RCVDiff - a stand-alone tool for representation, calculation and visualization of model differences

Mark van den Brand  
Eindhoven University of Technology  
Den Dolech 2  
5612 AZ Eindhoven  
m.g.j.v.d.brand@tue.nl

Zvezdan Protić  
Eindhoven University of Technology  
Den Dolech 2  
5612 AZ Eindhoven  
z.protic@tue.nl

Tom Verhoeoff  
Eindhoven University of Technology  
Den Dolech 2  
5612 AZ Eindhoven  
t.verhoeoff@tue.nl

ABSTRACT
In model comparison one can distinguish three major sub-problems: 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.

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.

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 EMCompare [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 metamodel depicted in Figure 1.

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

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

The differences models are instances of the DifferencesModel 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 are elements 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 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 metamodel 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 a specific metamodel. Thus, for all models that conform to a

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-level packages named R, C and V. The package named R contains data types for representing models and differences. The package named C contains tool components related to the calculation of model differences. The package named
V contains tool components related to the visualization of model differences.

The high-level data flow diagram of the RCVDiff tool is depicted in Figure 5. The tool has two external "access points". One access point is a differences calculator. The differences calculator receives an old and a new (evolved) model, their common metamodel, and a configuration file, and produces a differences model. The resulting differences model can be processed by other tools, or can be used as input to a differences visualizer, which is the second access point of the tool. The differences visualizer must also receive an old model, a metamodel, a configuration file for polymetric views and a configuration file for MMVisualizer.

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 5: High-level data flow diagram of the RCVDiff tool](image)

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

![Figure 7: Example new model](image)

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.

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.

![Figure 8: Example metamodel](image)

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
<CalculationConfiguration CID="0" metamodelid="SMMM">
  <ComparisonMMElement MMElementid="0" name="StateMachine" type="string" AID="1.1" referencedId=1 referenced=1 reference1::From reference2::To AO3::name value="SM1" version=2>
    <ComparisonMMAttribute attributeID="0.1" key="true" value="true" threshold="0.5" weight="1" sfunctionName="" externalsimfunction="" referencesThreshold="0.5" subobjectsThreshold="0.5"/>
  </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 `MMElem`id 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 `ComparisonMMAttribute`, 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 `MMElem`id 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 `ComparisonMMAttribute`. This `ComparisonMMAttribute` 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.

The instance of a `ComparisonMMElement` with `MMElem`id attribute having the value 2 is used to guide the comparison algorithm when comparing the instances of the metamodel element with LID 2 (i.e., transitions).

Next, in order to visualize the differences in a metamodel-specific way, a mapping between the metamodel model and the `dot` shapes 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:

```xml
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.

```xml
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

RCVDiff - a stand-alone tool for representation, calculation and visualization of model differences

Z. Protić
M.G.J. van den Brand
T. Verhoeff
Our own “Domain Specific” MetaMetaModel
Using our “Domain Specific” MetaMetaModel

- MetaModel is instance-of MetaMetaModel
- Model is instance-of MetaMetaModel
- MetaModel2 is instance-of MetaModel
- Model2 is instance-of Model
- MetaModel3 is instance-of MetaModel
- Model3 is instance-of Model

- MetaModel conforms-to Model
- Model conforms-to Transform (interpret)
- MetaModel2 conforms-to Model2
- Model2 conforms-to Transform (interpret)
- MetaModel3 conforms-to Model3
Why this research?

- Falcon project
- Multiple academic partners and one industrial partner
- Goal: Introduce MDE in the design process of the industrial partner
- This research focuses on the model configuration management part of the process
The problem of industrial partner:
Choosing a MetaMetaModel

- MOF
- Ecore (Eclipse)
  - “A metamodel is simply the model of a model, and if that model is itself a metamodel, then the metamodel is in fact a meta-metamodel.[4] Got it? If not, don’t worry about it, as it's really just an academic issue anyway.”
    - EMF Eclipse Modeling Framework (2nd Edition)
- GME
Model differences

- Representation
- Calculation
- Visualization
Model differences requirements

- **Model based**: The differences should be represented by a formal differences model.
- **Minimalistic**: The differences should contain a minimal number of objects.
- **Self-contained**: The differences model must contain all the information autonomously without relying on data contained in the compared models.
- **Transformative**: It should be possible to transform one model into another model using their differences model.
- **Invertible**: It should be possible to revert back to initial model using the transformed model and their differences model.
- **Compositional**: The result of subsequent or parallel modifications is a differences model whose definition depends only on difference models being composed and is compatible with the induced transformations.
- **Metamodel independent**: The differences metamodel should be independent of a particular metamodel (e.g. UML).
- **Layout independent**: The differences metamodel must be agnostic of the presentation issues.
Model differences representation - Differences MetaModel

- **Model**
  - name: String
  - version: String

- **DifferencesModel**
  - initial
  - final

- **ElementDifference**
  - subelement changes
  - attribute changes

- **ChangedElement**

- **DeletedElement**

- **AddedElement**

- **MElement**
  - subelements

- **MReference**
  - oldReference
  - newReference
  - reference changes

- **AttributeDifference**
  - oldValue
  - newValue

- **AddedReference**

- **DeletedReference**

- **ChangedReference**

- **MAttribute**
  - value: String
Model differences calculation – basic ideas

• Basic ideas adopted from SiDiff (Fujaba)
• Declarative model comparison based on tree-comparison algorithms
• However, our method is highly generic, and thus permits combining imperative and declarative mechanisms in comparison algorithm
Although in its core our approach facilitates a similarity based model matching, it permits all four recognized types of matching:

- Static-identity based
- Signature based
- Similarity based
- Language-specific based
Model Comparison Metamodel
Differences visualization: Basic ideas

• Schneiderman’s mantra for information visualization:
  • Overview first, zoom and filter, then details-on-demand
• Existing techniques if used alone struggle in at least one aspect of information visualization:
  • Text-based techniques have poor overview and filtering
  • Tree-based techniques have problems with details
  • Diagram-based techniques have problems with filtering
• Thus, do not use just one – but combine more techniques!
Differences visualization: Techniques used

• We decided to use polymetric views for overview, zoom and filtering
  • They have been already successfully used in this context

• For details-on-demand we use a framework for visualization of metamodel-based modeling languages
  • Similar to GMF (EuGENia actually)
  • Generic, declarative approach
Polymetric views:
Intro

- A lightweight approach to visualization of graph-based systems
- Basic concepts:
  - **View**: A graphical representation of a set of related elements
  - **Metrics**: Related to the elements in the visualized set
- Idea: Combine different types of **Views** with different system **Metrics** to represent different aspects of a system
  - Calculated metrics are used to determine the size, color or position of the visualized elements in a view
### Polymetric views: Example

#### EXAMPLE METRICS:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>Number of attributes of a model element</td>
</tr>
<tr>
<td>MR</td>
<td>Number of references of a model element</td>
</tr>
<tr>
<td>NC</td>
<td>Number of changes in a model element</td>
</tr>
</tbody>
</table>

#### EXAMPLE VIEW:

<table>
<thead>
<tr>
<th>Name: SYSTEM HOTSPOTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Layout</strong></td>
</tr>
<tr>
<td>Checker</td>
</tr>
</tbody>
</table>
Polymetric views: Examples

MODEL A: name = SM

MODEL A': name = SM

Simplified SYSTEM HOTSPOTS View

Example TRANSITION CHANGES View
Framework for visualization of metamodel-based modeling languages (MMVisualizer)

• In this context visualization can be considered as a way of expressing concrete syntax of a model
• However, our approach is generic, and thus one can also visualize abstract syntax of a model!
• Light, declarative approach, with roots in GMF (Eclipse) and GME
• User defines Rules, which are of predefined type, to map specific metamodel elements into dot elements
  • For each metamodel element one rule may be defined
  • Based on the type of the used rule, model elements conforming to the mapped metamodel element are visualized
MMVisualizer: Example

- **Example rules:**
  - **RULE 1:**
    - Type: TYPE1
    - MetamodelElementID: S1
    - Shape: Circle
    - LabelAttribute: Name
    - AttributesVisualization: HIDDEN
  - **RULE 2:**
    - Type: TYPE3
    - MetamodelElementID: T1
    - FromReferenceID: T1R1
    - ToReferenceID: T1R2
    - FromReferenceShape: none
    - ToReferenceShape: normal
  - **RULE 3:**
    - Type: TYPE2
    - MetamodelElementID: SM0
    - SubelementsIDList: S1, T1
    - LabelAttribute: Name
    - AttributesVisualization: HIDDEN
Example:

MODEL A

A → B → C → D

MODEL A'

A → C → BA → D

Simplified SYSTEM HOTSPOTS View

METAMODEL SPECIFIC View
Open questions

• Should a benchmarking methodology be defined that would allow comparing different approaches to model comparison?
• Should we all use Ecore or MOF, or some other MetaMetaModel to compare our results?