Tool Support for the Evaluation of Matching Algorithms in the Eclipse Modeling Framework

Sabrina Uhrig and Felix Schwägerl
Applied Computer Science 1, University of Bayreuth, Universitätsstr. 30, 95440 Bayreuth, Germany

Keywords: EMF Models, Model Comparison, Matching Algorithms, Evaluation.

Abstract: In the field of model-driven development, sophisticated support for comparing model versions is urgently needed. Unfortunately, algorithms for model matching have been rarely evaluated so far. This paper deals with two extensions to the Eclipse Modeling Framework (EMF) that facilitate the evaluation of matching algorithms for EMF models, with the goal to combine user involvement and automated testing in the evaluation process. First a tree editor is presented that allows for the manual and semi-automated creation of match models which formalize the intended matching result. Second a benchmarking procedure is implemented which – given the intended match and the actual results of matching algorithms – automatically derives the number of α and β errors in a target-performance comparison. These results are valuable for drawing conclusions about the specific qualities of matching algorithms or for finding an adequate set of parameters for a configurable algorithm.

1 INTRODUCTION

Models play a more and more important role in contemporary software engineering processes. Particularly in the field of model-driven development many different versions of models are created before the software is deployed. Therefore reliable comparison tools for models are needed to identify the differences between two versions, which may serve as a basis for merging, for tracing the evolution of the models or for maintenance.

Usually the comparison of two models consists of two steps, the identification of the corresponding model elements and the derivation of the differences based on the computed correspondences. With EMF Compare (Brun and Pierantonio, 2008), the Eclipse Modeling Framework (Steinberg et al., 2009) realizes a model-based approach for the comparison of EMF models[1]. For a 2-way comparison, the corresponding elements are determined by a match engine and stored as an EMF Compare match model. Then a diff engine is used to derive the resulting differences as a diff model. These can be visualized in the EMF Compare editor.

Our research focuses on algorithms for the first step – the matching of model elements – that do not rely on unique object identifiers but use strategies such as edit distance or similarity for the identification of corresponding model elements. In (Uhrig, 2011) different kinds of matching algorithms have been described that are configurable and can be alternatively used instead of the EMF Compare match engine. Beyond the correctness of the results of each single algorithm, which has to be verified, we are faced with the additional problem that different matching algorithms may compute different correct solutions (see example in section 4). This paper addresses the evaluation of these different matching results as the quality of the results may be judged differently by the user. As EMF Compare does not provide any support for measuring or benchmarking comparison results, this paper contributes two tools based on the Eclipse Modeling Framework which enable the evaluation of the different matching algorithms for EMF models, in order to find the preferred matching algorithm and – if it is configurable – the desired parameter configuration.

The rest of this paper is structured as follows: Sections 2 and 3 cover related work and our contribution. Next we provide an overview of the evaluation process and on how the tools interact with EMF. Sections 5 and 6 deal with the implementation and usage of the manual match editor and and the benchmarking tool. After a case study in section 7, section 8 concludes with a short outlook on current and future research.

[1] http://www.eclipse.org/emf/compare/

Uhrig S. and Schwägerl F..
Tool Support for the Evaluation of Matching Algorithms in the Eclipse Modeling Framework.
DOI: 10.5220/0004310801010110
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2013), pages 101-110
ISBN: 978-989-8565-42-6
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

2 RELATED WORK

Although there exist several approaches for model matching or model comparison[2] (Brun and Pierantonio, 2008; Kelter et al., 2005; Lin et al., 2007; Uhrig, 2011; van den Brand et al., 2010), there are few benchmark studies available which compare various algorithms with respect to the quality of the results. Existing evaluation studies are often limited to the approach presented by the authors, e.g. (Kelter et al., 2005). The authors in (van den Brand et al., 2011) compare their own algorithm to only one other approach (Brun and Pierantonio, 2008). The case study in (Kolovos et al., 2009) considers various approaches but contains few examples.

In our opinion, this gap can be put down to the following reasons: On one hand, the implementation of some of the approaches is not available or operates on internal data formats different from EMF Compare. On the other hand, the qualitative evaluation is impeded by the high effort coming with the manual review of the results. An automated evaluation is only possible if the used test sets consist of the input models to compare and the intended output, i.e. the match models for each pair of input models.

When modeling real-world examples, one possibility to obtain intended match models is recording the user's changes on the input model(s) in order to trace corresponding model elements. Unfortunately, the majority of modeling environments, including generated EMF editors, do not include a change-recording mechanism initially. Consequently, it must be plugged into existing tools (Herrmannsdoerfer and Koegel, 2010) or new editors with change-recording support must be provided. As an automation for change-recording, several model generators have been implemented which are designed for different purposes. The Ecore Mutator in the AMOR project[3] addresses the evaluation of model differencing and merging. The model generators described in (Pietsch et al., 2012) and (van den Brand et al., 2011) are designed for the evaluation of model differencing. Another way for obtaining intended match models is the construction of the intended output models by the user/developer, as realized in (van den Brand et al., 2011), which seems to be the preferred method when real-world examples without edit-history should be used or if an edit-history independent interpretation of the changes is desired.

With the manual construction of the intended (diff) models and the method of counting the errors of the computed differences, the approach in (van den Brand et al., 2011) is the closest to our work presented in this paper. We refer to section 6.1 for a detailed comparison of the two evaluation algorithms and just discuss the main issue, the different starting points, here: In (van den Brand et al., 2011), the result of the difference calculation is evaluated, whereas our approach compares the resulting match models with an intended match model. We consider the evaluation of the used match and diff engines in the first and the second step of model comparison as separate concerns, as it is possible to deduce different diff models from the same match model by using different edit operations (e.g. composed operations such as move and copy instead of atomic operations). This is also the point in which our evaluation approach differs from the other benchmarking strategies mentioned above.

3 CONTRIBUTION

The evaluation approach we present combines user involvement and automated testing and has been implemented by two tools: The first tool, a tree editor for EMF Compare match models, permits the manual construction of intended match models, which – together with the models to compare – form a test set. The second tool compares actual match models to the intended match models[4]. Our approach considers the following aspects:

Match-based Error Calculation. Our qualitative evaluation addresses the match models computed by matching algorithms, which are the basis for the ensuing deduction of differences in the second step of model comparison, and for the merging of models.

State-based Evaluation. We only take the actual states of the models into account instead of transforming the recorded edit operations into the intended match model. The fact that state-based differences may differ from history-based differences has been discussed earlier in the context of unique object identifiers (Ohst, 2004; Uhrig, 2011).

User Involvement. The manual match editor introduces user involvement into the evaluation process. This leads to extended evaluation capabilities as realistic test models for which no edit-history is available can be handled, too.

Construction of Match Model. The user is expected to model the intended match himself

[2] which include the step of model matching implicitly

[3] http://www.modelversioning.org/

[4] This step does not require another match based model comparison, as only identical match elements have to be identified for the automated calculation of errors

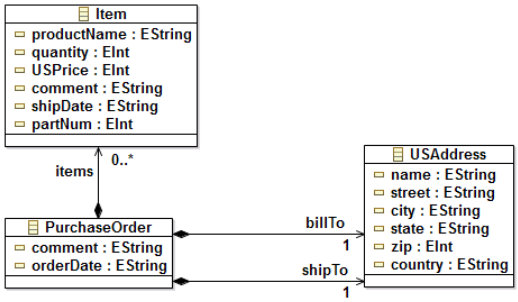

Figure 1: Ecore class diagram A.

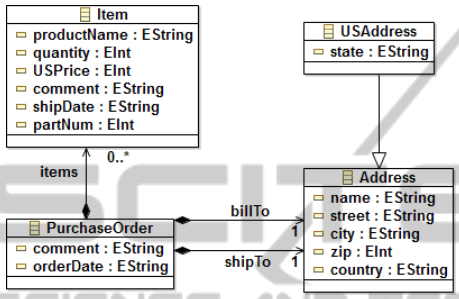

Figure 2: Ecore class diagram B.

from scratch. So we assume to get more reliable solutions than by reviewing existing solutions as the user is not influenced by the presented suggestions.

Semi-automated Matching. The manual match editor provides semi-automated sub-tree matching procedures in order to reduce the effort for the creation of the intended match model. This is recommended especially in the case of large models with minor changes.

Automated Error Calculation. The manually defined match model can be used for the benchmarking of several algorithms without the need for repeated manual reviews, as error calculation itself is completely automated.

4 TOOL OVERVIEW

As mentioned in the introduction, the comparison of a pair of models consists of two steps. In the matching step, corresponding elements are identified for the given model versions. Two elements correspond if they are equal or sufficiently similar in the view of the matching algorithm used (or the user). In the differencing step, change operations describing the differences are derived: Elements which do not correspond to another element have been deleted or created. Further the details of each pair of corresponding elements are compared in order to identify change operations like name changes. The fact that the derived model difference will vary depending on the computed correspondences is demonstrated by example 4.1.

4.1 Example

Figures 1 and 2 show two versions of an Ecore class diagram which models the well-known purchase order example in (Steinberg et al., 2009, p. 70). It is obvious that the classes `Item`/`Item` and `PurchaseOrder`/`PurchaseOrder` correspond, respectively. So two options for the class `USAddress` remain: If the class `USAddress` in Version A is matched with `Address` in version B, we get the following differences: the name change of the class `Address` to `USAddress`, the insertion of the attribute `state` and the deletion of the class `USAddress`. If we match both `USAddress` classes instead, we get the deletion of the class `Address`, change of the targets of the references `shipTo` and `billTo` and the insertion of the attributes `name`, `street`, `city`, `zip` and `country` into the class `USAddress`.

4.2 Proposed Workflow

Figure 3 illustrates the workflow that we propose for the evaluation of two algorithms A and B against a manual match model on the basis of the model pair `E1.ecore` and `E2.ecore`. Any matching algorithm that produces its result as an EMF Compare match model can be plugged into the framework. For the modeling of the intended match, which serves as a reference for the results of the two algorithms, the user is provided with the manual match editor described in section 5.3. The match models obtained from both the algorithms and the manual match editor are converted into diff models using the EMF Compare diff engine. The result, a model difference, is serialized in an EMF diff resource together with its match model.

In the evaluation step, the results of the algorithms (these are the actual match models in figure 3) are compared to the model that has been chosen as the intended match model (e.g. the manually created match model). Resulting false positive and false negative errors are collected in a *CSV* file (comma separated values) which can be used as a basis for further statistical investigations.

While the workflow described above only deals with the comparison of one single pair of Ecore models with two algorithms, we assume that a pairwise comparison of a number N of Ecore models with a number A of matching algorithms shall be performed. In sum, $A \cdot \frac{N(N-1)}{2}$ match models are computed. In-

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

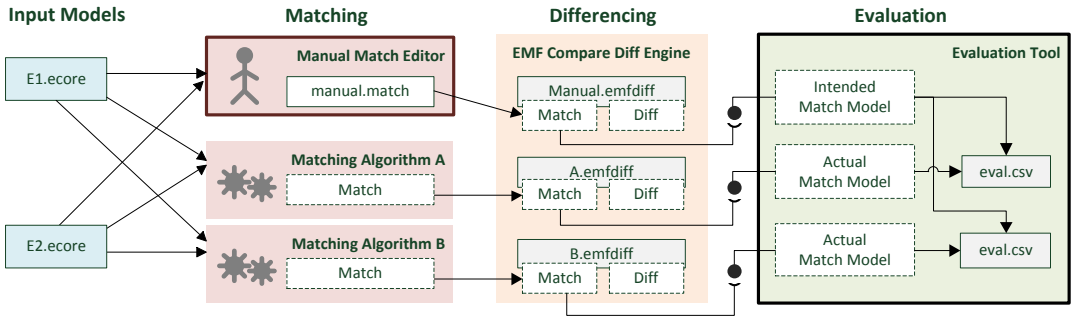

Figure 3: Comparing two Ecore models with two matching algorithms and evaluating the results versus a manually created match model.

volving the user in all cases requires the manual creation of $\frac{N(N-1)}{2}$ additional match models.

Although we propose to create the intended match model manually from scratch for the reasons we discussed in section 3, the tools presented in this paper also support other workflows: It is possible to select the result of a matching algorithm as the intended model, to edit generated match models in the manual match editor in order to obtain the intended model or to support the modeling from scratch with the help of semi-automated sub-tree matching procedures.

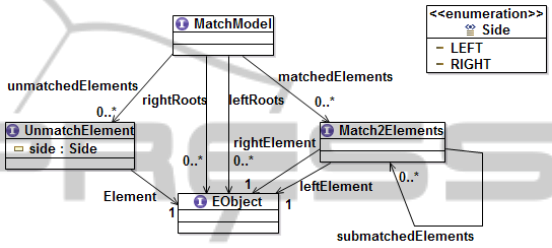

Figure 4: EMF Compare match metamodel (simplified).

5 THE MANUAL MATCH EDITOR

In order to save the corresponding elements found by the user, a match model has to be created. As the Eclipse Modeling Framework does not provide an editor for match models, the implementation of a user interface for the creation and editing of match models was necessary.

5.1 EMF Compare Match Metamodel

In the Eclipse Modeling Framework the results of a matching procedure are stored in a match model which conforms to the EMF Compare match metamodel shown in figure 4. A `MatchModel` consists of a set of match elements that link corresponding objects of the compared models. We focus on 2-way matches which are represented by the model class `Match2Elements`. It connects two arbitrary EMF model objects (`EObject`) from the left and right input model by the `leftElement` and `rightElement` references. The matches are organized in a containment tree by the `submatchedElements` self-reference. The hierarchy of match elements must conform to the hierarchy of the models to match. Furthermore, a match model contains a set of instances of `UnmatchElement` of both models. These elements refer to model objects that cannot be assigned a corresponding element on the opposite side. The attribute `side` describes whether the referenced model element is contained in the left or in the right model.

5.2 Implementation Approach

The fact that the EMF Compare match metamodel was available led us to the development of the user interface in a model-driven way. In (Steinberg et al., 2009, part 3) the generation of Java source code for tree editors based on existing EMF metamodels is described. The generated code comprises three plug-ins *model*, *edit* and *editor*. We customized these plug-ins in order to provide the user with a familiar interface and to preserve the consistency of the match model while he/she builds up the matching tree from scratch.

The generated *edit* plugin controls aspects of representation and manipulation of model instances. Classes of this plug-in have been customized in order to constrain or adapt the editing operations on corresponding models. In order to display match elements in a way the user is familiar with (e.g. icons and labels of tree nodes), we obtain an *item provider* specific to the metamodel of input models from the global EMF registry. This mechanism does not require any "hard-wired" domain-specific configuration.

The *editor* plug-in provides a preliminary user interface in the form of a tree-based editor that we used as a basis for further customizations.

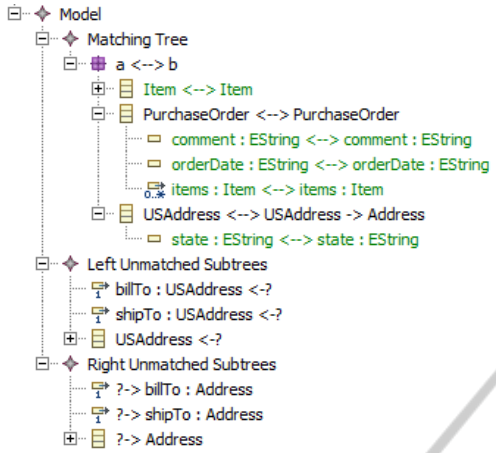

Figure 5: The manual match editor's main pane.

5.3 Usage of the Customized Editor

The manual match editor facilitates the process of user-defined matching in two ways. On the one hand, it can be used to build match models from scratch. On the other hand, it allows for the manipulation of existing match models.

5.3.1 Creating a Match Model from Scratch

The generated EMF tree editor requires an instance of `MatchModel` as the root of the editable tree. A customized wizard asks the user for a left and a right resource containing valid EMF models. Afterwards, the user has to select a root object from each input model, which will result in the creation of a top-level `Match2Elements` instance. Each other object part of the left or right model is added to the `unmatchedElements` set automatically, assigning its corresponding `Side`. The created model is persisted within a `.match` resource when the wizard is closed.

5.3.2 Extracting Matches from EMF Diffs

Typically, match models are part of an EMF diff resource which results from a two-step model comparison including matching and differencing. In order to allow for the manipulation of existing match models, a `.match` file can be extracted from a valid `.emfdiff` resource using the respective context menu entry. This is the recommended procedure in case there exists a nearly optimal matching result that only requires minor modifications.

5.3.3 Synthesizing a Match Model

A screenshot of the manual match editor is given in Figure 5. In its tree representation, the match model

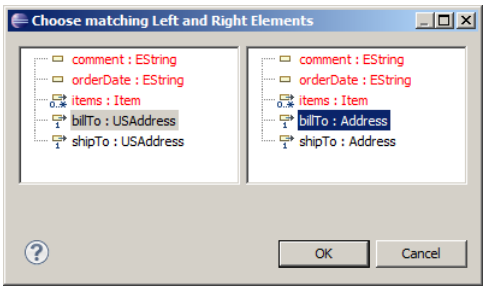

Figure 6: Selecting a pair of corresponding model elements in a dialog.

is grouped by three additional top-level nodes, containing the current *Matching Tree* and the *Unmatched Elements* of the left and right side. The *Matching Tree*'s content is given by the currently edited match model. For reasons of clarity and in order to facilitate the identification of pairs of corresponding model elements, the contents of the *Unmatched Elements* nodes are not displayed as a list, which is the default in the EMF Compare match metamodel, but as a sequence of sub-trees.

The matching tree itself is to be synthesized by the user step by step. For this purpose, the user can double-click on an existing parent match element, where a new pair of corresponding child elements shall be inserted. The subsequent dialog lets the user choose a pair of objects from the left and right input models (cf. Figure 6). In order to preserve the hierarchy, these must be either directly or indirectly contained by the respective parent match element. Accordingly, the user's choice is restricted to the sub-trees of the left and right parents, i.e. the editor takes care that the created match model will be valid. Elements of different hierarchical levels can be matched, which will result in the detection of move operations by the EMF Compare diff engine. In addition, a red coloring of objects that are already part of another match element facilitates the identification of unmatched elements and prevents the user from matching an element twice by mistake.

Having selected a pair of objects, a corresponding instance of `Match2Elements` is inserted as child element of the selected parent and shown in the editable tree. At the same time, corresponding entries are deleted from the *Left* and *Right Unmatched Subtrees* nodes in order to ensure the consistency of the overall match model. Conversely, unmatched elements are re-added when matches are deleted by the user. For the purpose of consistency, the direct manipulation of unmatched subtrees has been prohibited, too.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

5.3.4 Semi-automated Matching

Further customizations of the generated editor code facilitate the editing of match models. To begin with, the user is notified by a green coloring of match elements in the main pane when the corresponding mapping is "complete". This transitive property is true when under a given match element, all direct or indirect child elements of the left and right referenced model elements are already matched.

Another method of convenience was added in order to avoid the need of insertion of "trivial" match elements. For a sub-tree, an automated matching of child elements with pairwise identical labels can be invoked recursively by an extra context menu entry. This reduces the effort for a slightly differing model pair but should not be used without manual review by the user.

5.3.5 Exporting the Match Model

When the editing of the match model has been finished by the user, he/she can export the results using the respective context menu entry. Both match and derived diff are serialized inside an EMF diff resource which is needed for the subsequent evaluation steps. Besides exporting an `.emfdiff`, the user can inspect the comparison result in the EMF Compare editor.

6 BENCHMARKING MATCHING ALGORITHMS

In order to measure the qualities of different matching algorithms, we need procedures that benchmark their results. According to our definition, benchmarking matching algorithms means counting the number of errors in the resulting match models for each of them.

Our evaluation tool comprises support for both the execution of matching algorithms and the calculation of a score which counts the number of errors in their results, compared to the intended result. Both procedures run subsequently in a batch environment, comparing a number of pairs of EMF based input models using selected matching algorithms which are typically implemented by so called match engines. The user can control and parametrize the overall process in a graphical interface.

6.1 Computation of Errors

Before considering batch procedures, let us present the algorithm which calculates the number of errors, given an actual and an intended match model that both refer to the same pair of input models. In the following, M states the intended match model, which may have been modeled manually by the user, and M' represents a match model which is the actual result calculated by a matching algorithm. As in (Kappel et al., 2007), a match element that is contained in M' but not in M is regarded as *false positive* (α error). Each match element in the intended model that is not contained in the resulting model is a *false negative* (β error). Given an actual match model M' and an intended match model M, the basic idea how the numbers of false positive errors α and false negative errors β are calculated is described in the following:

1. $\alpha := 0, \beta := 0$.
2. For each match element m from the M' matching tree:
   (a) If M contains a match element that targets the same pair of left and right model elements as m, continue.
   (b) Else, $\alpha := \alpha + 1$.
3. For each match element n from the M matching tree:
   (a) If M' contains a match element that targets the same pair of left and right model elements as n, continue.
   (b) Else, $\beta := \beta + 1$.
4. return α, β.

In the actual implementation, we reduced the number of comparison steps by linearizing both matching trees into lists of match elements that are additionally grouped by metamodel classes[5]. After having identified two identical match elements in step 2, these are removed from both lists in order to reduce the number of subsequent comparison steps. In step 3, it is only necessary to count the remaining matches because due to commutativity and uniqueness of matches, we can assume that no further corresponding match exists in the linearized structure after step 2.

The values α and β are absolute and relate to the number of match elements inside the test set which have been classified false positive or false negative. The characteristics *precision* (p) and *recall* (r), in contrast, are relative and provide a comprehensive comparison criterion. For a given evaluation result, these values can be calculated directly from α and β and the value $|M|$ which represents the number of *true positive*, i.e. intended matches:

$$p = \frac{|M|}{|M| + \alpha} \qquad r = \frac{|M|}{|M| + \beta}$$

[5]This implies that only objects which belong to the same metamodel class can be matched.

In (van den Brand et al., 2011) a related method is proposed for the evaluation of model differences which constitute the result of different comparison algorithms to evaluate. Differences are classified into three distinct categories: *added*, *deleted* and *changed*. In contrast to our method, which takes matches and not differences into account, moves cannot be detected. Differences are considered identical when they belong to the same category and refer to the same element. The set of intended ("correct") differences is compared to the set of actual ("found") differences in order to determine the intersection set ("matching differences") which serves as a basis for the calculation of *precision* and *recall* values.

6.2 Batch Comparison and Evaluation

The overall benchmarking of matching algorithms can be controlled with a graphical user interface which is provided as an Eclipse view. The comparison and evaluation procedures do not explicitly depend on each other but they are meant to be used subsequently. A screenshot of the batch user interface (UI) is given in Figure 7.

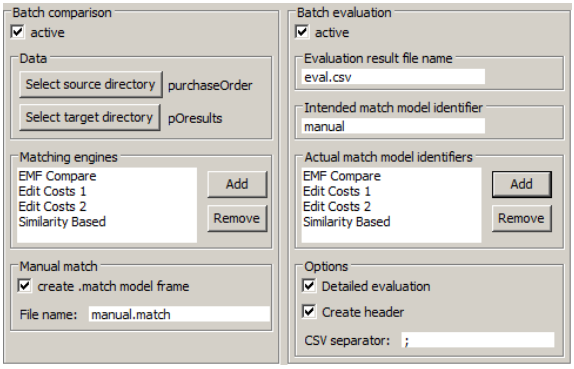

Figure 7: The batch UI.

In the left-hand part of the view, the comparison phase is configured. The user can choose a directory containing a number of files containing models conforming to the same metamodel which will be compared pairwise. Also, an output folder must be selected where the calculated `.emfdiff` files containing match and diff models will be placed for each compared model pair. Below, the user can add the matching algorithms to compare[6] and configure their parameters if supported by the respective match engine. Furthermore he/she has to assign a unique name to each chosen engine which is needed to identify the generated outputs in the second phase.

When the first batch comparison phase is started, each of the selected match engines produces an `.emfdiff` file per model pair and engine, where the file name is given by the identifier assigned to the engine. The batch procedure automatically organizes the resulting files inside a new sub-folder per model pair. A convenience option inside the left-hand part of the batch UI allows for the creation of an empty match model inside each sub-folder referring to a pair of compared models. This is where the manual match editor described in section 5.3 can be applied to provide the intended result manually before the second batch phase starts.

The right-hand part of the view covers the benchmarking of the matching algorithms. The UI components let the user choose identifiers that have been assigned in the left-hand part for the intended or actual match models. Iterating through the sub-folders of the outpout directory, each representing one pair of input models, the second batch procedure benchmarks all actual match models specified against the intended match model using the algorithm described in section 6.1. In the view, the user can also specify the name of the CSV file to contain the benchmark result, i.e. α and β errors broken down to every compared pair and per matching algorithm. An additional option allows for a more detailed report which itemizes the α and β errors by the respective metamodel class (see example in Table 2).

7 CASE STUDY

In order to evaluate the use and the usability of our tools we applied them to a real-world case study of versioned Ecore models of the *ModGraph* project. ModGraph (Winetzhammer, 2012) is a tool for describing the behavior of EMF models by graph transformations. We chose six different versions of the ModGraph metamodel, which contain between 102 and 168 objects. Our formalized intended matching result comprises 15 diagram pairs, each including from 86 up to 132 match elements.

We compared the results of the following three matching algorithms:

EMF Compare Match Engine (emf). (Brun and Pierantonio, 2008) The generic match engine can be applied to the comparison of any pair of EMF based models conforming to the same metamodel. The algorithm works in a hierarchical and similarity-based way. We used the version 1.3.1 installed in Eclipse Juno and disabled the

[6] A dialog lists all extensions of the extension point `org.eclipse.emf.compare.match.engine` found in the registry of the current Eclipse installation.

Table 1: Key figures from the evaluation run. Numbers of matches and errors constitute the sum of all 15 test runs, while the precision and recall numbers are average values.

matching algorithm	intended matches	actual matches	correct matches	α errors	β errors	precision	recall
Similarity Based (sim)	1477	1401	1308	93	169	0.9357	0.8878
Estimated Edit Costs (ec)	1477	1274	1207	67	270	0.9525	0.8367
Estimated Edit Costs with Threshold (ec th)	1477	1241	1455	22	236	0.9841	0.8582
EMF Compare Match Engine (emf)	1477	1533	1404	129	73	0.9079	0.9442

options which rely either on functional or XMI identifiers.

Similarity based Matching (sim). (Uhrig, 2011) This matching algorithm is specialized on Ecore class diagrams. It computes similarity values for all possible correspondences and finally selects the matches with the largest total similarity values using a heuristic greedy strategy. The weights of the single similarity values are configurable. During the batch procedure, we kept the default configuration.

Matching with Estimated Edit Costs (ec). (Uhrig, 2011) In contrast to similarity-based approaches, this algorithm for Ecore class diagrams considers the matching as an optimization problem that tries to minimize the edit costs, e.g. the costs of transforming one object into another by changing its details. Optionally, a threshold prevents the unintentional matching of a pair of `EClass` instances with a too high editing distance. For the evaluation we used two instances of the matching algorithm, the default configuration and another one with an activated threshold.

The batch comparison step calculated matches for each of the 15 possible combinations of the six input models and four selected match engine instances. Before the evaluation, we used the manual match editor to formalize our intended matching result for each model pair. In many cases, classes had been renamed or split between the model versions. Nevertheless, the semi-automated matching support facilitated the manual matching procedure. Finally, we started the batch evaluation where the manual match was specified as the intended result.

The results are displayed in Table 1: While the intended match models contain 1477 match elements in total, the number of match elements found in the actual match models varies from 1241 up to 1533 (see column 2). The comparison of the match elements delivers the number of correct match elements that are identical in both the actual and the intended match model (see column 3), the α and β errors and the derived precision and recall values. As we can see, each of the tested matching algorithms produced errors compared to the user-defined intended match models. The emf algorithm attains the highest number of false positive matches and the lowest number of false negative matches. For the given test set, we can draw the conclusion that emf tends to show a rather greedy behavior and produces a surplus of matches, resulting in a lower precision value. Table 2 presents the detailed key figures for each pair of models compared. Having a closer look at the last four columns, which contain the α and β errors divided into the different metamodel classes, one will realize that the high total number of α errors is due to false positive matches of `EReference` and `EGenericType` instances.

By interpreting the results of the matching algorithms, one can also draw conclusions about the impact of different parametrizations of the same matching algorithms: The basic configuration of the editing distance algorithm (ec) attains 67 α and 270 β errors. The use of the threshold option (ec th) obviously helps reducing the number of α errors, in our example even by a factor of three. Considering the detailed figures again, we realize that the matching was especially improved for instances of `EClass` and their contained structural features. In a similar way, our evaluation tool could also help finding an optimal parametrization for the weights of the similarity values of the sim algorithm.

8 CONCLUSIONS

In this paper we presented tool support for benchmarking different matching algorithms by comparing their matching results against an intended model. Both tools can be plugged into the Eclipse Modeling Framework and deal with arbitrary EMF models. The manual match editor allows to create the intended match model either from scratch or by revising the nearly optimal match created by an algorithm. Therefore user involvement is supported in the evaluation process and intended match models can be created in a semi-automated way even if no edit-history is avail-

Table 2: Detailed key figures of the case study performed on a Intel Core i7@2,60GHz machine (single threaded).

left and right model	matching algorithm	intended matches	actual matches	intended unmatched elements	actual unmatched elements	total α errors	total β errors	precision	recall	runtime (msec)	α/β EClass	α/β EAttribute	α/β EReference	α/β EGeneric-Type
1 2	sim	86	78	40	22	4	12	0.9535	0.8723	25	1/1	0/0	1/5	2/6
	ec		73		26	0	13	1	0.8687	16	0/0	0/0	0/6	0/7
	ec th		73		27	1	14	0.9884	0.8586	29	1/1	0/1	0/5	0/7
	emf		90		12	7	3	0.9186	0.9634	38	1/1	0/1	3/0	3/1
1 3	sim	86	78	38	21	4	12	0.9535	0.8723	14	1/1	0/0	1/5	2/6
	ec		74		24	5	17	0.9418	0.8265	15	1/1	1/1	1/6	2/9
	ec th		73		26	1	14	0.9884	0.8586	16	1/1	0/1	0/5	0/7
	emf		92		9	9	3	0.8953	0.9625	26	1/1	0/1	4/0	4/1
1 4	sim	86	74	40	30	4	16	0.9535	0.8367	39	2/2	0/2	1/4	1/8
	ec		74		28	8	20	0.9070	0.7959	12	2/2	1/2	2/6	3/10
	ec th		71		30	1	16	0.9884	0.8416	13	1/1	0/2	0/5	0/8
	emf		84		17	3	5	0.9651	0.9432	48	1/1	0/2	1/0	1/2
1 5	sim	82	70	84	49	2	14	0.9756	0.8511	19	2/2	0/2	0/4	0/6
	ec		72		44	5	15	0.9390	0.8370	15	1/1	1/2	1/5	2/7
	ec th		67		45	0	15	1	0.8454	14	0/1	0/2	0/5	0/7
	emf		81		34	4	5	0.9512	0.9398	26	0/1	0/2	1/0	3/2
1 6	sim	79	71	81	41	19	27	0.7595	0.6897	11	8/8	0/1	4/6	7/12
	ec		68		44	12	23	0.8481	0.7444	11	6/6	1/2	1/4	4/11
	ec th		63		44	7	23	0.9114	0.7579	28	5/6	0/2	0/4	2/11
	emf		83		29	21	17	0.7342	0.7733	28	6/6	0/2	4/1	11/8
2 3	sim	114	114	6	3	0	0	1	1	12	0/0	0/0	0/0	0/0
	ec		102		9	0	12	1	0.9048	11	0/0	0/0	0/6	0/6
	ec th		102		9	0	12	1	0.9048	10	0/0	0/0	0/6	0/6
	emf		116		1	2	0	0.9825	1	19	0/0	0/0	1/0	1/0
2 4	sim	103	101	30	19	0	2	1	0.9810	12	0/0	0/1	0/0	0/1
	ec		89		25	1	15	0.9903	0.8718	11	1/1	0/1	0/6	0/7
	ec th		89		24	0	14	1	0.8803	10	0/0	0/1	0/6	0/7
	emf		107		11	6	2	0.9417	0.9798	22	0/0	0/1	3/0	3/1
2 5	sim	95	91	82	41	6	10	0.9368	0.8990	19	2/2	0/2	0/1	4/5
	ec		80		52	5	20	0.9474	0.8182	13	3/3	1/2	0/6	1/9
	ec th		81		49	0	14	1	0.8716	12	0/1	0/1	0/6	0/6
	emf		104		30	14	5	0.8526	0.9419	20	0/1	0/1	4/0	10/3
2 6	sim	95	91	73	36	10	14	0.8947	0.8586	14	4/4	0/1	2/2	4/7
	ec		80		43	4	19	0.9579	0.8273	14	3/3	0/1	0/5	1/10
	ec th		81		44	3	17	0.9684	0.8440	31	2/2	0/1	0/5	1/9
	emf		106		23	18	7	0.8105	0.9167	90	1/1	0/1	5/0	12/5
3 4	sim	105	103	24	16	0	2	1	0.9813	11	0/0	0/1	0/0	0/1
	ec		91		22	1	15	0.9905	0.8739	8	1/1	0/1	0/6	0/7
	ec th		91		21	0	14	1	0.8824	8	0/0	0/1	0/6	0/7
	emf		105		12	2	2	0.9810	0.9810	14	0/0	0/1	1/0	1/1
3 5	sim	99	93	72	38	4	10	0.9596	0.9048	14	2/2	0/2	0/1	2/5
	ec		82		49	5	22	0.9495	0.8103	13	3/3	1/2	0/6	1/11
	ec th		83		46	0	16	1	0.8609	10	0/1	0/1	0/6	0/8
	emf		102		31	8	5	0.9192	0.9479	17	0/1	0/1	2/0	6/3
3 6	sim	96	93	69	33	10	13	0.8958	0.8687	13	4/4	0/1	2/2	4/6
	ec		82		40	4	18	0.9583	0.8364	12	3/3	0/1	0/5	1/9
	ec th		83		41	3	16	0.9688	0.8532	9	2/2	0/1	0/5	1/8
	emf		104		24	15	7	0.8438	0.9205	69	1/1	0/1	3/0	11/5
4 5	sim	110	108	52	22	6	8	0.9455	0.9286	14	2/2	0/1	1/1	3/4
	ec		93		37	5	22	0.9545	0.8268	13	3/3	1/1	0/7	1/11
	ec th		95		35	1	16	0.9910	0.872	10	1/1	0/0	0/7	0/8
	emf		113		21	6	3	0.9455	0.9720	15	0/1	0/0	1/0	5/2
4 6	sim	109	108	45	19	7	8	0.9358	0.9273	13	3/3	0/0	1/1	3/4
	ec		95		26	4	18	0.9633	0.8537	14	3/3	0/0	0/6	1/9
	ec th		96		27	3	16	0.9725	0.8689	9	2/2	0/0	0/6	1/8
	emf		114		16	10	5	0.9083	0.9519	17	1/1	0/0	1/0	8/4
5 6	sim	132	128	34	24	17	21	0.8712	0.8456	19	8/8	0/1	2/2	7/10
	ec		119		32	8	21	0.9394	0.8552	39	4/4	0/0	0/7	4/10
	ec th		115		34	2	19	0.9848	0.8725	11	1/2	0/0	0/7	1/10
	emf		132		16	4	4	0.9697	0.9697	19	1/1	0/0	0/1	3/2

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

able. The batch view controls the invocation of match engines for the pairwise comparison of models in a test set. The proposed benchmarking tool is configurable in terms of input models, algorithms to compare and the level of detail of the reported α and β errors.

In a case study, we used the presented manual match editor in order to model the intended matching results for our test cases, which belong to a real-world example for which only different versions but no edit-history was available. The benchmark procedure with evaluation support helped to gain insights on how the different matching algorithms behave when being applied to a certain set of test cases. If the results of the algorithms are not identical with the intended match model, it is recommended to inspect the detailed reports of α and β errors in the different categories of model elements in order to judge the overall performance of the tested algorithms. Consequently, the user can select his/her preferred procedure and – if necessary – adapt the parametrization of configurable matching algorithms in order to obtain better results.

Future work might address the editing comfort of the manual match editor. We think that editing a match model could be facilitated on the one hand by extending the semi-automated matching capabilities by supporting the partial application of matching algorithms for sub-trees. On the other hand, we consider a graphical (e.g. GMF[7] based) editor for the model matching since working on a tree representation is not ideal for users only familiar with concrete graphical syntax. In contrast to the generically implemented tree editor, a specific graphical editor must depend on the metamodel of matched EMF models in order to be able to integrate their concrete graphical syntax. Additionally, we plan to extend our evaluation tool in order to support multiple intended match models. For each compared model pair and algorithm, the minimum of the error figures obtained from all intended models could be regarded as a measure of the quality of results.

REFERENCES

Brun, C. and Pierantonio, A. (2008). Model differences in the eclipse modelling framework. *UPGRADE*, IX(2):29–34.

Herrmannsdoerfer, M. and Koegel, M. (2010). Towards a generic operation recorder for model evolution. In *Proceedings of the 1st International Workshop on Model Comparison in Practice*, IWMCP '10, pages 76–81, New York, NY, USA. ACM.

Kappel, G., Kargl, H., Kramler, G., Schauerhuber, A., Seidl, M., Strommer, M., and Wimmer, M. (2007). Matching metamodels with semantic systems - an experience report. In *Datenbanksysteme in Business, Technologie und Web (BTW 2007), Workshop Proceedings*, pages 38–52. Verlag Mainz.

Kelter, U., Wehren, J., and Niere, J. (2005). A generic difference algorithm for UML models. In Liggesmeyer, P., Pohl, K., and Goedicke, M., editors, *Software Engineering 2005*, Lecture Notes in Informatics, pages 105–116. Gesellschaft für Informatik.

Kolovos, D. S., Di Ruscio, D., Pierantonio, A., and Paige, R. F. (2009). Different models for model matching: An analysis of approaches to support model differencing. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, CVSM '09, pages 1–6, Washington, DC, USA. IEEE Computer Society.

Lin, Y., Gray, J., and Jouault, F. (2007). DSMDiff: A Differentiation Tool for Domain-Specific Models. *European Journal of Information Systems*, pages 349–361.

Ohst, D. (2004). *Versionierungskonzepte mit Unterstützung für Differenz- und Mischwerkzeuge*. PhD thesis, Universität Siegen, Siegen, Germany.

Pietsch, P., Yazdi, H. S., and Kelter, U. (2012). Controlled generation of models with defined properties. In Jähnichen, S., Küpper, A., and Albayrak, S., editors, *Software Engineering 2012*, Lecture Notes in Informatics, pages 95–106. Gesellschaft für Informatik, Bonn.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF Eclipse Modeling Framework*. The Eclipse Series. Addison-Wesley, Upper Saddle River, NJ, 2nd edition edition.

Uhrig, S. (2011). *Korrespondenzberechnung auf Klassendiagrammen*. PhD thesis, Universität Bayreuth, Bayreuth, Germany.

van den Brand, M., Hofkamp, A., Verhoeff, T., and Protić, Z. (2011). Assessing the quality of model-comparison tools: a method and a benchmark data set. In *Proceedings of the 2nd International Workshop on Model Comparison in Practice*, IWMCP '11, pages 2–11, New York, NY, USA. ACM.

van den Brand, M., Protić, Z., and Verhoeff, T. (2010). Fine-grained metamodel-assisted model comparison. In *Proceedings of the 1st International Workshop on Model Comparison in Practice*, IWMCP '10, pages 11–20, New York, NY, USA. ACM.

Winetzhammer, S. (2012). Modgraph - generating executable emf models. In Margaria, T., Padberg, J., Taentzer, G., Krause, C., and Westfechtel, B., editors, *Proceedings of the 7th International Workshop on Graph Based Tools*, volume 54 of *Electronic Communications of the EASST*, pages 32–44, Bremen, Deutschland.

[7] http://www.eclipse.org/modeling/gmp/