A Survey of Model Comparison Approaches and Applications

Matthew Stephan and James R. Cordy
Queen's University, Kingston, Ontario, Canada
{*stephan, cordy*}*@cs.queensu.ca*

Keywords: Model Comparison: Model-Driven Engineering: Model Versioning: Model Clone Detection

Abstract: This survey paper presents the current state of model comparison as it applies to Model-Driven Engineering. We look specifically at how model matching is accomplished, the application of the approaches, and the types of models that approaches are intended to work with. Our paper also indicates future trends and directions. We find that many of the latest model comparison techniques are geared towards facilitating arbitrary meta models and use similarity-based matching. Thus far, model versioning is the most prevalent application of model comparison. Recently, however, work on comparison for versioning has begun to stagnate, giving way to other applications. Lastly, there is wide variance among the tools in the amount of user effort required to perform model comparison, as some require more effort to facilitate more generality and expressive power.

1 INTRODUCTION

Model-Driven Engineering (MDE) involves using high-level software models as the main artifacts within the development cycle. As MDE becomes more prevalent in software engineering, the need for effective approaches for finding the similarities and differences among high-level software models, or *model comparison*, becomes imperative. Since MDE involves models being considered *first class* artifacts by developers, it is clear model comparison is important as it assists model management and may help facilitate analysis of existing projects and MDE research. It not only provides merit as a stand-alone task, but helps engineers with other MDE tasks such as model composition and inferring and testing of model transformations (Kolovos et al., 2006).

Despite model comparison's importance in MDE, there are no definitive surveys on the state of the art in model comparison research. There are a few papers that touch upon it, however, they look at only a very small subset of the work in existence and either at those intended for only a specific model type or application. In this survey paper, we present the current state of model comparison research and discuss the area's future directions. The purpose in doing so is to investigate and describe the approaches to accomplish model comparison, the various applications that approaches are used for, and to categorize the approaches by the types of models they are able to compare. As such, this survey can be used by engineers as a quick reference guide, organized by the types of models being compared: If they must work with a specific model type or need a specific application to be accomplished via model comparison, they can use this survey to determine if an approach is right for them or if they should build upon an existing one.

We begin with background information on model comparison in Section 2. Section 3 categorizes and describes the existing model comparison approaches by the type and subtype of models that they compare. Section 4 provides a summary of the survey results and future directions of the area. Section 5 identifies other techniques and approaches related to the model comparison work discussed in this survey, as well as other surveys.

2 BACKGROUND

We begin by defining model comparison as it applies in this survey. Much of the existing work can be split into two categories: those techniques aimed at model versioning, and those aimed at model similarity analysis, or "clone detection", so we additionally define those categories.

2.1 Model Comparison

This section discusses model comparison as a task in MDE and refers specifically to the act of identifying similarities or differences between model elements. Aside from versioning and model clone detection, model comparison has merit in other areas of MDE including being the foundation for *model composition*, *model transformation testing* (Kolovos et al., 2006; Lin et al., 2004; Stephan and Cordy, 2013), and others discussed in this survey.

Kolovos, Paige, and Polack (2006) define model comparison as an operation that classifies elements into four categories: (1) Elements that match and conform, (2) Elements that match and do not conform, (3) Elements that do not match and are within the domain of comparison, and (4) Elements that do

not match and are not within the domain of comparison. Matching refers to elements that represent the same idea or artifact, while conformance is additional matching criteria. An example of non conformance in an UML class diagram can be when a class in both models has the same name but one is abstract. So while they likely represent the same artifact, they do 'match enough' or conform to one another (Kolovos et al., 2006). The specific definition of conformance is dependent on the current comparison context and goal. The domain of the comparison can be viewed as matching criteria. Non-matching elements that are outside the domain of comparison can come from matching criteria that is either incomplete or intentionally ignoring them.

In the context of model versioning, model comparison has been decomposed into three phases (Brun and Pierantonio, 2008): Calculation, Representation, and Visualization. However, there is nothing about this decomposition that is specific to model versioning. In the following paragraphs we elaborate on these phases and provide examples of approaches.

Calculation Calculation is the initial step of model comparison and is the act of identifying similarities and differences between different models. It can seen as the classification of elements into the four categories presented earlier. Calculation techniques can work with specific types of models, such as done with UML-model comparison methods (Alanen and Porres, 2003; Girschick, 2006; Kelter et al., 2005; Ohst et al., 2003; Xing and Stroulia, 2005). However, it is possible to have meta-model independent comparisons, such as techniques for Domain Specific Models (Lin et al., 2007) and other such approaches (Rivera and Vallecillo, 2008; Selonen and Kettunen, 2007; van den Brand et al., 2010). Kolovos, Di Ruscio, Pierantonio, and Paige (2009) break down calculation methods into four different categories based on how they match corresponding model elements: 1]*static identity-based matching*, which uses persistent and unique identifiers; 2]*signature-based matching*, which is based on an element's uniquely-identifying signature that is calculated from a user-defined function; 3]*similarity-based matching*, which uses the composition of the similarity of an element's features; and 4]*custom language-specific matching algorithms*, which uses matching algorithms designed to work with a particular modeling language. In the remainder of this survey we focus mainly on this phase, as calculation is the most researched and is of the greatest interest to our industrial partners since it will help us with our goal of considering how to discover common sub-patterns among models (Alalfi et al., 2012).

Representation This phase deals with the underlying form that the differences and similarities detected during calculation will take. One approach to representation is the notion of edit scripts (Alanen and Porres, 2003; Mens, 2002). Edit scripts are an operational representation of the changes necessary to make one model equivalent to another and can be comprised of primitive operations such as add, edit, and delete. They tend to be associated with calculation methods that use unique identifiers and may not be user-friendly due to their imperative nature. In contrast, model-based representation (Ohst et al., 2003) is a more declarative approach. It represents differences by recognizing the elements and sub-elements that are the same and those that differ. Most recently, an abstract-syntax-like representation (Brun and Pierantonio, 2008) has been proposed that represents the differences declaratively and enables further analysis. There are properties (Cicchetti et al., 2007; Van den Brand et al., 2010) for representation techniques that separate them from calculation approaches and make them ideal for MDE environments. Cicchetti, Di Ruscio, and Pierantonio (2008) provide a representation that is ideal for managing conflicts in distributed environments.

Visualization Visualization involves displaying the differences in a desirable form to the end user. Visualization is considered somewhat secondary to calculation and representation and may be tied closely to representation. For example, model-based representation can be visualized through colouring (Ohst et al., 2003). Newer visualization approaches try to separate visualization from representation (Van den Brand et al., 2010; Wenzel, 2008; Wenzel et al., 2009). There is also an approach (Schipper et al., 2009) to extend visualization techniques for comparing text files to work with models by including additional features unique to higher level representations, such as folding, automatic layout, and navigation.

2.2 Model Versioning

The need for collaboration amongst teams in MDE projects is equally significant as it is in traditional software projects. Traditional software projects achieve this through Version Control Systems (VCS) such as CVS[1] and Subversion[2]. Similarly, for MDE, it is imperative that modelers are able to work independently but later be able to reintegrate updated versions into the main project repository. Traditional VCS approaches do not work well with models as they are unable to handle model-specific problems

[1] http://www.nongnu.org/cvs/

[2] http://subversion.tigris.org

like the "dangling reference" problem and others (Altmanninger et al., 2009).

Model versioning is broken into different phases by different people (Alanen and Porres, 2003; Altmanninger et al., 2008; Kelter et al., 2005). Generally, it is seen to require some form of model comparison or matching, that is, the identification of what model elements correspond to what other model elements; detection and representation of differences and conflicts; and model merging, that is, combining changes made to corresponding model elements while accounting for conflicts. For the purpose of this survey, we focus mainly on the first phase, model comparison. While the other phases are equally as important, we are interested mostly in the way model versioning approaches achieve model comparison.

2.3 Model Clone Detection

Another example of model comparison being used in a different and specific context is model clone detection. In traditional software projects, a *code clone* refers to collections of code that are similar to one another in some measure of similarity (Koschke, 2006). One common reason that code clones arise in these projects is the implementation of a similar concept throughout the system. The problem with code clones is that a change in this one concept means that the system must be updated in multiple places. Research in code clones is very mature and there are many techniques and tools to deal with them (Roy et al., 2009).

The analogous problem of *model clone*s refers to groups of model elements that are shown to be similar in some defined fashion (Deissenboeck et al., 2009). By comparison with code clone detection, research in model clone detection is quite limited (Deissenboeck et al., 2010), with the majority of approaches thus far tailored for Simulink data-flow models.

3 MODEL COMPARISON APPROACHES AND APPLICATIONS

This section categorizes existing model comparison methods by the specific types of models they compare and discusses the applications of the comparisons. Within each section, we further classify the approaches by the sub type of model they are intended for. While some methods claim they can be extended to work with other types of models, we still place each one in only the category they have demonstrated they work with and make note of the claimed extendability.

3.1 Methods for Multiple Model Types

This section looks at approaches that are able to deal with more than one type of model, such as both structural and behavioral models. Approaches like this can be more general, but can not use information specific to the model type.

3.1.1 UML Models

UML is a meta model defined by the MOF and is the most prominent and well known example of a MOF instance. This section looks at model comparison approaches intended to compare more than one type of UML model.

Alanen and Porres (2003; 2005) perform model comparison as a precursor to model versioning. Their model matching is achieved through static identity-based matching as it relies on the UML's universally unique identifiers(UUID). Their work is focused on identifying differences between matched models. They calculate and represent differences as directed deltas, that is, operations that turn one model into the other and that can be reversed through a dual operation. Their difference calculation is achieved through the three steps that results in a sequence of operations: 1] Given a model V and V', map matched model elements through UUID comparison; 2] Add operations to create the elements within V' that do not exist in V and then add operations to delete the elements that are in V that are not in V'; 3] For all the elements that are in both V and V', any changes that have occurred to the features within these elements from V to V' are converted to operations that ensure that feature ordering is preserved.

Rational Software Architect (RSA) is an IBM product intended to be a complete MDE development environment for software modeling. Early versions of RSA, RSA 6 and earlier, allow for two ways of performing model comparison on UML models, both of which are a form of model versioning: Comparing a model from its local history and comparing a model with another model belonging to the same ancestor (Letkeman, 2005). In both cases, model matching is done using UUIDs. The calculation for finding differences between matched elements is proprietary.

Figure 1 shows an example of the "compare with local history" RSA window. The bottom pane is split between the two versions while the top right pane describes the differences found between the two. The window and process for comparing a model with another model within the same ancestry is very similar. In RSA version 7 and later a facility is provided by the tool to compare and merge two software models that are not necessarily from the same ancestor (Letkeman, 2007). This is accomplished manually through

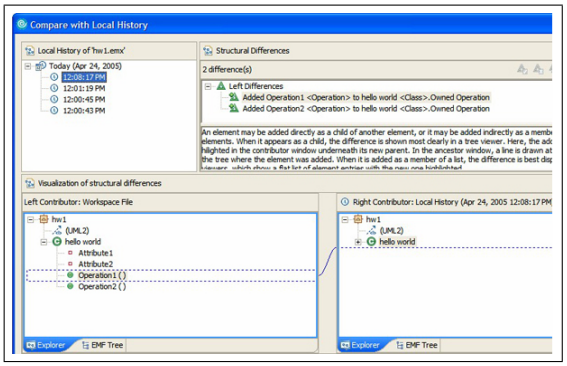

Figure 1: RSA "Compare with Local History" Window, adapted from (Letkeman, 2005).

user interaction, that is, there is no calculation taking place other than the option to do a relatively simple structural and UUID comparison. The user selects elements from the source that should be added to the target, maps matching elements from the source and target, and chooses the final names for these matched elements.

Another example of a technique that compares UML models and utilizes their UUIDs is proposed by Ohst, Welle, and Kelter (2003). This technique transforms UML models to graphs and then traverses each tree level with the purpose of searching for identical UUIDs. The technique takes into account differences among the matched model elements, such as features and relationships.

Selonen and Kettunen (2007) present a technique for model matching that derives signature-match rules based on the abstract syntax of the meta model describing the modeling language. Specifically, they say that two models match if they belong to the same meta class, have the same name and the same primary context, which includes the surrounding structure of the model comprised of neighbours and descendants. They state that this technique can be extended to work with any MOF-based modeling languages. Additional rules can be added by extending the model with appropriate stereotypes that the technique can interpret.

Odyssey VCS (Oliveira et al., 2005) is a model versioning tool intended to work with all types of UML models in CASE environments. It does not perform model matching as all elements are linked to a previous version, starting with a baseline version. Differences or conflicts are detected by processing XML Metadata Interchange (XMI) files and using UML-specific knowledge to calculate what elements have been added, modified, or deleted.

Storrle (2010) developed the *MQlone* tool to experiment with the idea of detecting UML model clones. They convert XMI files from UML CASE models and turn them into Prolog [3]. Once in Prolog, they attempt to discover clones using static identity matching combined with similarity metrics such as size, containment relationships, and name similarity.

3.1.2 EMF Models

Eclipse Modeling Framework (EMF) models [4] are MOF meta-meta models that can define meta models, such as UML. They are intended for use within the Eclipse development environment.

EMFCompare (Brun and Pierantonio, 2008) is an Eclipse project. Rather than relying on EMF models' UUIDs, they use similarity based-matching to allow the tool to be more generic and useful in a variety of situations. The matching calculation is based on various statistics and metrics that are combined to generate a match score. This includes analyzing the name, content, type, and relations of the elements. They also filter out element data that comes from default values. It should be noted that while *EMFCompare* is specific to EMF models, the underlying calculation engine is meta-model independent, similar to the approaches discussed in Section 3.1.3

TopCased (Farail et al., 2006) is a project providing an MDE environment that uses EMF models and is intended specifically for safety critical applications and systems. It performs its matching and differencing using static identity-based matching, similar to Alanen and Porres.

SmoVer (Reiter et al., 2007) is another model versioning tool that can work with any EMF-based model. In this approach they do both version-specific comparisons like Odyssey VCS, termed syntactical, and semantic comparisons. Semantic comparisons are those that are carried out on semantic views. Semantic views, in this context, are the resulting models that come from a user-defined model transformation that SmoVer executes on the original models being compared in order to give the models meaning from a particular view of interest. These transformations are specified in the Atlas Transformation Language (ATL) [5]. Models are first compared syntactically, then transformed, then compared once again. Matching is accomplished through static-identity based matching and differences are calculated by comparing structural changes. The authors note that the amount of work required for creating the ATL transformations is not too large and that "the return on investment gained in better conflict detection clearly outweighs the initial effort spent on specifying the semantic views." The ex-

[3] www.swi-prolog.org
[4] http://www.eclipse.org/emf/
[5] http://eclipse.org/atl/

amples they discuss required roughly 50 to 200 lines of ATL transformation language code.

Riveria and Vallecillo (2008) employ similarity-based matching in addition to checking persistent identifiers for their tool, *Maudeling*. They use *Maude* (Clavel et al., 2002), a high-level language that supports rewriting-logic specification and programming, to facilitate the comparison of models from varying meta models specified in EMF. They provide this in the form of an Eclipse plugin called *Maudeling*. Using Maude, they specify a difference meta model that represents differences as added, modified, or deleted elements. The process first begins by using UUIDs. If there is no UUID match, they then rely on a variant of similarity-based matching that calculates the similarity ratio by checking a variety of structural features. Then, to calculate differences, they go through each element and categorize it based on a number of scenarios and incorporate this information into its difference meta model. There is no user work required as *Maudeling* has the ATL transformations within it that transforms the EMF models into their corresponding Maude representations. Operations are executed in the Maude environment automatically, requiring no user interaction with Maude.

3.1.3 Metamodel-Agnostic Approaches

This section discusses comparison approaches for models that can conform to an arbitrary meta model, assuming it adheres to specific properties.

Examples of Metamodel-independent approaches that use similarity-based matching strategies include the *Epsilon Comparison Language* (Kolovos, 2009) and Domain-Specific Model Diff*DSMDiff* (Lin et al., 2007). *DSMDiff* is an extension of work done on UML model comparison techniques. *DSMDiff* uses both similarity- and signature- based matching. The similarity-based matching focuses on the similarity of edges among different model nodes. *DSMDiff* evaluates differences between matched elements and considers them directed deltas. While *DSMDiff* was developed using DSMLs specified in the Generic Modeling Environment (GME), the techniques can be extended to work with any DSML creation tool. The creators of *DSMDiff* propose allowing user-interaction that will enable one to choose the mappings (matches) from a list of applicable candidates.

Epsilon Comparison Language (ECL) was developed after *DSMDiff* and *SiDiff*, which is discussed later, and attempts to address the fact that its predecessors do not allow for modellers to configure language-specific information that may assist in matching model elements from different meta models (Kolovos, 2009). This is accomplished in a imperative yet high-level manner. ECL allows modelers to specify model comparison rule-based algorithms to identify matched elements within different models. The trade off is that, while complex matching criteria can be expressed using ECL, it requires more time and knowledge of ECL. Kolovos acknowledges that metamodel-independent approaches using similarity-based matching typically perform fairly well, but there often are "corner cases" that ECL is well suited to identify.

A plugin for meta-Case applications was developed (Mehra et al., 2005) that performs model version comparison for models defined by a meta-Case tool. Meta-Case tools operate similarly to their CASE counterparts except they are not constrained by a particular schema or meta model. This plugin matches all the elements by their unique identifiers and calculates the differences as directed deltas. Oda and Saeki (2005) describe a graph-based VCS that is quite similar in that it works with meta-Case models and matches them using baselines and unique identifiers. Differences are calculated as directed deltas with respect to earlier versions.

Nguyen (2006) proposes a VCS that can detect both structural and textual differences between versions of a wide array of software artifacts. This approach utilizes similarity-based matching by assigning all artifacts an identifier that encapsulates the element and representing them as nodes within a directed attributed graph, similar to model clone detection approaches.

Van den Brand, Protic, and Verhoeff(2010) define a set of requirements for difference representations and argue that current meta-modeling techniques, such as MOF, are not able to satisfy them. They present their own meta-modeling technique and define differences with respect to it. They provide a model comparison approach and prototype that allows user configuration of what combination of the four model-matching strategies to employ. The authors provide examples where they extend the work done previously for *SiDiff*, combining it with other matching techniques, like using a UUID. This generality comes at a cost of a large amount of configuration, work, and user-interaction.

There are methods that translate models into another language or notation that maintain the semantics of the models to facilitate model comparison. One example is the work done by Gheyi, Massoni,and Borba (2005) in which they propose an abstract equivalence notion for object models, in other words, a way of representing objects that allows them to be compared. They use an alphabet, which is the set of relevant elements that will be compared, and views,

which are mappings that express the different ways that one element in one model can be interpreted by elements of a different model. Equivalence between two models is defined as the case where, for every interpretation or valid instance that satisfies one model, there exists an equivalent interpretation that satisfies an instance in the other model. They illustrate their approach using Alloy models [6]. Similarly, Maoz, Ringert, and Rumpe (2011b) present the notion of *semantic diff operators*, which represent the relevant semantics of each model, and *diff witnesses*, which are the semantic differences between two models. Semantics are represented through the use of mathematical formalisms. From these ideas, Maoz et al. provide the tools *cddiff* and *addiff* for class diagram differencing and activity class diagram differencing, respectively. Other examples of translating models into another language include UML models being translated into Promela/SPIN models (Chen and Cui, 2004; Latella et al., 1999; Lilius and Paltor, 1999), although this work does not intend to perform model comparison nor differencing explicitly.

The *Query/View/Transform*(QVT) standard provides the outline for three model transformation languages that can act on any arbitrary model or metamodel that conform to the MOF specification. One of these languages, *QVT-Relations*(QVT-R) allows for a declarative specification of two-way (bi-directional) transformations and is more expressive than the other QVT languages. This expressiveness allows for a form of model comparison through its *checkonly mode*, which is the mode where models are checked for consistency rather than making changes (Stevens, 2009). In brief, game theory is applied to QVT-R by using a verifier and refuter. The verifier confirms that the check will succeed and the refuter's objective is to disprove it. The semantics of QVT are expressed in such a way that implies "the check returns true if and only if the verifier has a winning strategy for the game". This allows one to compare models assuming a transformation was expressed in this language that represented the differences or similarities being searched for. In this case, the transformation will yield true if the pair/set of models are consistent according to the transformation.

3.2 Methods for Behavior/Data-Flow Models

3.2.1 Simulink and Matlab Models

CloneDetective (Deissenboeck et al., 2009) is an approach that uses ideas from graph theory and is applicable to any model that is represented as a data-flow

[6] http://alloy.mit.edu

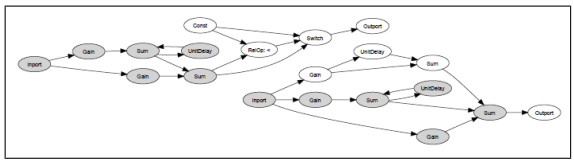

Figure 2: Normalised model graph of model clone example (Deissenboeck et al., 2009).

graph. Models are first flattened and unconnected lines are removed. Subsequently, they are normalised by taking all of the blocks and lines found within the models and assigning them a label that consists of essential information for comparison. The information described in the label changes according to the type of clones being searched for by the tool. Figure 2 displays the result of this step on a Simulink model, including the labels that come from normalisation, such as **UnitDelay** and **RelOp:**⟨. The grey portions within the graph represent a clone. This is accomplished by *CloneDetective* in its second phase: clone pair extraction. This phase is the most similar to the model matching discussed thus far, however, rather than matching single elements, it is attempting to match the largest common set of elements. It falls into the category of similarity-based matching as the features of each element are extracted, represented as a label during normalisation, and compared.

Similarly, *eScan* and *aScan* algorithms attempt to detect exact-matched and approximate clones, respectively (Pham et al., 2009). Exact-matched clones are groups of model elements having the same size and aggregated labels, which contain topology information and edge and node label information. Approximate clones are those that are not exactly matching but fit some similarity criteria. *aScan* uses vector-based representations of graphs that account for a subset of structural features within the graph. The main difference between these algorithms and *CloneDetective* is that these algorithms group their clones first and from smallest to largest. They claim that this helps detect clones that *CloneDetective* can not. This is later refuted, however, by the authors of *CloneDetective* (Deissenboeck et al., 2010). *aScan* is able to detect approximate clones while *CloneDetective* is not. Much like *CloneDetective*, these algorithms utilize similarity-based matching.

Al-Batran, Schatz, and Hummel (2011) note that existing approaches deal with syntactic clones only, that is they can detect only syntactically and structural similar copies. Using normalization techniques that utilize graph transformations, they extend these approaches to cover semantic clones that may have similar behavior but different structure. So, a clone in this context is now defined as two (sub)sets of models that have "equivalent unique normal forms" of mod-

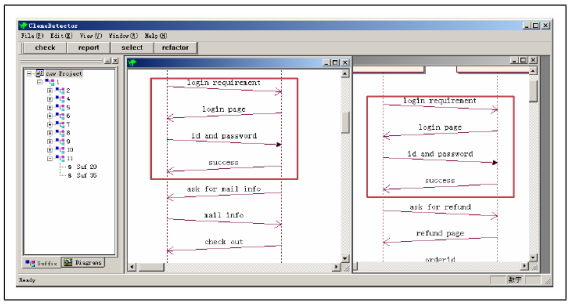

Figure 3: Duplicate sequence fragment (Liu et al., 2007).

els. These unique normal forms are acquired by performing 40 semantic-preserving transformations on Simulink models. It was found that extending clone detection in this way yields more clones than simple syntactic comparison.

Most recently, Alalfi, Cordy, Dean, Stephan, and Stevenson (2012) developed *Simone*, which detects near-miss clones in Simulink models. This is done by modifying existing code-clone techniques to work with the textual representations of the Simulink models while still being model-sensitive. In comparison to *CloneDetective*, they detect the same exact clones and some additional near-miss ones.

3.2.2 Sequence Diagrams

Liu, Ma, Zhang, and Shao (Liu et al., 2007) discover duplication in sequence diagrams. They convert sequence diagrams into an array and represent that array as a suffix tree. This tree is traversed and duplicates are extracted by looking for the longest common prefix, or elements that lead to the leaf node, of two suffixes. Duplicates are defined as a set of sequence-diagram fragments that contain the same elements and have the same sequence-diagram specific relationships. Figure 3 demonstrates an example of a duplicate. In this case, the largest common prefix include the four elements highlighted within each diagram. The image is taken from their tool *DuplicationDetector*. Much like the model clone detection approaches discussed, this technique employs a variation of similarity-based matching as it is comparing a graph representation of a fragment's features/elements.

3.2.3 Statechart Diagrams

Nejati et al. (2007) match state chart diagrams for the purpose of model merging. They accomplish this by using heuristics that include looking at terminological, structural, and semantic similarities between models. The heuristics are split into 2 categories: static heuristics that use attributes without semantics, such as the names or features of elements; and behavioural heuristics, which find pairs that have similar dynamic behavior. Due to the use of heuristics, this approach requires a domain expert look over the relations and add or remove relations, accordingly, to acquire the desired matching relation. This approach employs both similarity-based matching through static heuristics and custom-language specific matching through dynamic heuristics. After similarity is established between two state charts, the merging operation calculates and represents differences as variabilities, or guarded transitions, from one model to the other.

3.3 Methods for Structural Models

This section discusses approaches that are designed to work with models that represent the structure of a system. They benefit from the domain knowledge gained by working with structural models only.

Early work on comparing and differencing software structural diagrams was done by Rho and Wu (1998). They focus on comparing a current version of an artifact to its preceding ancestor.

3.3.1 UML Structural Models

UMLDiff (Xing and Stroulia, 2005) uses custom language-specific matching by using name-similarity and UML structure-similarity to identify matching elements. These metrics are combined and compared against a user-defined threshold. It is intended to be a model versioning reasoner: It discovers changes made from one version of a model to another.

UMLDiff$_{cld}$ (Girschick, 2006) focuses on UML class diagram differencing. It uses a combination of static identity-based and similarity-based matching within its evaluation function, which measures the quality of a match. Similarly, *Mirador* (Barrett et al., 2010) is a plugin created for the Fujaba (From Uml to Java and Back Again)[7] tool suite that allows for user directed matching of elements. Specifically, users can select match candidates that are ranked according to a similarity measure that is a combination of static identity-based and similarity-based matching, like *UMLDiff*$_{cld}$.

Reddy et al. (2005) use signature-based matching in order to compare and compose two UML class models in order to assist with Aspect-oriented modeling (Elrad et al., 2002). Models are matched based on their signatures, or property values associated with the class. Each signature has a signature type, which is the set of properties that a signature can take. Using KerMeta[8], a model querying language, the signatures used for comparison are derived by the tool based on the features it has within the meta model.

[7] http://www.fujaba.de/

[8] http://www.kermeta.org

Berardi, Calvanese, and De Giacomo (2005) translate UML class diagrams into ALCQI, a "simple" description logic representation. They show that it is possible for one to reason about UML class diagrams as ALCQI description logic representations and provide an encoding from UML class diagrams to ALCQI. While the translation does not maintain the entire semantics of the UML classes, it preserves enough of it to check for class equivalence. Because they use UML-specific semantics, we argue that theirs is a form of language-specific matching.

Maoz, Ringert, and Rumpe (2011a) extend work on semantic differencing and provide a translation prototype, called *CD2Alloy*, that converts UML classes into Alloy. This Alloy code contains constructs that determine the corresponding elements of two UML class diagrams and also allows for semantic comparisons, such as determining if one model is a refinement of another. It can be considered to be a custom-language specific comparison because of its use of UML semantics.

3.3.2 Metamodel-Agnostic Approaches

Preliminary work on model comparison was done by Chawathe, Rajaraman, Garcia-Molina, and Widom (1996) in which they devised a comparison approach intended for any structured document. They convert the data representing the document structure into a graph consisting of nodes that have identifiers derived from the corresponding elements they represent. This approach, which is analogous to the model clone detection techniques, uses similarity-based matching and describes differences in terms of directed deltas.

SiDiff (Kelter et al., 2005) is very similar to *UMLDiff* except *SiDiff* uses a simplified underlying comparison model in order to be able to handle any models stored in XMI format. Similarly to *UMLDiff*, it uses similarity-based metrics. In contrast to *UMLDiff*, as shown by the up arrow in Figure 4, its calculation begins bottom-up by comparing the sub elements of a pair of model elements starting with their leaf elements. This is done with respect to the elements' similarity metrics. An example of a weighted similarity is having a class element consider the similarity of its class name weighted the highest. So, in the case of the two **Class** elements being compared in Figure 4, all of the **Classifier** elements are compared first. If a uniquely identifying element is matched, such as a class name, they are immediately identified as a match. This is followed by top-down propagation of this matching pair. This top-down approach allows for the algorithm to deduce differences by evaluating a correspondence table that is the output of the matching phase.

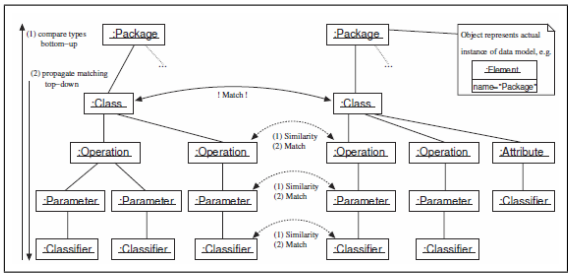

Figure 4: Example SiDiff comparison (Kelter et al., 2005).

Similarly to the translation of UML class diagrams into ALCQI, D'Amato; Staab;and Fanizzi (2008), propose a comparison measure for description logics, such as those used in the Semantic Web [9]. This is accomplished by using existing ontology semantics. They describe a semantic similarity measure that is able use the semantics of the ontology that the concepts refer to.

3.4 Methods for Product Line Architectures

Chen, Critchlow, Garg, Van der Westhuizen, and Van der Hoek (Chen et al., 2004) present work to do comparisons of product line models for merging. The assumption in this work is that the comparison is being done between two versions of the same artifact. Comparison is done recursively and is increasingly fine grained as the algorithm delves deeper into the product-line hierarchy. This approach employs similarity-based matching: lower elements in the hierarchy compare interfaces, optionality, and type; and higher level elements compare the elements contained within them. Differences are represented as directed deltas.

Rubin and Chechik (2012) devise a framework for comparing individual products that allows for them to be updated automatically to a product line conforming to those used in product-line engineering. They use similarity-based matching, that is, products are viewed as model elements and a match is defined as the case where a pair of model elements have features that are similar enough to be above a defined weighted threshold. The authors note that "(their) refactoring framework is applicable to a variety of model types, such as UML, EMF or Matlab/Simulink, and to different compare, match and merge operators". The foreseeable user work that would be required in using this framework includes having a domain expert specify the similarity weights and features of interest and also determining the optimum similarity thresholds.

[9] http://semanticweb.org/

3.5 Methods for Process Models

Soto and Munch (2006) discuss the need for ascertaining differences among software development process models and outline what such a difference system would require. They devise *Delta-P* (Soto, 2007), which can work with various UML process models. *Delta-P* converts process models into Resource Description Framework (RDF) [10] notation in order to have them represented in a normalized triplebased notation. It then performs an identity-based comparison and calculates differences. They employ static-identity based matching as they use unique identifiers. Differences are represented as directed deltas, which can be grouped together to form higher level deltas.

Similarly, Dijkman, Dumas, Van Dongen, Karik, and Mendling(Dijkman et al., 2011) discuss three similarity metrics that help compare stored process models: node matching similarity, which compares the labels and attributes attached to process model elements; structural similarity, which evaluates labels and topology; and, behavioral similarity, which looks at labels in conjunction with causal relations from the process models.

4 SUMMARY AND FUTURE DIRECTIONS

Table 1 summarizes the approaches discussed in this survey paper organized by the type and sub type of model they can compare. As seen in the table, similarity-based matching is the most commonly employed strategy. It is clear that one future direction of work in this area is the focus on tools that are able to work with models that conform to an arbitrary meta model. This result is consistent with the recent trend in domain-specific modeling.

The majority of work in model comparison appears to be geared towards model versioning, however the most recent work is not. Much of the recent work is focusing on model transformation testing and model clone detection. Also, new extensions of existing model comparison approaches are being attempted such as the extension of model clone detection in order to detect common sub-structures or patterns within models (Stephan et al., 2012). These patterns can ideally be used by project engineers to facilitate analysis and assist in the development of future MDE projects.

The last column showcases the relative amount of user work required to accomplish model comparison. This is based on our research performed for this survey and is purely qualitative. The justification for each tool that has one * or more can be found in the textual descriptions provided previously. ECL and the approach presented by Van den Brand et al. require the most user work, however this is intentional to allow for more power and generality. Many approaches require no user interaction as they operate under specific conditions or are dynamic enough to understand the context or meta models they are working with.

[10] http://www.w3.org/RDF/

Table 1: Summary of Model Comparison Approaches.

Types of models	Subtype of model	Specific approach/ tool	Matching strategy+	Primary use#	WR4C~
Multiple types of models	UML Models	Alanan	SI	MV	-
		RSA (Same/Different Ancestor)	SI	MV and MM	-/**
		Ohst	SI	MV	-
		Odyssey-VCS	-	MV	**
		Selonen	SIG	MM	*
	EMF Models	EMF Compare	SIM	MV	-
		TopCased	SI	MV	-
		SMoVer	SI	MV	**
		Maudeling	SI and SIM	MV	-
	Metamodel-Agnostic (Independent)	ECL	SIM	MM and MTT	***
		DSMDiff	SIM	MTT and MV	*
		Mehra-MetaCase	SI	MV	-
		Van den Brand	SI, SIM, SIG,CLS	MV and MM	***
		Nguyen	SIM	MV	-
		QVT-R	SI	MTS	**
Behavioural or Data-Flow models	Mathlab/Simulink	CloneDetective	SIM	MCD	-
		eScan / aScan	SIM	MCD	-
		Simone	SIM	MCD	-
	UML Sequence Diagram	Liu	SIM	MCD/VER	-
	Statecharts	Nejati	SIM and CLS	MM	*
Structural models	UML Models	UMLDiff	CLS	MV	-
		UMLDiff$_{cld}$	SI and SIM	MV	-
		Reddy	SIG	AOM	-
		Mirador	SIM	MV and MM	**
		CD2Alloy	CLS	GC	-
		Convert to ALCQI	CLS	GC	-
	Metamodel-Agnostic (Independent)	Chawathe	SIM	MV	-
		SiDiff	SIM	MV	**
Product Line architectures	Any PLA	Chen	SIM	MV	-
		Rubin	SIM	MM	*
Process models	Software Development Process Models	Delta-P	SI	MV	-

Legend
+ - SI = Static-identity based, SIM=Similarity based, SIG=Signature based, CLS=Custom-language specific
# - MV=Model Versioning, MM=Model Merging AOM= Aspect-Oriented Modeling, VER=Verification MTT=Model Transformation Testing , MTS=Model Transformation Specification, GC=General Comparison
~ - Work required for comparison (WR4C) Scale:
- = Very usable. No user work required for comparison. * = Very little work required.
** = Some work required. *** = Difficult to use. Inordinate level of user work required.

5 RELATED WORK

5.1 Other Comparison Approaches

There is an abundance of work in comparing software artifacts at the code level: However, there are many arguments as to why these are not easily transferable to the modeling domain (Altmanninger et al., 2009).

There are approaches that serialize models to text, for example, XML document difference detection (Cobena et al., 2002), schema-based matching (Shvaiko and Euzenat, 2005), and tools like *Xlinkit* (Nentwich et al., 2003), which compares XMI representations of models. These approaches focus on too low a level of abstraction in that they can not account for model-specific features such as inheritance and cross referencing (Kolovos et al., 2006).

As discussed earlier, code clone detection techniques (Roy et al., 2009) have difficulty and are unable to account for model-specific relations since they are strictly textual. Also, the notion of a code clone and model clone are quite different things.

Similarity Flooding (Melnik et al., 2002) is an example of a labelled graph matching algorithm. It can be seen as a similarity-based matching approach in that it uses the similarity of an element's neighbouring nodes to discover matching elements. If two node's neighbours are similar, then their similarity measure increases. The problem with this approach is that it works on too high of a conceptual level and is not able to use diagram- or context- specific information (Fortsch and Westfechtel, 2007). A similar problem is experienced by the method presented by Krinke (2002) in which similar code is discovered through program dependency graph analysis.

5.2 Related Survey Papers

This section discusses other reviews that overlap with the model comparison review of this survey.

Altmanninger, Seidl, and Wimmer (2009) perform a survey on model versioning approaches. This survey differs from the work done in ours in that they focus very little on matching, investigate versioning systems only, and discuss only a subset of the model comparison approaches that we do. They are much more concerned with merging than comparison.

Sebastiani and Supiratana (2008) provide a summary of various differencing approaches, however, they look at only three specific approaches at a high level and compare them: *SiDiff*, *UMLDiff*, *DSMDiff*. This is done with a focus on how these techniques can trace a model's evolution.

Selonen (2007) performs a brief review of UML model comparison approaches. It looks at five specific approaches, all of which we cover. Similar to this survey, they also note the various usage scenarios or applications of the approaches. In contrast, our survey looked at techniques that deal with various types of models and included additional UML approaches not identified by Selonen.

Finally, this paper is an updated and condensed version of our own technical report surveying this area (Stephan and Cordy, 2011).

6 CONCLUSION

Model comparison is a relatively young research area that is very important to MDE. It has been implemented in various forms and for various purposes, predominantly in model versioning, merging, and clone detection.

We have provided an overview of the area, and have observed that the majority of recent approaches allow for models belonging to arbitrary meta-models. Similarity-based matching is the approach taken by most methods. Model versioning appears to be the most common goal for model comparison up to this point, but that is starting to shift. Lastly, some approaches require more user effort to perform model comparison, however this is to facilitate flexibility and strength. Many of the approaches require no user interaction because they are intentionally constrained or are made to deal with multiple situations.

There is still much room for maturity in model comparison and it is an important area that must be in the minds of MDE supporters, as it has many benefits and is widely-applicable. It is our hope that this survey paper of model comparison acts as a reference guide for both model-driven engineers and researchers.

REFERENCES

Al-Batran, B., Schatz, B., and Hummel, B. (2011). Semantic clone detection for model-based development of embedded systems. *Model Driven Engineering Languages and Systems*, pages 258–272.

Alalfi, M. H., Cordy, J. R., Dean, T. R., Stephan, M., and Stevenson, A. (2012). Models are code too: Near-miss clone detection for simulink models. In *ICSM*, volume 12.

Alanen, M. and Porres, I. (2003). Difference and union of models. In *UML*, pages 2–17.

Alanen, M. and Porres, I. (2005). Version control of software models. *Advances in UML and XML-Based Software Evolution*, pages 47–70.

Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Seidl, M., Schwinger, W., and Wimmer, M. (2008). AMOR-towards adaptable model versioning. In *MCCM*, volume 8, pages 4–50.

Altmanninger, K., Seidl, M., and Wimmer, M. (2009). A survey on model versioning approaches. *International Journal of Web Information Systems*, 5(3):271–304.

Barrett, S., Butler, G., and Chalin, P. (2010). Mirador: a synthesis of model matching strategies. In *IWMCP*, pages 2–10.

Berardi, D., Calvanese, D., and De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1-2):70–118.

Brun, C. and Pierantonio, A. (2008). Model differences in the Eclipse modelling framework. *The European Journal for the Informatics Professional*, pages 29–34.

Chawathe, S., Rajaraman, A., Garcia-Molina, H., and Widom, J. (1996). Change detection in hierarchically structured information. In *ICMD*, pages 493–504.

Chen, J. and Cui, H. (2004). Translation from adapted UML to promela for corba-based applications. *Model Checking Software*, pages 234–251.

Chen, P., Critchlow, M., Garg, A., Van der Westhuizen, C., and van der Hoek, A. (2004). Differencing and merging within an evolving product line architecture. *PFE*, pages 269–281.

Cicchetti, A., Di Ruscio, D., and Pierantonio, A. (2007). A metamodel independent approach to difference representation. *Technology*, 6(9):165–185.

Cicchetti, A., Di Ruscio, D., and Pierantonio, A. (2008). Managing model conflicts in distributed development. In *Models*, pages 311–325.

Clavel, M., Duran, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J., and Quesada, J. (2002). Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243.

Cobena, G., Abiteboul, S., and Marian, A. (2002). Detecting changes in XML documents. In *ICDE*, pages 41–52.

D'Amato, C., Staab, S., and Fanizzi, N. (2008). On the influence of description logics ontologies on conceptual similarity. *Knowledge Engineering: Practice and Patterns*, pages 48–63.

Deissenboeck, F., Hummel, B., Juergens, E., Pfaehler, M., and Schaetz, B. (2010). Model clone detection in practice. In *IWSC*, pages 57–64.

Deissenboeck, F., Hummel, B., Jurgens, E., Schatz, B., Wagner, S., Girard, J., and Teuchert, S. (2009). Clone detection in automotive model-based development. In *ICSE*, pages 603–612.

Dijkman, R., Dumas, M., Van Dongen, B., Karik, R., and Mendling, J. (2011). Similarity of business process models: Metrics and evaluation. *Information Systems*, 36(2):498–516.

Elrad, T., Aldawud, O., and Bader, A. (2002). Aspect-oriented modeling: Bridging the gap between implementation and design. In *Generative Programming and Component Engineering*, pages 189–201.

Farail, P., Gaufillet, P., Canals, A., Le Camus, C., Sciamma, D., Michel, P., Cregut, X., and Pantel, M. (2006). The topcased project: a toolkit in open source for critical aeronautic systems design. *ERTS*, pages 1–8, electronic.

Fortsch, S. and Westfechtel, B. (2007). Differencing and merging of software diagrams state of the art and challenges. *ICSDT*, pages 1–66.

Gheyi, R., Massoni, T., and Borba, P. (2005). An abstract equivalence notion for object models. *Electronic Notes in Theoretical Computer Science*, 130:3–21.

Girschick, M. (2006). Difference detection and visualization in UML class diagrams. *Technical University of Darmstadt Technical Report TUD-CS-2006-5*, pages 1–15.

Kelter, U., Wehren, J., and Niere, J. (2005). A generic difference algorithm for UML models. *Software Engineering*, 64:105–116.

Kolovos, D. (2009). Establishing correspondences between models with the epsilon comparison language. In *Model Driven Architecture-Foundations and Applications*, pages 146–157.

Kolovos, D., Di Ruscio, D., Pierantonio, A., and Paige, R. (2009). Different models for model matching: An analysis of approaches to support model differencing. In *CVSM*, pages 1–6.

Kolovos, D., Paige, R., and Polack, F. (2006). Model comparison: a foundation for model composition and model transformation testing. In *IWGIMM*, pages 13–20.

Koschke, R. (2006). Survey of research on software clones. *Duplication, Redundancy, and Similarity in Software*, pages 1–24,electronic.

Krinke, J. (2002). Identifying similar code with program dependence graphs. In *WCRE*, pages 301–309.

Latella, D., Majzik, I., and Massink, M. (1999). Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, 11(6):637–664.

Letkeman, K. (2005). Comparing and merging UML models in IBM Rational Software Architect. *IBM Rational - Technical Report (Online)*.

Letkeman, K. (2007). Comparing and merging UML models in IBM Rational Software Architect: Part 7. `http://www.ibm.com/developerworks/rational/library/07/0410_letkeman/`.

Lilius, J. and Paltor, I. (1999). Formalising UML state machines for model checking. *UML*, pages 430–444.

Lin, Y., Gray, J., and Jouault, F. (2007). DSMDiff: a differentiation tool for domain-specific models. *European Journal of Information Systems*, 16(4):349–361.

Lin, Y., Zhang, J., and Gray, J. (2004). Model comparison: A key challenge for transformation testing and version control in model driven software development. In *OOPSLA Workshop on Best Practices for Model-Driven Software Development*, volume 108, pages 6, electronic.

Liu, H., Ma, Z., Zhang, L., and Shao, W. (2007). Detecting duplications in sequence diagrams based on suffix trees. In *APSEC*, pages 269–276.

Maoz, S., Ringert, J., and Rumpe, B. (2011a). Cd2alloy: Class diagrams analysis using alloy revisited. *Model Driven Engineering Languages and Systems*, pages 592–607.

Maoz, S., Ringert, J., and Rumpe, B. (2011b). A manifesto for semantic model differencing. In *ICMSE*, MODELS'10, pages 194–203.

Mehra, A., Grundy, J., and Hosking, J. (2005). A generic approach to supporting diagram differencing and merging for collaborative design. In *ASE*, pages 204–213.

Melnik, S., Garcia-Molina, H., and Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128.

Mens, T. (2002). A state-of-the-art survey on software merging. *TSE*, 28(5):449–462.

Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., and Zave, P. (2007). Matching and merging of statecharts specifications. In *ICSE*, pages 54–64.

Nentwich, C., Emmerich, W., Finkelstein, A., and Ellmer, E. (2003). Flexible consistency checking. *TOSEM*, 12(1):28–63.

Nguyen, T. (2006). A novel structure-oriented difference approach for software artifacts. In *CSAC*, volume 1, pages 197–204.

Oda, T. and Saeki, M. (2005). Generative technique of version control systems for software diagrams. In *ICSM*, pages 515–524.

Ohst, D., Welle, M., and Kelter, U. (2003). Differences between versions of UML diagrams. *ACM SIGSOFT Software Engineering Notes*, 28(5):227–236.

Oliveira, H., Murta, L., and Werner, C. (2005). Odyssey-vcs: a flexible version control system for UML model elements. In *SCM*, pages 1–16.

Pham, N., Nguyen, H., Nguyen, T., Al-Kofahi, J., and Nguyen, T. (2009). Complete and accurate clone detection in graph-based models. In *ICSE*, pages 276–286.

Reddy, R., France, R., Ghosh, S., Fleurey, F., and Baudry, B. (2005). *Model Composition - A Signature-Based Approach*.

Reiter, T., Altmanninger, K., Bergmayr, A., Schwinger, W., and Kotsis, G. (2007). Models in conflict-detection of semantic conflicts in model-based development. In *MDEIS*, pages 29–40.

Rho, J. and Wu, C. (1998). An efficient version model of software diagrams. In *APSEC*, pages 236–243.

Rivera, J. and Vallecillo, A. (2008). Representing and operating with model differences. *Objects, Components, Models and Patterns*, pages 141–160.

Roy, C., Cordy, J., and Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470–495.

Rubin, J. and Chechik, M. (2012). Combining related products into product lines. In *15th International Conference on Fundamental Approaches to Software Engineering*. To Appear.

Schipper, A., Fuhrmann, H., and von Hanxleden, R. (2009). Visual comparison of graphical models. In *ICECCS*, pages 335–340.

Sebastiani, M. and Supiratana, P. (2008). Tracing the differences on an evolving software model. `http://www.idt.mdh.se/kurser/ct3340/archives/ht08/papersRM08/28.pdf`.

Selonen, P. (2007). A Review of UML Model Comparison Approaches. In *NW-MoDE*, pages 37–51.

Selonen, P. and Kettunen, M. (2007). Metamodel-based inference of inter-model correspondence. In *ECSMR*, pages 71–80.

Shvaiko, P. and Euzenat, J. (2005). A survey of schema-based matching approaches. *Journal on Data Semantics IV*, pages 146–171.

Soto, M. (2007). Delta-p: Model comparison using semantic web standards. *Softwaretechnik-Trends*, 2:27–31.

Soto, M. and Munch, J. (2006). Process model difference analysis for supporting process evolution. *Software Process Improvement*, pages 123–134.

Stephan, M., Alafi, M., Stevenson, A., and Cordy, J. (2012). Towards qualitative comparison of simulink model clone detection approaches. In *IWSC*, pages 84–85.

Stephan, M. and Cordy, J. R. (2011). A survey of methods and applications of model comparison. Technical report, Queen's University. TR. 2011-582 Rev. 2.

Stephan, M. and Cordy, J. R. (2013). Application of model comparison techniques to model transformation testing. In *MODELSWARD*. to appear.

Stevens, P. (2009). A simple game-theoretic approach to checkonly qvt relations. *Theory and Practice of Model Transformations*, pages 165–180.

Storrle, H. (2010). Towards clone detection in uml domain models. In *ECSA*, pages 285–293.

van den Brand, M., Protic, Z., and Verhoeff, T. (2010). Fine-grained metamodel-assisted model comparison. In *IWMCP*, pages 11–20.

Van den Brand, M., Protic, Z., and Verhoeff, T. (2010). Generic tool for visualization of model differences. In *IWMCP*, pages 66–75.

Wenzel, S. (2008). Scalable visualization of model differences. In *CVSM*, pages 41–46.

Wenzel, S., Koch, J., Kelter, U., and Kolb, A. (2009). Evolution analysis with animated and 3D-visualizations. In *ICSM*, pages 475–478.

Xing, Z. and Stroulia, E. (2005). UMLDiff: an algorithm for object-oriented design differencing. In *ASE*, pages 54–65.