ABSTRACT

In model comparison one can distinguish three major subproblems: representation, calculation, and visualization of model differences. Existing stand-alone tools for managing model differences usually focus on one or sometimes two aspects of model comparison. In this paper we discuss our stand-alone tool called RCVDiff, that incorporates all three aspects of model comparison. Furthermore, we comment on some issues in the process of model comparison that are not easily perceived if the three aspects of model comparison are considered independently.

Keywords
Metamodeling, Model comparison, Model differences

1. INTRODUCTION

Model Driven Software Engineering (MDSE) is a field of Software Engineering which focuses on models as main design artifacts, and uses model transformations as means of relating models. Consequently, mature model configuration management systems are required to manage the complexity of modeled systems in MDSE environments. One of the major functions of model configuration management systems is model comparison. Model comparison (model differencing) is a complex process which consists of three concerns: representation, calculation, and processing of differences [9]. The rationale behind this separation of concerns is that usually it is not only required to calculate differences, but it is required to store, process, and visualize them in the context of a model configuration management system.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXX-XX-X/XX/XX ...$10.00.

Usually, research in the area of model comparison is focused on only one aspect of model comparison. For example, model differences representation mechanisms are discussed in [4, 10, 7]. Model differences calculation is discussed in [6, 8, 5]. Model differences visualization is discussed in [16]. Consequently, tools that are developed to validate the research of a certain aspect, focus only on that specific aspect. This inevitably creates the problem of integrating different tools into one system for model comparison. This problem becomes even worse because these tools may use different metametamodels to describe their metamodels, models and model differences. Thus, (error-prone) transformations must be defined to transform metamodels, models and difference models used in one tool to their counterparts used in other tools.

In this paper we discuss the details of our tool that combines all three aspects of model comparison [3]. Our tool is based on our research on model differences representation and calculation [14], and model differences visualization [15]. Furthermore, this tool is stand-alone and can be easily adapted to be used as a part of a custom configuration management system or a larger CASE tool.

The theoretical basis of our tool is discussed in Section 2. Thereafter, in Section 3, a description of a static and a dynamic architecture of our tool is given. Next, in Section 4, a use-case scenario is presented, which describes the usage of our tool. Finally, in Section 5, we conclude the paper.

2. PRELIMINARIES

In this section we first describe our approach to the representation of model differences. In our approach, the differences are represented by differences models which conform to a differences metamodel, similarly to approaches like EMFCompare [2] or the one presented by Cicchetti et. al. [4]. Next, we briefly describe our approach to the calculation of model differences, that produces differences models that conform to the specified differences metamodel. The details of both approaches can be found in [14]. Finally, we describe our approach to visualization of model differences. The details of this approach can be found in [15].

2.1 Representation of model differences

Our approach for the representation of model differences allows those differences to be seamlessly used in modeling environments. Thus, the differences between two models are represented by a difference model which conforms to a differences metamodel. The differences metamodel is based
on the metametamodel depicted in Figure 1.

This metametamodel describes both metamodels and models. Metamodels are obtained by instantiating the Meta-model element, and models are obtained by instantiating the Model element. This metametamodel can be considered as a domain specific metametamodel which is geared towards representation of model differences, and not towards general modeling (like MOF or Ecore). Thus, this metametamodel is comparable to the cores of MOF and Ecore.

The differences metametamodel is an extension of the introduced metamodel and is depicted in Figure 2.

The differences models are instances of the Differences-Meta-model element. The main building blocks of the differences models are instances of ChangedElement, DeletedElement, and AddedElement. Assuming that the differences model represents the differences between models A and B, then the instances of the AddedElement elements are that are in model B but not in model A, the instances of the DeletedElement elements are that are in model A but not in model B, and the instances of the ChangedElement elements are that represent the same entity in both models but are not structurally identical.

2.2 Calculation of model differences

Traditional approaches for the calculation of model differences are based on tree-matching algorithms. These algorithms match the nodes of two trees that represent two models being compared, and based on this matching the differences are calculated. Several types of matching are recognized: static-identity, signature-based, similarity based or differences are calculated. Several types of matching are recognized: static-identity, signature-based, similarity based or language-specific [9].

Our algorithm for calculating differences is also based on tree-matching algorithms. Unlike traditional approaches, that support only one type of matching, our algorithm is defined in such a way to support all four types of matching. In order to allow such a highly configurable calculation process, we extend our metamodel with additional elements. The extended metametamodel is interpreted as a calculation metamodel and is depicted in Figure 3.

Calculation models are used by our algorithm and they have two important features. The first feature is represented by instances of the CalculationConfiguration element. This feature of the calculation model opens the possibility of specifying the metamodel-specific configurations that are used to influence the calculation process of models related to the specific metamodel. Thus, for all models that conform to a specific metamodel, only one calculation configuration needs to be set. The second feature is represented by instances of the ComparisonMElement. These instances are used instead of model elements to represent nodes in model trees. An instance of a ComparisonMElement is generated for each model element in a preprocessing step, with the help of the metamodel-specific configuration.

2.2.1 Model comparison algorithm

The input for our comparison algorithm are two models A and B and the metamodel-specific configuration. Our comparison algorithm consists of three steps. In the first step, the trees representing models are traversed bottom-up, and the similarities between objects in models A and B are calculated. We say that two objects are similar if they can be considered as the same entity. We define similarities by using a similarity function which returns true if two objects are similar, and false otherwise.

In the second step, based on the similarities found, a matching of objects is calculated. The matching is performed by traversing the tree top-down. At the first level, based on the similarities found, some objects may be matched. For all matched objects at the first level, the matching process continues recursively until the bottom of the tree is reached or there are no sub-objects that can be matched.

In the last step, the calculation of differences is done based on the matchings found.

2.3 Visualization of model differences

In our approach to visualization of model differences, we adopt the idea of information visualization proposed by Shneiderman [13]: Overview first, zoom and filter, then details-on-demand. The reason for adopting this idea is based on the fact that model differences are “information content” that need to be visualized. Thus, it is required to have overview capabilities, such that the global meaning of differences can be comprehended. Next, it should be possible to zoom in and filter differences, such that the users of configuration management systems can syntactically and semantically associate the differences to the parts of the models that those differences are related to. Finally, the selected differences should be rendered by using a sufficient level of detail to help the users understand them better.
Traditional (e.g. text-based, tree-based or diagrammatic) approaches to visualization of model differences, if considered in separation, do not fully support the idea of information visualization [15]. Thus, in order to fully support this idea, we combine two visualization approaches.

The first approach we use are polymetric views, first described in [11]. The polymetric view is a lightweight graphical component for visualizing a set of related entities. This is accomplished by defining metrics that can be applied to the set of entities that is to be visualized, and by specifying a view. The view is specified by relating defined metrics to the graphical attributes of shapes that will represent entities. Thus, by calculating the metrics on elements of the set, and by inferring the values of graphical attributes of the shapes that will represent entities, the entities can be visualized on a graphical canvas. In the context of models, the metrics are based on metamodel elements. Consequently, they can be calculated for model elements conforming to those metamodel elements. This allows a visualization of model elements by using the calculated metrics.

The second approach we use is a framework for visualization of metamodel based languages (MMVisualizer in further text). MMVisualizer uses a declarative approach for the visualization of models. In order to visualize a model, the user needs to specify a set of rules. Each rule maps one metamodel element (of the metamodel that the model conforms to), to a graphical shape. Based on a type of the used rule, a predefined shape (e.g. rectangle, oval, line,...) is used to visualize model elements conforming to the mapped metamodel element. Although the rules are designed such that predefined positioning information can be used for actual positioning of model elements, this is not required per-se, but the layout of a model is calculated by using the dot [1] framework.

In order to use the two specified approaches the old model and the differences model are combined in one unified model. In the unified model the model differences are related to, and thus in a sense annotate, model elements. Thus, the metrics can be calculated on the unified model by also considering the model differences. Furthermore, by examining the unified model, model elements can be colored appropriately in MMVisualizer (deleted elements to the old model are colored red, added elements to the new model are colored green, and changed elements are colored blue).

3. TOOL ARCHITECTURE

This section gives an overview of the architecture of the RCVDiff tool. The tool has been implemented in the Java programming language. A logical view of the tool architecture is provided by the package dependencies depicted in Figure 4. The package hierarchy contains three high-
4. TOOL USE CASE

This section provides a small use-case in which two designers work on the same model consecutively, and wish to compare their consecutive versions of that model. Assume that the first designer has created a state machine model $A$ depicted in Figure 6, and that the second designer has changed that model to a model $A'$ depicted in Figure 7.

![Figure 6: Example old model](image-url)

The metamodel of models $A$ and $A'$ is the same and is depicted in Figure 8. Both models $A$ and $A'$, as well as their metamodel $M_A$, conform to the metamodel depicted in Figure 1.

![Figure 7: Example metamodel](image-url)

Notice that both models, as well as their metamodel, are represented in the form of a tree in order to reflect the fact that, at this point, the visualization aspects have not yet been defined. Also, notice that we use $LID$ attributes of metamodel and model elements to denote locally unique identifiers of those elements. These identifiers can be supplied by the tool, or can be automatically generated (we have devised a procedure for the automatic generation of locally unique identifiers that assigns an identical identifier to the same metamodel or model element each time it is invoked on the same metamodel or model [12]). Furthermore, the attributes and references of metamodel elements are assigned (attribute or reference) identifiers as well. The existence of these identifiers is crucial in approaches to model versioning which rely on references as a way of relating elements of differences models to elements of models. Thus, the locally unique identifiers of model elements are used as references of these elements in the differences model. Moreover, model elements reference their conforming metamodel elements and thus metamodel elements must also be equipped with identifiers. Furthermore, identifiers of metamodel elements, attributes and references are also used in the calculation and visualization configuration files.

Next, assume that the first designer would like to inspect the changes to his model in the new model. In order to do that, he must compare the old and the new model, and visualize the obtained model differences.

Our difference calculation tool initially uses a predefined metamodel-independent calculation configuration, however this configuration can (and should) be changed by a domain expert to obtain more accurate results. Each configuration is specific for a metamodel, and thus can be used in comparison of all pairs of models conforming to that specific metamodel. An example of the configuration, specific for the example metamodel $M_A$, is given in Listing 1:

```xml
<ComparisonMMElement name="State" referencedID="1" RID="1">...
<ComparisonMMElement name="Transition" referencedID="2" RID="2">...
</ComparisonMMElement>
</CalculationConfiguration>
```
The calculation configuration of Listing 1 contains three instances of a `ComparisonMMElement` since there are three instances of an `MMElement` in the metamodel. The instance of a `ComparisonMMElement` with an `MMElementID` attribute having the value 0 is used to guide the comparison algorithm when comparing the instances of the metamodel element with `LID` 0 (i.e. state machines). Since state machines have one attribute (name, having an `AID` 0.1), the `ComparisonMMElement` used in comparing state machines contains one instance of the `ComparisonMMAtribute`, related to that attribute. This attribute is set to be the key, thus two state machines that have identical names are considered the same.

The instance of a `ComparisonMMElement` with an `MMElementID` attribute having the value 1 is used to guide the comparison algorithm when comparing the instances of the metamodel element with `LID` 1 (i.e. states). Since states, like state machines, have one attribute, the `ComparisonMMElement` used in comparing states contains one instance of the `ComparisonMMAtribute`. This `ComparisonMMAtribute` uses a comparison function named `compareTwoStrings_Identical`, which considers two state names to be equal if they are identical. Any other user-defined function could be used instead of the used function, the only requirement is that the comparison function must take two strings as arguments, and must return a value between 0 and 1. Then, a configured threshold is used to determine if the two compared attributes are equal.

Next, in order to visualize the differences in a metamodel-specific way, a mapping between the metamodel model and the dot shape needs to be defined. A mapping consists of a set of rules. Each rule is used to map one metamodel element to one dot shape. An example mapping is given in Listing 2:

```
Listing 2: Mapping for visualization of model differences by using MMVisualizer

The first rule in Listing 2 specifies that the state machines (MetamodelElementID: 0) are mapped into containers (Type: TYPE2) which contain graphical representations of states and transitions (SubelementsIDList: 1, 2). The second rule specifies that the states (MetamodelElementID: 1) are mapped into circular shapes (Type: TYPE1, Shape: Circle) which are labeled with the value of the attribute name (LabelAttribute: Name). The third rule specifies that the transitions (MetamodelElementID: 2) are mapped into edges connecting states (Type: TYPE3, FromReferenceID: 1, ToReferenceID: 2).

Next, the designer can choose a set of predefined polymetric views, or she can define and use custom views. It is also possible to define new custom metrics which can be used in the custom views. A custom metric is defined as a Java function that takes as arguments an old model, a differences model, and a metamodel element identifier (LID), and returns a hashtable which contains mappings of all model elements, conforming to the specified metamodel element, to values of type `double`. A definition of an example custom view is given in Listing 3.

```
Listing 3: Custom view

A custom view in Listing 3 specifies that all the model elements should be represented as a tree (LAYOUT: Tree). The length of an element is set to the number of attributes of the element (LENGTH: AttributesNumber). The width of an element is set to the number of references of the element (WIDTH: ReferencesNumber). The color of an element is set to red hue based on the number of changes to the element (COLOR: ChangesNumber). The outline color is set based on the custom metric named `MyCustomMetric`, and the elements are sorted by their color (OUTLINE: MyCustomMetric, SORT: Color).

We will now assume that the differences between models $A$ and $A'$ have been calculated in the calculation part of the tool by using the example calculation configuration presented in this section. Then, the initial result of the visualization part of the tool, using the calculated differences model, and the example configuration files for MMVisualizer and polymetric views, is depicted in Figure 9. The initial result contains the differences represented by using a `GLOBALTREEVIEW` view. Next, if a user clicks the most red element (representing the state machine), the MMVisualizer is activated and produces the result depicted in Figure 10.
5. CONCLUSION

In this paper we described our tool that handles representation, calculation and visualization of model differences. This tool provides a stand-alone Java-based framework for dealing with model differences. This kind of tool is needed because existing tools either deal with only one aspect of model differences (and thus it is hard to combine them), or they are too tightly integrated into a larger CASE tool in order to be easily used in a model configuration management system. We designed our tool to be generic in the following ways: First, it allows the users to use all graph-based metamodels and conforming models. Next, it allows for a highly customizable model comparison process. Furthermore, the results of a model comparison are completely metamodel-independent model differences, represented as models conforming to a differences metamodel, which can be processed further. Finally, our tool allows for a user-guided visualization of model differences. This is achieved by incorporating polymetric views in a framework for visualization of metamodel-based languages.

6. REFERENCES