers whole functionality presented in use case model. See Fig. 5 for class diagram of extension template.

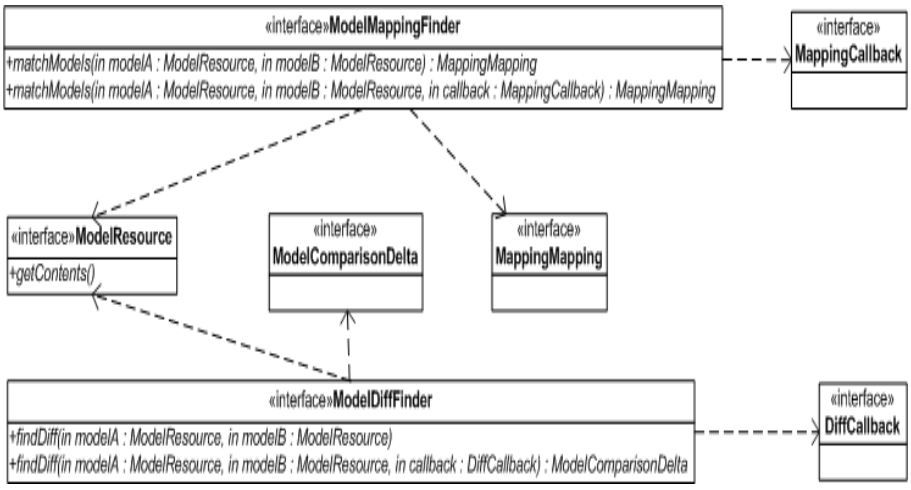

Fig. 5. Extension Template Class Diagram

4.2 Infrastructure

Current infrastructure is based on two Java implementations of MOF standard. Those are Eclipse EMF [8] and NetBeans MDR [18]. Both enable users to define

their meta-models and provide reflective API to access model data. Those APIs are used to implement matching and comparison algorithms. MetaDiff framework do not provide common model access interface yet. However, it is one of the future goals.

4.3 Extensions

MetaDiff framework comes with a collection of different extensions. In authors opinion most important are *TucsDiffFinder*, *DeltaXmlDiffFinder*, and *IdBasedMappingFinder*.

- *TucsDiffFinder* extends *ModelDiffDinfer* by implementing diff algorithm suggested in [2]. Implementation is based on Ecore meta-model.
- *DeltaXmlDiffFinder* is a comparison implementation based on commercial DeltaXML XML comparison tool and can compare models stored in XML.
- *IdBasedMappingFinder* is a *ModelMappingFinder* extension class. It uses object ID to make the matching between two models. This type of mapping is perfectly suitable for comparing different versions of one model.

We find that following extensions could be also beneficial :

- Meta-model specific algorithms, that use information provided in meta-model to improve efficiency or other quality attribute of the algorithm. Eg. UML comparison algorithms.
- Different extension of *ModelMapingFinder*. [23] gives a detailed classification of possible schema matching algorithms.
- *ModelComparisonDelta* extension as a set of elements to be added or removed from the model.
- *ModelComparisonDelta* extension as a model. Having delta represented as model can give user a possibility to apply model transformations also on comparison delta itself. It may require special treatment from comparison algorithm to make sure that delta is a well-formed model [5].

5 Examples

We have developed two application examples of MetaDiff framework. Complete source code of both examples is available at [17].

5.1 UML Model Matching Tool

Lets assume we have a task to develop command line model matching tool for UML 1.4 [22]. Specific of this matching tool is that it should not identify mapping automatically, but instead ask human to map each model element.

We follow Scenario: *Create Model Comparison Tool* but keeping in mind that we need just model matching functionality.

– There is UML 1.4 meta-model based on NetBeans MDR available for our use.
– We use *CallbackMatchingFnder* extension to provide human interaction. We also implement our specific implementation of *MappingCallback* interface. For each callback it will ask caller to choose one of elements to match.
– We implement *Framework Caller* using java class to access MetaDiff functionality.

5.2 Creating Diff Tool for Swallow Models

Lets assume we have a task to develop command line comparison tool for Swallow models. Swallow is a simple meta-model developed by Eclipse GMT open source project as an example of Model Driven Software Development [6] approach. This example is described in details in paper [9]. You can see EMF class diagram of this meta-model in Fig. 6.

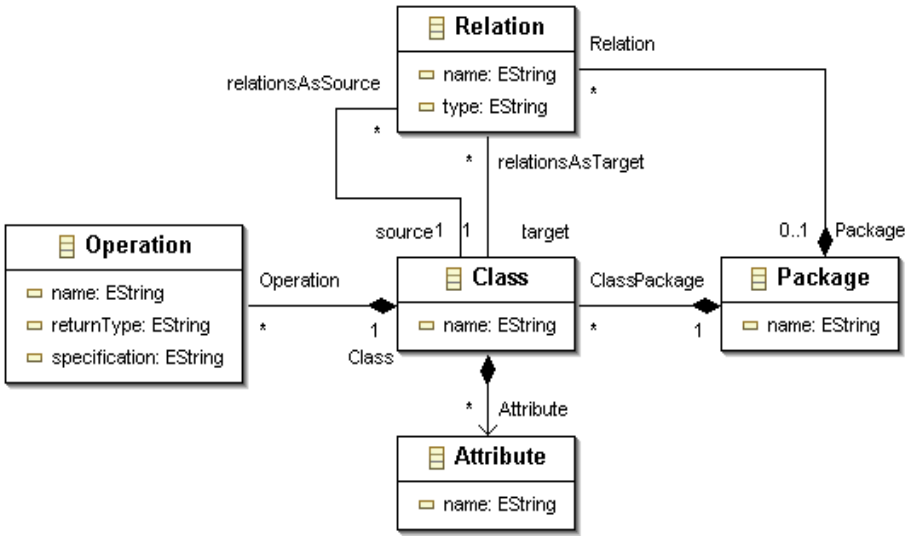

Fig. 6. Swallow Meta-Model

We follow Scenario: *Create Model Comparison Tool*.

– Swallow meta-model is defined using EMF Ecore and so is supported by current framework infrastructure,
– Now we load Swallow meta-model. We do it by adding needed jar files into the classpath,
– We decide to use algorithm implementation based on paper [2] provided with a framework, so we do not need to implement our own algorithm

– We implement *Framework Caller* using java class to access MetaDiff functionality. [17]

To test the tool we borrow simple guest book model described in [9]. Based on Swallow meta-model first version of Guest Book model was defined and then changes applied to that model. You can see tree representation of both models in Fig. 7. Following changes were done in the initial guest book model.

– Rename Package *guestbook* to *gb*
– Add new attribute to GuestBook Class
– Delete Attribute *text* from Class *GuestEntry*

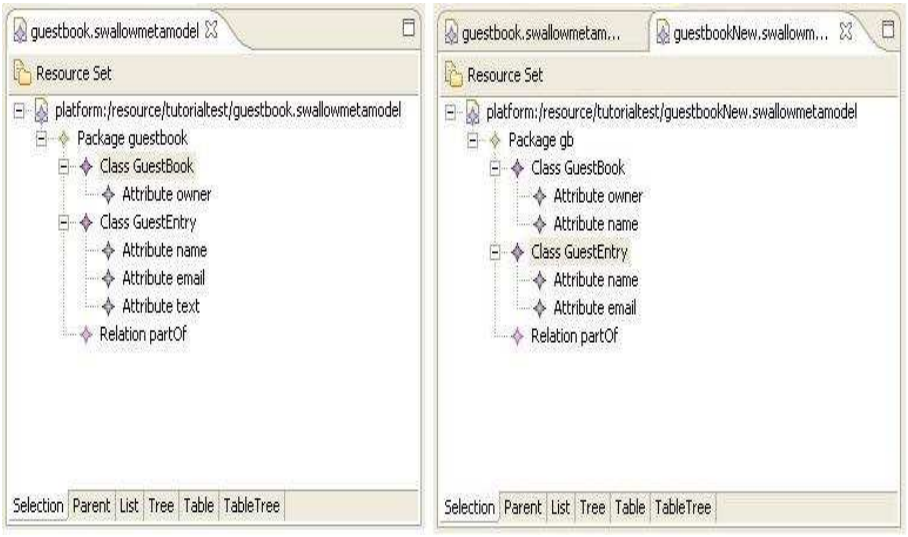

Fig. 7. Guest Book Model Versions

Running created tool has given a result of following diff operations:

– Create new element of type Attribute
– Set the value of feature *name* of element *Attribute* from value "null" to new value = "name"
– Set the value of feature *name* of *Attribute* with name "text" from value "text" to new value "null"
– Set the value of feature *name* of *Package* from value "guestbook" to new value "gb"
– Remove a link from element *Class* with name "GuestEntry" to feature with name "null"
– Insert a link from feature *Attribute* of element *Class* with name "GuestBook" to *Attribute* with name "name"
– Delete element *Attribute* with name "null"

6 Conclusion

6.1 Results Achieved

The central contribution is the development of MetaDiff - an extensible model comparison framework. The framework provides Extension Template - a collection of interfaces and guidelines to extend them, Infrastructure, which enables integration of MOF standard implementations and set of meta-models, and a set of extensions to serve as a proof of concept for Extension Template.

Benefits of current work are the following :

- Based on MetaDiff framework the following research is simplified by ability to create prototypes. Experimenting with framework extensions is considered as important improvement for researchers.
- MetaDiff *Extension Template* could be extended to provide wider range of model management functionality and so to deal with more general theoretical problems.
- Created framework has also a practical value. Developers of Domain Specific Languages could use this framework to provide model comparison solution for their domains. Simple example of how to do this has been shown for Eclipse GMT Swallow meta-model.

6.2 Future Work

We identify different streams of possible future work related to current thesis.

Clarify Extension Template. It is important not to stop on achieved results. Availability of different framework extension can lead to ideas for *Extension Template* improvement. Some generic abstract classes could be added as part of Extension Template to support development of extensions.

Create New Framework Extensions. Value of the framework will increase with the number of different extensions.

Here is the list of extensions we believe would be most important to implement next:

- *ModelMappingFinder* extensions not based on object identifiers. This would assure applicability of framework in wider range of situations,
- implementation of comparison algorithms for popular meta-models,
- create a well-structured hierarchy of *ModelComparisonDelta* extensions. This will improve applicability of the framework and help to work on visualization of comparison deltas,
- *ModelDiffFinder* and *ModelMatchFinder* extension that able to compare and match models based on different meta-models.

Include Other Model Management Functionality. If framework finds its users, then it is important to widen the functionality it provides. The authors believes framework functionality can move in direction of becoming implementation of full model management functionality.

Research on Team Development Issues. Here are some team development problems in context of model driven development, that the authors believe developed framework can help to address:

- The visualization of comparison deltas. This is important issue to make model comparison real in industry. Especially, considering graphical notation of industrial modeling languages. Clear and informative way to illustrate model differences is important goal. Research ideas could be tested on top of current framework by building the prototypes
- Work on scalability of the framework will assure model comparison solutions are feasible in real world large team environments and possible to apply to large models.

References

1. Alacig, S., Bernstein, P.A.: A Model Theory for Generic Schema Management, Proc DBPL 2001, Springer Verlag LNCS, 2001
2. Alanen, M. and Porres, I.: Difference and Union of Models. TUCS Turku Centre for Computer Science, Department of Computer Science, bo Akademi University (2003)
3. AndroMDA Code Generator Framework. http://www.andromda.org/
4. Bernstein, P.A., Shapiro, L.: Summary of NSF IDM Workshop Breakout Session NSF IDM Workshop (2003)
5. Bernstein, P.A.: Applying Model Management to Classical Meta Data Problems. Proceedings of the CIDR Conference (2003)
6. Bettin, J.: Introduction to Model-Driven Software Development. Business Proccess Trends, MDA Journal, Aprill 2004
7. Clark, T., Evans, A., Sammut, P., Willans, J.: Applied Metamodelling - A Foundation for Language Driven Development, Version 0.1. http://www.xactium.com/
8. Eclipse: Eclipse Modeling Framework. http://www.eclipse.org/emf
9. Emde Boas, G. van: Template Programming for Model-Driven Code Generation OOPSLA/GPCE: Best Practices for Model-Driven Software Development (2004)
10. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language, Second Edition. (1999)
11. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition. pp 79-83 (2003)
12. Eclipse: Generative Model Transformer. http://www.eclipse.org/gmt
13. Greenfield, J. and Short, K.: Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools. John Wiley & Sons, (2004)
14. Johannesson, P.: Schema Integration, Schema Translation, and Interoperability on Federated Information Systems. Doctoral Thesis, Department of Computer & System Sciences, Stockholm University, pp 1-25 (1993)
15. Lin, Y., Zhang, J., Grey, J.: Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development. OOPSLA/GPCE: Best Practices for Model-Driven Software Development (2004)
16. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA destilled. Principles of Model-Driven Architecture, Addisson-Wesley (2004)
17. MetaDiff Model Comparison Framework. http://www.dsv.su.se/ emismak/metadiff/index.html

18. NetBeans : Metadata Repository, http://mdr.netbeans.org/
19. Ohst, D., Welle, M., Kelter, U.: Differences between Versions of UML Diagrams. European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (2003)
20. OMG : Model Driven Architecture, http://www.omg.org/mda
21. OMG: Meta Object Facility (MOF) 2.0 Core Specification. (2003)
22. OMG: Unified Modeling Language, Version 1.4. (2003)
23. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema mathing. The VLDB Journal 10: pp 334350 (2001)