A Framework for Relating Syntactic and Semantic Model Differences

Shahar Maoz and Jan Oliver Ringert

School of Computer Science
Tel Aviv University, Israel

***Abstract*—Model differencing is an important activity in model-based development processes. Differences need to be detected, analyzed, and understood to evolve systems and explore alternatives.**

Two distinct approaches have been studied in the literature: syntactic differencing, which compares the concrete or abstract syntax of models, and semantic differencing, which compares models in terms of their meaning. Syntactic differencing identifies change operations that transform the syntactical representation of one model to the syntactical representation of the other. However, it does not explain their impact on the meaning of the model. Semantic model differencing is independent of syntactic changes and presents differences as elements in the semantics of one model but not the other. However, it does not reveal the syntactic changes causing these semantic differences.

We define a language independent, abstract framework, which relates syntactic change operations and semantic difference witnesses. We formalize fundamental relations of *necessary* and *sufficient* sets of change operations and analyze their properties. We further demonstrate concrete instances of the framework for three different popular modeling languages, namely, class diagrams, activity diagrams, and feature models. The framework provides a novel foundation for combining syntactic and semantic differencing.

I. Introduction

Model differencing is an important activity in model-based development processes [2], [3]. Differences between model versions need to be detected, analyzed, and understood by engineers in order to evolve systems and explore design alternatives.

Two distinct approaches to model differencing have been studied in the literature: syntactic differencing, e.g., [2], [5], [14], [15], [17], [27], which compares the concrete or abstract syntax of models in terms of additions, deletions, and modifications of elements (whether reverse engineered or recorded during editing), and semantic differencing, e.g., [1], [8], [9], [18], [19], [21], [22], [28], which compares models in terms of their meaning, e.g., instances or executions allowed only by one and not the other.

Syntactic differencing identifies (partially ordered) sets of change operations that transform the syntactical representation of one model into the syntactical representation of the other. However, these do not explain the impact of the changes on the model in terms of elements removed or added to its semantics. Models with very different syntactic representations may have the same semantics; models with only few syntactic changes between them may have very different semantics. A syntactic comparison is unable to capture these differences.

Semantic model differencing is independent of syntactic changes and presents differences as elements in the semantics of one model but not the other [22]. Example elements in the semantics of models are the object models allowed by a class diagram (CD) [21], the traces of action executions possible for an activity diagram (AD) [19], or the product configurations defined by a feature model (FM) [1], [9]. The elements in the semantics of one model but not the other are called diff witnesses. However, such witnesses do not reveal the causes for the semantic difference in terms of the syntactic change operations that the engineer has applied. Awareness of these causes is important in most model evolution activities, e.g., differencing and merging, which center around syntactic change operations [3].

Despite the achievements of both, the syntactic differencing approach and the semantic differencing approach, a framework that relates them has not yet been suggested. Thus, no previous work provides the means to reason about the relations between syntactic changes on the level of individual change operations and diff witnesses.

Our objective is to support understanding the impact of change operations on the semantics of models and tracing a diff witness to syntactic changes. On the one hand we aim to answer questions raised during the analysis of a syntactic difference given as change operations:

- Which diff witnesses are guaranteed when applying this set of change operations?
- Which diff witnesses will not appear when applying these change operations?

Given two ADs, a concrete example question is: *Which traces of actions are guaranteed to remain in the semantic difference of the two ADs when applying the change that removes node* `CloseClaim`*?*

On the other hand we want to provide answers also to questions raised during the inspection of diff witnesses resulting from a semantic comparison of two models:

- Which syntactic change operations guarantee this diff witness?
- Which syntactic change operations should be omitted to avoid this witness?

J. O. Ringert acknowledges support from a postdoctoral Minerva Fellowship, funded by the German Federal Ministry for Education and Research.

978-1-4673-6908-4/15 © 2015 IEEE

Given two CDs, a concrete example question is: *Which change operations should be omitted to avoid an object of class* `Manager` *managing itself?*

In this paper we define a language independent, abstract framework, which relates change operations and semantic diff witnesses. We formalize the relations of *necessary* change operations that a witness depends on and of *sufficient* change operations whose application guarantees the existence of a diff witness. Both relations relate sets of changes to diff witnesses. As the set of diff witnesses is in many cases very large or even infinite a concrete representation of the relations is not always possible. We sketch algorithms to compute necessary and sufficient sets for a given diff witness and discuss sampling diff witnesses of the inverse relation.

We further instantiate and apply the abstract framework to three concrete modeling languages, activity diagrams, class diagrams, and feature models, with existing, previously defined and published families of change operations from [17], [25], [28] on the one hand, and semantic differencing operators from [1], [18], [19], [21] on the other hand. The languages used in these applications are very different in terms of their syntax, semantics, and kinds of change operations, and so demonstrate the broad applicability of the framework. Thus, the framework provides a general foundation for combining syntactic and semantic differencing.

We organize the remainder of this paper as follows. First, in the next section we describe three examples of changes to ADs, CDs, and FMs and their analysis. Section III presents formal background and foundations of our work. We relate syntactic and semantic differences in Sect. IV as the main contribution of our work. Section V shows instantiations of our framework for concrete languages and difference formalizations. We discuss challenges and limitations in Sect. VI and related work in Sect. VII. Section VIII concludes.

II. Examples

We start with semi-formal examples of syntactic differencing, semantic differencing, and their combinations, applied to activity diagrams, class diagrams, and feature models.

A. Insurance Company Activity Change

We adapt an example from [16] of an insurance company that changes its workflow for handling insurance claims. The ADs ad_1 before and ad_2 after the change are shown in Fig. 1. A syntactic analysis of the two versions reveals a set of change operations, e.g., inserting the merge and decision nodes `M1` and `D1`, inserting the action node labeled `RetrieveAddData`, deleting the action node labeled `CloseClaim` etc.

After performing the changes the engineer is interested in the effect of inserting action node `RetrieveAddData`. An analysis based on our framework reveals that all executions tr_1 = <RecordClaim, CheckClaim, RetrieveAddData, CheckClaim, ..> starting with recording and checking the claim and then retrieving additional data and checking again are semantic diff witnesses that this

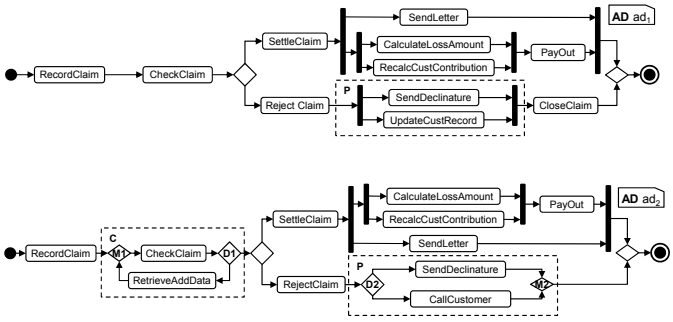

Fig. 1. Two versions of an activity for handling insurance claims (adapted from [16]).

operation is necessary for. They cannot exist without it. These execution traces are only possible in ad_2 and not in ad_1.

Later, another engineer observes an execution of the activity where the claim is recorded, checked, then rejected and a declinature is sent but the claim is not closed. This execution is a new behavior only possible in ad_2 and not in ad_1 (as is found by our ADDiff tool [19]). An analysis based on our new framework reveals that the trace tr_4 = <RecordClaim, CheckClaim, RejectClaim, SendDeclinature> is an effect of the necessary and sufficient pair of change operations deleting action nodes labeled `UpdateCustRecord` and `CloseClaim`.

The above two analyses, enabled by our framework, are not supported by any previous work.

B. Class Model Evolution in Graphical Modeling Framework

The Graphical Modeling Framework (GMF) is an Eclipse framework for creating graphical editors[1]. Its meta-models are formalized as Ecore models. Fig. 2 shows excerpts from two consecutive versions of GMF Ecore models as CDs cd_1 and cd_2. Examples for changes from cd_1 to cd_2 are the addition of the abstract class `CustomAttributeOwner` as a superclass of `RealFigure` and `CustomClass`, the move of the composition between `CustomClass` and `CustomAttribute` to be owned by the new class `CustomAttributeOwner`, and the addition of the class `CenterLayout`.

During semantic differencing of the two CDs, e.g., using our CDDiff tool [21], an engineer wonders about an `Ellipse` owning instances of `CustomAttribute` (see Fig. 3, om_1), which was not possible in cd_1. She investigates the change using our framework and finds that the addition of the class `CustomAttributeOwner`, the inheritance to `RealFigure`, and the moving of the composition with `CustomAttribute`, are sufficient changes for this witness, i.e., independent of other changes these guarantee the existence of the witness.

Another engineer uses our framework to check whether the addition of class `CenterLayout` is necessary for any semantic difference. The change indeed allows new instances of cd_2

[1]Eclipse Graphical Modeling Project http://www.eclipse.org/modeling/gmp/.

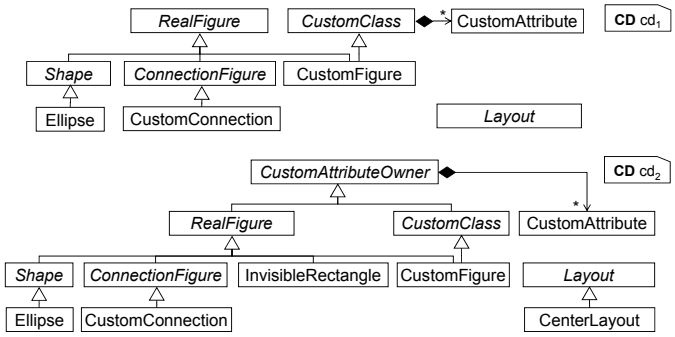

Fig. 2. Excerpts of two consecutive versions of the Eclipse GMFgraph metamodel cd_1 committed 2012-04-29 and cd_2 committed 2012-05-14.

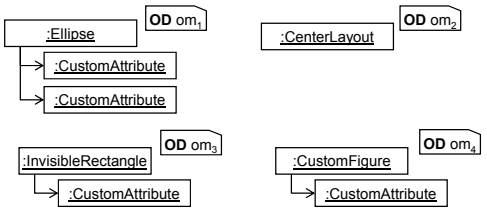

Fig. 3. Object model instances of class diagram cd_2 depicted as object diagrams.

not possible in cd_1 with objects of type `CenterLayout` (see Fig. 3, om_2). The change is necessary and by itself already sufficient for the witness om_2.

Again, the above analyses, enabled by our new framework, are not supported by any previous work.

C. Car Feature Configurations with Feature Models

Consider a car configuration system inspired by similar examples from [6]. Valid configurations are formalized by the feature models shown in Fig. 4. The FMs express alternatives for the engine and locking system of the car. In fm_1 the choice between the keyless entry locking system (feature `keyless`), phone operated locking system (feature `phone`), and fingerprint secured locking system (feature `fingerprint`) is an exclusive choice (see `xor` feature group `G2` in fm_1 from Fig. 4), i.e., exactly one of the options has to be chosen.

To make the configuration process clearer an engineering team has decided to add the explicit feature `hybrid` in fm_2 for the engine of the car instead of selecting both `electric` and `gas` in fm_1. The engine selection is now an exclusive alternative in fm_2. Also, the feature `fingerprint` is a subfeature of `phone` in fm_2 and the choice of a locking system is no longer exclusive.

While browsing configurations in the semantic difference of the two FMs an engineer detects the configuration c_1 = {`car, engine, electric, locking, phone, fingerprint`} and is interested in the change operations this witness depends on. She finds that both, moving `fingerprint` below `phone` or making feature group `G2` a non-exclusive choice, are sufficient changes for the witness.

As a routine check one of the engineers checks whether all change operations contribute diff witnesses. This is not

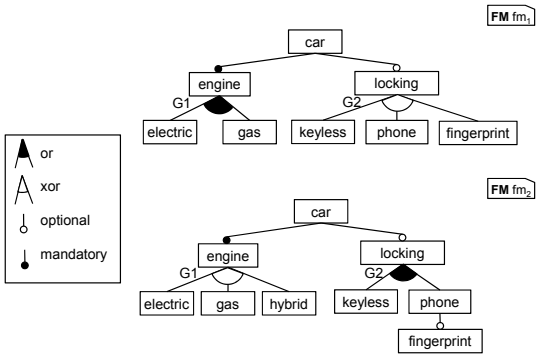

Fig. 4. Two feature models shown as feature diagrams describing valid configurations of a car inspired by similar examples from [6].

the case for the operation changing feature group `G1` to an exclusive alternative. It turns out that this change is a refinement, which removes but does not add valid configurations. All other change operations participate in minimal sets of changes sufficient for diff witnesses.

These examples illustrate the relations between syntactic change operations and semantic witnesses and how their explication provides explanations for syntactic causes and semantic effects.

III. Preliminaries

We define a general framework for connecting syntactic and semantic differences of modeling languages. The definition of relations we identified apply to modeling languages with set-based semantics, and syntactic differences characterized as syntactic change operations. We now review these general properties of modeling languages, syntactic differences, and semantic differences.

Modeling Language. A modeling language $\mathcal{L} = \langle \mathcal{M}, \mathcal{S}, sem \rangle$ is a tuple where $\mathcal{M}$ is the set of all syntactically correct (i.e., well-formed) models according to some syntax definition, $\mathcal{S}$ is a semantic domain, and $sem : \mathcal{M} \rightarrow \wp(\mathcal{S})$ defines the meaning of a model $m \in \mathcal{M}$ by mapping the model to a set of elements of $\mathcal{S}$ (see [12]). The semantic domain is typically a well-understood model that allows for formal analyses. Examples are action traces or object models. The semantic mapping relates well-formed models to elements of the semantic domain to define their meaning. As an example, a semantic mapping for ADs may relate a diagram to the set of traces describing all possible executions of the activity.

Syntactic Difference. Given two models $m_1, m_2 \in \mathcal{M}$ we describe the syntactic difference of the models in terms of partially ordered change operations $\mathcal{C}$ as $\Delta(m_1, m_2) = \{c_1, .., c_n\} \subseteq \mathcal{C}$ where the application of the change operations $\{c_1, .., c_n\}$ to the first model leads to the second model (see, e.g., [2], [5], [16], [17], [27]). Change operations are not necessarily independent [2], [5], [16], [27] and their dependencies are typically represented as a partial application order (reflexive, transitive, antisymmetric). It is important to note that the partial order is defined for change operations in

the syntactic difference $\Delta(m_1, m_2)$ and in general not for the set of all possible change operations $\mathcal{C}$.

We denote the application of change operations (according to their given order) using the operator $\oplus : \mathcal{M} \times \wp(\mathcal{C}) \rightharpoonup \mathcal{M}$, e.g., $m_1 \oplus \Delta(m_1, m_2) = m_2$. The operator is partial because some applications of change operations are not possible due to application dependencies [11], [27] or their application does not produce well-formed models in $\mathcal{M}$. To denote that the application of change operations $C \subseteq \Delta$ to a model $m \in \mathcal{M}$ leads to a well-formed model we write $m \oplus C \in \mathcal{M}$.

Semantic Difference. The semantic difference of two models $m_1, m_2 \in \mathcal{M}$ is a set of diff witnesses in the semantic domain $\mathcal{S}$. Diff witnesses are elements added to the semantics of the model between the two versions. The semantic differencing operator is defined as $\delta(m_1, m_2) = sem(m_2) \setminus sem(m_1)$ [22].

Our new operators are defined with respect to minimal and maximal sets based on standard definitions of the partial order of set inclusion. Given a set of sets $\mathcal{X}$ characterizing a property, a set $C_{max} \in \mathcal{X}$ is maximal wrt. this property iff $\forall C \in \mathcal{X} : C_{max} \subseteq C \Rightarrow C = C_{max}$ and a set $C_{min} \in \mathcal{X}$ is minimal iff $\forall C \in \mathcal{X} : C \subseteq C_{min} \Rightarrow C = C_{min}$.

IV. Relating Change Operations and Diff Witnesses

We are now ready to introduce the main contribution of this paper. We formally define the fundamental relations of necessary and sufficient syntactic changes for semantic diff witnesses in Sect. IV-A. We present language independent algorithms for computing necessary and sufficient sets of change operations in Sect. IV-B. Complementary to the algorithms we present characterizations of diff witnesses for necessary and sufficient sets in Sect. IV-C.

A. Necessary and Sufficient Change Operations

We define two complementary relations between sets of syntactic change operations and semantic diff witnesses. The following definitions express when are sets of change operations necessary or sufficient for a witness.

For two models $m_1, m_2 \in \mathcal{M}$ a set of change operations from the set of all change operations in the syntactic difference $C_{nec} \subseteq \Delta(m_1, m_2)$ is necessary for the existence of a diff witness $w \in \delta(m_1, m_2)$ iff for all applications of change operations $C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M}$ the existence of the diff witness in $sem(m_1 \oplus C')$ depends on the containment of C_{nec} in C'. Formally:

Definition 1 (Necessary change operations)**.** For models $m_1, m_2 \in \mathcal{M}$, a set of necessary change operations for a diff witness $w \in \delta(m_1, m_2)$ is a set $C_{nec} \subseteq \Delta(m_1, m_2)$ such that $\forall C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M} : w \in sem(m_1 \oplus C') \Rightarrow C_{nec} \subseteq C'$.

We denote the set of all sets of necessary change operations for models $m_1, m_2 \in \mathcal{M}$ and a witness $w \in \delta(m_1, m_2)$ by $nec(m_1, m_2, w)$.

As a simple example consider the change operations from cd_1 to cd_2 shown in Fig. 2. The addition of class `CenterLayout` is in the set of necessary change operations for the witness om_2 consisting of an object of this class. In contrast, the addition of class `CustomAttributeOwner` is not in the set of necessary change operations for this witness.

Note that there may be witnesses without necessary change operations. As an example consider the changes from fm_1 to fm_2 shown in Fig. 4. The diff witness c_1 = {`car`, `engine`, `electric`, `locking`, `phone`, `fingerprint`} has no necessary change operations. It can be obtained via two independent applications of change operations that are both sufficient (see Sect. II-C).

We now explore some interesting properties of the relation from Def. 1. First, we note that any subset of a set of necessary change operations is also a set of necessary change operations.

Theorem 1 (Subsets necessary sets)**.** Given a set of necessary change operations $C_{nec} \in nec(m_1, m_2, w)$, all subsets $C' \subseteq C_{nec}$ are also sets of necessary change operations for w.

Thm. 1 follows directly from Def. 1 because the containment of C_{nec} implies the containment of all its subsets.

Second, for all models and witnesses the empty set is the unique minimal necessary set (see Def. 1).

Third, the maximal set of necessary change operations for a witness is unique.

Theorem 2 (Max necessary set unique)**.** The maximal necessary set of change operations $C_{nec} \in nec(m_1, m_2, w)$ is unique.

Proof. Assume C_{nec} and C'_{nec} both maximal members of $nec(m_1, m_2, w)$. From Def. 1 it follows that $\forall C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M} : w \in sem(m_1 \oplus C') \Rightarrow C_{nec} \subseteq C' \wedge C'_{nec} \subseteq C'$ and thus $(C_{nec} \cup C'_{nec})$ is a necessary set. From the definition of maximal we get $C_{nec}, C'_{nec} \subseteq (C_{nec} \cup C'_{nec}) \Rightarrow C_{nec} = (C_{nec} \cup C'_{nec}) = C'_{nec}$. □

Next, we are not only interested in sets of necessary change operations but also in sets of change operations that are sufficient to produce a given witness in the semantic difference. Sufficiency is robust against additional change operations, i.e., the application of a set of sufficient change operations $C_{suff} \subseteq \Delta(m_1, m_2)$ guarantees the existence of the witness regardless of the application of additional changes from $\Delta(m_1, m_2)$. Formally:

Definition 2 (Sufficient change operations)**.** For models $m_1, m_2 \in \mathcal{M}$, a set of sufficient change operations for a diff witness $w \in \delta(m_1, m_2)$ is a set $C_{suff} \subseteq \Delta(m_1, m_2)$ such that $(m_1 \oplus C_{suff}) \in \mathcal{M}$ and $\forall C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M} : C_{suff} \subseteq C' \Rightarrow w \in sem(m_1 \oplus C')$.

We denote the set of all sets of sufficient change operations for models $m_1, m_2 \in \mathcal{M}$ and a witness $w \in \delta(m_1, m_2)$ by $suff(m_1, m_2, w)$.

As a simple example for a sufficient set of change operations consider the deletions of the action nodes labeled `UpdateCustRecord` and `CloseClaim` from Fig. 1. The two operations together are sufficient for the diff witness tr_4

from the example in Sect. II-A. Each operation alone is not sufficient for this witness.

It is very important to note that sufficiency is indeed stronger than simply *exhibiting* the diff witness after application of a set of change operations $C' \subseteq \Delta(m_1, m_2)$, i.e., $w \in \delta(m_1, m_1 \oplus C')$. As an example consider cd_3 and cd_4 shown in Fig. 7 and the diff witness om_6 shown in Fig. 8. The diff witness om_6 is in the semantics of cd_3 after adding the generalization between classes `Manager` and `Employee`. This change operations is however not sufficient to guarantee the existence of the diff witness. The diff witness is lost when further adding class `Person` and moving the association end `addressOwner` to `Person`.

Interestingly, the set of sufficient change operations for a witness w is not necessarily unique, and sufficient sets can be disjoint as in the example of FMs for witness c_1 (see Sect. II-C). However, adding change operations to a set of sufficient change operations for w results in a set of sufficient change operations for w.

Theorem 3 (Well-formed extensions of sufficient are sufficient)**.** Given a set of sufficient change operations $C_{suff} \in \mathit{suff}(m_1, m_2, w)$ all its extensions C' with $(m_1 \oplus C') \in \mathcal{M}$ are sets of sufficient change operations for w, i.e., $\forall C_{suff} \subseteq C' \subseteq \Delta(m_1, m_2) : (m_1 \oplus C') \in \mathcal{M} \Rightarrow C' \in \mathit{suff}(m_1, m_2, w)$.

Thm. 3 follows directly from Def. 2. As a corollary, note that the maximal set $C' = \Delta(m_1, m_2)$ is sufficient for every witness. Due to the extensibility of sets of sufficient change operations up to $\Delta(m_1, m_2)$, we consider the minimal sufficient sets for a witness the most informative.

The two relations of witnesses and their necessary and sufficient sets of change operations are complementary in the sense that the existence of a witness implies the application of its necessary change operations while the application of the sufficient change operations implies the existence of the witness.

Finally, all necessary change operations for w are contained in all sets of sufficient change operations for w.

Theorem 4 (Sufficient contains necessary)**.** For every witness, all sets of necessary change operations are contained in all sets of sufficient change operations:

$$\forall w \in \delta(m_1, m_2),\\ C_{nec} \in \mathit{nec}(m_1, m_2, w), C_{suff} \in \mathit{suff}(m_1, m_2, w) : \\ C_{nec} \subseteq C_{suff}$$

Proof. Follows from Def. 1 and Def. 2 because every sufficient set C_{suff} has a well-formed application to m_1 and satisfies the antecedent of implication $w \in \mathit{sem}(m_1 \oplus C') \Rightarrow C_{nec} \subseteq C'$. □

Note that the intersection of all sufficient sets contains but not necessarily coincides with the maximal necessary set. As an example consider the changes from cd_3 to cd_4 shown in Fig. 7. The witness om_6 consisting of a manager with one address (Fig. 8) has the addition of the generalization between employee and manager as the maximal necessary set of change operations but its minimal sufficient set contains also the creation of class `Person` and the move of the association end of `addressOwner`.

B. Computation of Necessary and Sufficient Sets

We present two algorithms. Given two models and a diff witness the first computes the maximal necessary set of change operations and the second computes all minimal sufficient sets of change operations. We give these algorithms in pseudo-code. Both algorithms iterate over sets of change operations from the syntactic difference of the models. Basic operations used by the algorithms are the application of change operations (operator $\oplus$), checking well-formedness ($m \in \mathcal{M}$), checking membership of a witness in the semantics of a model ($w \in \mathit{sem}(m)$), and standard set manipulation.

Algorithm 1 computes the maximal necessary set of change operations maxNec for a given witness $w \in \delta(m_1, m_2)$. Intuitively it computes the intersection of all sets of change operations that yield well-formed models with w in their semantics. The possibly empty intersection is a necessary set and satisfies Def. 1 by construction because Algorithm 1 iterates over all C' from Def. 1. The algorithm starts with the set of all change operations (l. 2), iterates over all subsets of the set of change operations (l. 3), and checks whether the application of change operations yields a well-formed model with w in its semantics (l. 5). If the model has w in its semantics the set maxNec is updated to be the intersection with the applied change operations (l. 6).

Algorithm 1 Compute max necessary set for $w \in \delta(m_1, m_2)$

```
1: define maxNec as set
2: maxNec ← Δ(m1, m2)
3: for all C ⊆ Δ(m1, m2) do
4:     m ← m1 ⊕ C
5:     if m ∈ M ∧ w ∈ sem(m) then
6:         maxNec ← maxNec ∩ C
7:     end if
8: end for
9: return maxNec
```

As an example, given cd_1, cd_2, and om_1 Algorithm 1 computes the maximal necessary set of change operations consisting of adding class `CustomAttributeOwner`, making it a superclass of `RealFigure`, and moving the association end `attribOwner` to `CustomAttributeOwner`.

As another example, given ad_1, ad_2, and tr_4 Algorithm 1 computes the maximal necessary set of change operations consisting of only deleting the action node labeled `CloseClaim`.

The time complexity of Algorithm 1 is exponential in the size of the syntactic difference of the models m_1 and m_2 because the loop in l. 3 iterates over all subsets of $\Delta(m_1, m_2)$.

Algorithm 2 computes the set of all minimal sufficient sets $\mathit{minSuff}$ for a given witness $w \in \delta(m_1, m_2)$. Intuitively for every candidate set of change operations the algorithm checks whether all extension of the set that produce a well-formed model m have $w \in \mathit{sem}(m)$. If this is the case for

all extensions the candidate set is a sufficient set according to Def. 2. The algorithm finds minimal sufficient sets first by iterating over the candidate sets of change operations in ascending order of their size (l. 4). Two sets of the same size are either equivalent or incomparable with the partial order of set inclusion. Candidate sets that are not minimal, i.e., contain a minimal sufficient set, are filtered out in l. 5.

Algorithm 2 Compute min sufficient sets for $w \in \delta(m_1, m_2)$

```
 1: define minSuff as set of sets
 2: define allExtW as Boolean
 3: minSuff ← ∅
 4: for all C ⊆ Δ(m1, m2) for growing size |C| do
 5:    if (m1 ⊕ C) ∈ M ∧ ∄C' ∈ minSuff : C' ⊆ C then
 6:       allExtW ← true
 7:       for all C ⊆ C' ⊆ Δ(m1, m2) do
 8:          m ← m1 ⊕ C'
 9:          if m ∈ M ∧ w ∉ sem(m) then
10:             allExtW ← false
11:          end if
12:       end for
13:       if allExtW then
14:          add C to minSuff
15:       end if
16:    end if
17: end for
18: return minSuff
```

As an example, given cd_1, cd_2, and om_1 Algorithm 2 computes a minimal sufficient set of change operations consisting of adding class `CustomAttributeOwner`, making it a superclass of `RealFigure`, and moving the association end `attribOwner` to `CustomAttributeOwner`.

As another example, given ad_1, ad_2, and tr_4 Algorithm 2 computes two minimal sufficient sets of change operations: (1) delete `CloseClaim` and `UpdateCustRecord` and (2) delete `CloseClaim` and convert fragment `P` from parallel to alternative.

The time complexity of Algorithm 2 is exponential in the size of the syntactic difference of the models m_1 and m_2 because the loop in l. 4 and the nested loop in l. 7 both iterate over all subsets of $\Delta(m_1, m_2)$.

C. Characterization of Diff Witnesses

Given a set of change operations our framework defines the diff witnesses these are necessary or sufficient for. The set of diff witnesses compared to the set of change operations is typically very large or even infinite and it is thus impractical to enumerate all diff witnesses. It is however possible to express the diff witnesses of necessary and sufficient sets of change operations using basic set operations and the semantic mapping $sem : \mathcal{M} \to \mathcal{S}$.

The diff witnesses a set $C \subseteq \Delta(m_1, m_2)$ is necessary for are:

$$\delta(m_1, m_2) \setminus \left(\bigcup_{\substack{C \not\subseteq C' \subseteq \Delta(m_1, m_2) \\ m = m_1 \oplus C' \in \mathcal{M}}} sem(m) \right).$$

The diff witnesses a set $C \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C) \in \mathcal{M}$ is sufficient for are:

$$\delta(m_1, m_2) \cap \left(\bigcap_{\substack{C \subseteq C' \subseteq \Delta(m_1, m_2) \\ m = m_1 \oplus C' \in \mathcal{M}}} sem(m) \right).$$

The above formalization can be employed naively as an extension of existing semantic differencing approaches based on constraint solving, where union and intersection translate to disjunction and conjunction.

V. Application to Languages and Differencing Operators

We now show how the language independent, abstract framework defined above can be applied to concrete modeling languages. We demonstrate instantiations of the framework for ADs, CDs, and FMs. These modeling languages are all very different in their concrete syntax and semantics.

The following subsections present several examples.

A. Relating Activity Diagrams Syntactic and Semantic Changes

We instantiate the relations and operators defined in Sect. IV for UML ADs.

Language: ADs define activities, the actions they are composed of, and their data- and controlflow. They are widely used for behavior modeling and in variants (BPNM) they provide means for the description of processes and workflows. We denote the syntactic domain of ADs by AD.

We define the semantics of an AD as the set of execution traces it allows for a given data input provided by the environment. The execution traces in the semantic domain of ADs consist of the names of actions executed in the activity. We denote the domain of execution traces by T and the semantic mapping that associates an AD with the execution traces in its semantics as the function $sem_{AD} : AD \to \wp(T)$. An operational semantics of ADs based on a translation to SMV following the above semantics definition is available from [23] and has been applied in an implementation of a semantic differencing operator for ADs [19].

As an approach for syntactic differencing of ADs we adapt a framework based on *compound operations* for BPNM models introduced by Küster et al. [17]. Examples of the change operations[2] presented in [17] are `insertAction(`*name*`,` *pred*`,` *succ*`)` to insert the action node labeled *name* between node *pred* and *succ*, `moveAction(`*name*`,` *pred*`)` to move the action named *name* after node *pred*, and `insertCyclicFragment(`*pred*`,` *frag*`)` to insert the cyclic fragment *frag* after node *pred*. These operations are called compound operations because they perform additional necessary steps to obtain well-formed models. As an example the operation `moveAction` removes an action, adds a controlflow edge between its former predecessor and successor,

[2]Removed first parameter (process model) and successor node parameter (in addition to *dest*) when clear in examples.

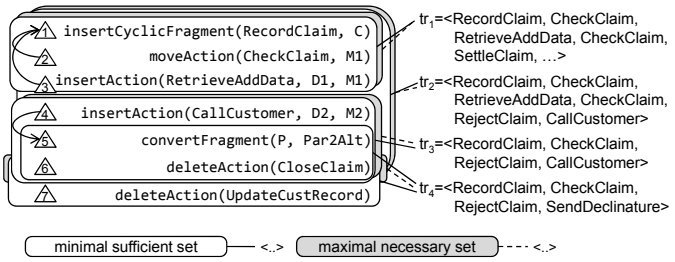

Fig. 5. Change operations from ad_1 to ad_2 as shown in Fig. 1, with application dependencies, semantic diff witnesses, and identification of minimal sufficient and maximal necessary sets of change operations. Not all witnesses are shown.

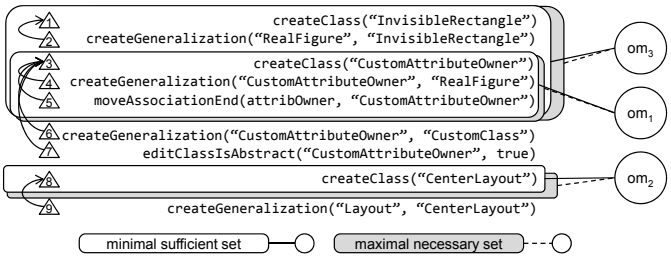

Fig. 6. Change operations from cd_1 to cd_2 as shown in Fig. 2, with application dependencies, semantic diff witnesses from Fig. 3, and identification of minimal sufficient and maximal necessary sets of change operations. Not all witnesses are shown (there are infinitely many witnesses).

and also inserts new controlflow edges at the destination position of the action.

Application Example: We illustrate the relation between the syntactic changes from ad_1 to ad_2 as shown in Fig. 1 and resulting semantic diff witnesses in Fig. 5. The left side of Fig. 5 shows a complete set of change operations and their application dependencies, e.g., the change operation `moveAction(CheckClaim, M)` depends on the insertion of the cyclic fragment `C` (here consisting of nodes `M1` and `D1`). The right side shows diff witnesses in the semantics of ad_2 but not ad_1.

Solid lines in Fig. 5 connect minimal sufficient sets of change operations (in white boxes) and traces added to the semantics of the second model through their application. Dashed lines connect maximal necessary sets of change operations (in gray boxes) and their witnesses. In this example for diff witnesses tr_1, tr_2, and tr_3 the individual minimal sufficient and the maximal necessary sets coincide, i.e., applying the necessary change operations already guarantees the existence of the witnesses regardless of other change operations applied. An interesting case is demonstrated by diff witness tr_4. The diff witness has two different but overlapping minimal sufficient sets of change operations. The maximal necessary set of change operations for tr_4 consists of the single element in the intersection of both minimal sufficient sets.

B. Relating Class Diagrams Syntactic and Semantic Changes

We instantiate the relations and operators defined in Sect. IV for UML CDs.

Language: CDs define classes, their properties, and the relations between them. They are widely used in object-oriented design and its variants Ecore and MOF are prevalent and standardized means for the definition of metamodels of modeling languages. We denote the syntactic domain of CDs by CD.

The semantics of a CD is the set of object models (OM) it allows to instantiate from its classes. Object models contain objects with concrete values for attributes defined in the CD and links between objects that conform to associations of the CD in direction and multiplicities. We denote the semantic domain OM of CDs by OM and the semantic mapping from a CD to the OMs it allows to instantiate as the function $sem_{CD} : CD \rightarrow \wp(OM)$. A definition of CD semantics based on a translation to Alloy is available, e.g., from [20]. Implementations of the semantic differencing operator for CDs were presented in [18], [21].

As an approach for syntactic differencing of CDs we choose a framework based on low-level edit operations as defined by Alanen and Porres [2] with a CD specific formalization [14], [25]. Rindt et al. [25] formalize edit operations on CDs as graph transformations implemented with EMF-Henshin. The list comprises around 100 operations, e.g., `createClass(`$cName$`)` to create a new class with name $cName$, `createGeneralization(`$parent$`,` $child$`)` to declare $child$ a subclass of $parent$, and `moveAssociationEnd(`$eName$`,` $target$`)` to move association end $eName$ to class $target$.[3]

Application Example: We illustrate the relation between the syntactic changes from cd_1 to cd_2 as shown in Fig. 2 and resulting semantic diff witnesses om_1 to om_3 in Fig. 6. The left side of Fig. 6 shows a complete set of change operations and their application dependencies, e.g., the change operation `createGeneralization("RealFigure", "InvisibleRectangle")` depends on the creation of the class `InvisibleRectangle`. The right side lists the names of diff witnesses in the semantics of cd_2 but not cd_1. The OM diff witnesses are depicted as ODs in Fig. 3.

Solid lines in Fig. 6 connect minimal sufficient sets of change operations and their OM diff witnesses. Dashed lines connect maximal necessary sets of changes and their diff witnesses. Again, for the witnesses om_3, om_1, and om_2 shown in Fig. 3 and used as examples throughout this paper, the minimal sufficient and the maximal necessary sets coincide. The last change operation `createGeneralization("Layout", "CenterLayout")` is not part of any minimal sufficient or any maximal necessary set for all witnesses in the semantic difference since the generalization has no effect on instances of cd_2 as shown in Fig. 2.

As an additional example to illustrate the relation between syntactic changes and semantic effects consider the CDs shown in Fig. 7 adapted from similar examples of [21]. The CDs show a simple model of employees with addresses and managers managing employees.

[3]Shortened names of CD change operations defined in [25]

Fig. 7. Two CDs cd_3 and cd_4 modeling employees and management adapted from similar examples of [21].

Fig. 8. Two object models in the semantics of cd_4 and not cd_3 from Fig. 7.

The data format of the ERP system of a company has been adapted from cd_3 to cd_4: the ownership of `Address` by `Employee` has been pulled up to an abstract class `Person` added in cd_4 and managers are now also employees (`Employee` identified as generalization of `Manager` in cd_4).

While performing some tests with cd_4 an engineer discovers that now managers may manage themselves while they were previously not managed. A witness of this change in the semantics is shown as om_5 in Fig. 8. The engineer wants to know which syntactic changes from cd_3 to cd_4 listed in Fig. 9 relate to this difference. She finds out that a minimal sufficient set of change operations for diff witness om_5 is the singleton set containing `createGeneralization("Manager", "Employee")`.

The engineer is unsure whether the change operation should be reverted. To further understand its effect on the semantics she checks whether there are any other diff witnesses that the change operation is necessary for. An analysis reveals that the change operation is in the maximal necessary set of om_6 (Fig. 8), which shows an instance of `Manager` that owns an instance of `Address`. In a review meeting the team decides based on this information to not revert the change operation.

We illustrate the relation between the syntactic changes from cd_3 to cd_4 as shown in Fig. 7 and resulting semantic diff witnesses in Fig. 9. The left side of Fig. 9 shows a complete set of change operations and their application dependencies, e.g., change operations numbered 3-5 depend on the change operation `createClass("Person")`. The right side lists the names of diff witnesses in the semantics of cd_4 but not cd_3. The OMs of the witnesses are depicted as ODs in Fig. 8.

Fig. 9 illustrates the relations of necessary and sufficient sets of change operations as above. An interesting case is diff witness om_6 where the maximal necessary set of changes is not empty but smaller than the minimal set of sufficient changes. The exclusive application of change operation 1 would exhibit om_6 in the semantics of the new CD. The set is however not a sufficient set because the application of change operations numbered 1-3 leads to a CD without om_6 in its semantics.

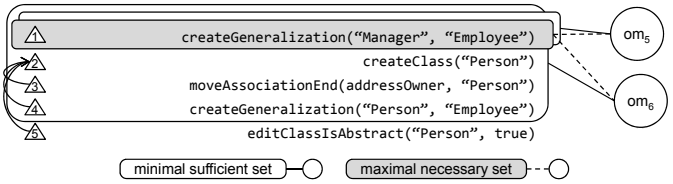

Fig. 9. Change operations from cd_3 to cd_4 as shown in Fig. 7, with application dependencies, semantic diff witnesses from Fig. 8, and identification of minimal sufficient and maximal necessary sets of change operations. Not all witnesses are shown (there are infinitely many witnesses).

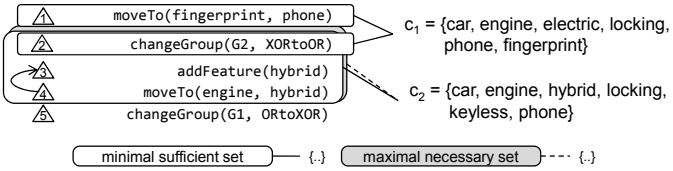

Fig. 10. Change operations from fm_1 to fm_2 from Fig. 4 with application dependencies, semantic diff witnesses, and identification of minimal sufficient and maximal necessary sets of change operations. Not all witnesses are shown.

C. Relating Feature Models Syntactic and Semantic Changes

We instantiate the relations and operators defined in Sect. IV for FMs.

Language: FMs define features and their inclusion and exclusion dependencies in valid product configurations. They are a popular description mechanism in product line engineering. We denote the syntactic domain of FMs by FM.

The semantics of a FM is defined in terms of the configurations it allows. A configuration is a valid set of features that meets all inclusion and exclusion constraints of the FM. We denote the semantic domain of FMs by $Conf$ and the semantic mapping from a FM to the configurations it allows as the function $sem_{FM} : FM \rightarrow \wp(Conf)$. A definition of FM semantics based on translations to SAT and Binary Decision Diagrams is available from [6] and has been applied in a definition of a semantic differencing operator for FMs [1].

As an approach for syntactic differencing of FMs we choose a framework based on elementary FM specific edit operations, e.g., used in Thüm et al. [28]. Examples of change operations are `moveTo(`f`,` $parent$`)` to move feature f to parent feature $parent$, `makeOptional(`f`)` to make the inclusion of feature f on the selection of its parent optional, and `addConstraint(`c`)` and `removeConstraint(`c`)` to add and remove custom constraints between features.

Application Example: We illustrate the relation between the syntactic changes from fm_1 to fm_2 shown in Fig. 4 and resulting semantic diff witnesses in Fig. 10. The left side of Fig. 10 shows a complete set of change operations. The right side lists configurations from the semantic difference between fm_1 and fm_2. Not all diff witnesses are shown.

Solid lines in Fig. 10 connect minimal sufficient sets of change operations (in white boxes) and configurations added to the semantics of the second model through their application. Dashed lines connect maximal necessary sets of change operations (in gray boxes) and their witnesses. The configuration

c_1 is an example for a diff witness with two disjoint sets of minimal sufficient change operations. Disjoint sufficient sets imply that the diff witness does not have a necessary set of change operations. The operation `changeGroup(G1, ORtoXOR)` does not participate in any minimal sufficient or any maximal necessary set of change operations for any witness in $\delta(fm_1, fm_2)$ because it is a refinement of the FM semantics and does not add diff witnesses.

VI. Discussion

We discuss challenges and limitations of our framework.

Usage Scenarios. It is important to note that our framework focuses on comprehension and not on manipulation of model differences. Comprehension of differences by engineers is important in many model maintenance tasks.

The two main usage scenarios we motivated with example questions in the introduction and concretized in Sect. II are (1) understanding the impact of a subset of change operations on the semantics of the two models and (2) tracing a diff witness to change operations. We thus complement existing differencing approaches by allowing engineers to investigate the relation between syntactic and semantic differences.

Set-Based Semantics. Our framework applies to modeling languages with set-based semantics, i.e., the language definition includes a semantic mapping that defines the semantics of a well-formed model as a set of elements in its semantic domain (see Sect. III).

Many languages have semantics definitions of this form, e.g., the configurations possible to instantiate from a FM. Most languages allow different ways to define their semantics, e.g., the semantics of ADs can be defined using symbolic transition systems [7] or using sets of possible execution traces. To apply our framework only the latter definition makes sense. Finally, there are some languages for which set-based semantics are less natural or have not been defined. Our framework does not currently support such languages.

Implementation and Computation. Existing implementations of semantic differencing operators [1], [18], [19], [21] exhibit high computational complexity for computing diff witnesses. The generic algorithms we present in Sect. IV-B do not require the computations of diff witnesses but only the check whether a given witness is in the semantics of a model after the application of different change operations. However, both algorithms have a time complexity exponential in the size of the syntactic difference (typically at least an order of magnitude smaller than the models themselves).

We expect that both the iteration of subsets of the syntactic difference and the membership check of a witness in the semantics of a model allow for language specific heuristics and optimizations. As an example, the iteration of change operation subsets that lead to well-formed ADs can be supported by known change operation application dependencies [11]. As another example, a heuristics for membership of an OM in the semantics of a CD is starting by checking that all classes with instances in the OM exist in the CD.

Presentation and Diff Interaction. Our goal in relating syntactic and semantic differences is to assist engineers with information required for comprehension of model evolution and maintenance tasks. A tight integration into IDEs and model management tools will require new ways of interaction. Existing user interfaces, e.g., the ones in our prototypes for CD and AD semantic differencing [19], [21], need to be extended with the computed relations and seamless presentation of both change operations and diff witnesses.

In other previous work, we have generated natural language explanations to help engineers understand the reasons for inconsistencies [24]. Störrle [26] used natural language text to present CD differences. The relations between syntactic operations and semantic witnesses in our framework suggest that a similar approach, to generate natural language supporting descriptions, may be effective here too.

VII. Related Work

We briefly summarize related work on syntactic differencing, semantic differencing as discussed here, alternative approaches to semantic differencing, and works that combine syntactic and semantic differencing.

Syntactic differencing operators and comparison frameworks have been suggested by many authors [2], [14], [15], [17], [27]. These frameworks range from low-level change operations [2], [25], [27] to higher-level and composed change operations [14], [15], [17]. These works do not consider the semantic effects of syntactic changes. However, many of these works are suited to provide the foundation for the syntactical change operations part in our framework; indeed in Sect. V we used change operations of ADs as defined by Küster et al. [17], of CDs as defined by Rindt et al. [25], and of FMs as used by Thüm et al.[28].

Semantic differencing operators, e.g., for ADs, CDs, and FMs, which compute elements in the semantics of one model and not the other, have been defined and implemented in works by us and by others [1], [18], [19], [21]. None has related syntactic and semantic differences. Our framework relies on the definitions of semantic difference and diff witnesses provided in these works (see instantiation examples in Sect. V).

Fahrenberg et al. [9] motivate a different approach to semantic differencing where the difference between models is expressed as a model in the same language, e.g., representing the difference of two CDs by a single CD [8]. They present a compositional algebra of CDs with subtyping, conjunctive and disjunctive merge and difference, where the operators are described by means of manipulations of syntactic elements of the diagrams with proven sound semantic implications. Major differences between our work and [8] should be noted. First, while their work is specific to CDs, ours is general, and can be applied, as we show, to many and very different modeling languages. Second, while their work defines specific syntactic manipulations, ours maps syntactic manipulations to their semantic consequences, manifested using witnesses. Their compact representation of difference might however be

an interesting complement to a fine grained analysis of change operations and diff witnesses.

Fisler et al. [10] presented impact analysis for modified access-control policies in the Margrave policy analyzer. They compare two versions of an access-control policy and define the change-impact as actions allowed after the change. The work can be seen as an instance of semantic differencing without an analysis of individual change operations. Fisler et al. support queries on the set of diff witnesses, which could be an interesting extension to existing operators and to our framework.

Semantic differencing of programs has been studied by Jackson and Ladd [13] for procedural code by presenting as diff witnesses dependencies of variables added in the new version. Apiwattanapong et al. [4] studied differencing for object-oriented programs. They use control-flow graphs extended with semantics of object-oriented concepts. These works neither compute explicit syntactic changes nor relate syntax and semantics.

Thüm et al. [28] compare FM edits relating their semantics, e.g., as refactorings or generalization, using a SAT solver. However, they do not explicitly map between specific syntactic change operations and their semantic consequences.

Kehrer et al. [14] lift low-level edit operations (as in [25], Sect. V-B) to more conceptual descriptions of model modifications. However, these descriptions are still syntactic. We believe that our framework may take advantage of lifted operations for the presentation of syntactic changes. Gerth et al. [11] use a semantic comparison of process models to detect false merge conflicts computed by syntactic comparisons (operations from [17] as used in Sect. V-A). A false conflict exists if one change operation prevents the other from being applied but both applications allow the same set of traces after application. We leave lifting the relation between change operations and diff witnesses to merge scenarios to future work.

VIII. Conclusion

We presented a novel, language independent framework, which relates syntactic and semantic model differences. The framework builds on the notion of necessary and sufficient sets of change operations for semantic diff witnesses. We formalized the framework, proved several of its important properties, and demonstrated its application to three different concrete modeling languages.

The introduction of the framework opens the way for interesting future work directions. First, the operators defined in Sect. IV can be implemented so as to extend previously presented prototype tools for semantic differencing, e.g., our own CDDiff and ADDiff tools [19], [21]. Some of these operators may be difficult to compute efficiently, and so symbolic approaches or heuristics, as we have used in CDDiff and ADDiff, may be required. We leave these for future work.

Acknowledgments

We thank the anonymous reviewers for their comments, which helped us improving the paper.

References

[1] M. Acher, P. Heymans, P. Collet, C. Quinton, P. Lahire, and P. Merle. Feature model differences. In *CAiSE*, 2012.
[2] M. Alanen and I. Porres. Difference and Union of Models. In *Proc. 6th Int. Conf. on the UML*, 2003.
[3] K. Altmanninger, M. Seidl, and M. Wimmer. A Survey on Model Versioning Approaches. *IJWIS*, 5(3):271–304, 2009.
[4] T. Apiwattanapong, A. Orso, and M. J. Harrold. JDiff: A differencing technique and tool for object-oriented programs. *Autom. Softw. Eng.*, 14(1):3–36, 2007.
[5] A. Cicchetti, D. D. Ruscio, and A. Pierantonio. A metamodel independent approach to difference representation. *Journal of Object Technology*, 6(9):165–185, 2007.
[6] K. Czarnecki and A. Wasowski. Feature diagrams and logics: There and back again. In *SPLC*. IEEE Computer Society, 2007.
[7] R. Eshuis. Symbolic model checking of UML activity diagrams. *ACM Trans. Softw. Eng. Methodol.*, 15(1):1–38, 2006.
[8] U. Fahrenberg, M. Acher, A. Legay, and A. Wasowski. Sound merging and differencing for class diagrams. In *FASE*, volume 8411 of *LNCS*, pages 63–78. Springer, 2014.
[9] U. Fahrenberg, A. Legay, and A. Wasowski. Vision paper: Make a difference! (semantically). In *MODELS*, 2011.
[10] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE 2005*, pages 196–205. ACM, 2005.
[11] C. Gerth, J. M. Küster, M. Luckey, and G. Engels. Detection and Resolution of Conflicting Change Operations in Version Management of Process Models. *Software and System Modeling*, 12(3):517–535, 2013.
[12] D. Harel and B. Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Computer*, 37(10):64–72, 2004.
[13] D. Jackson and D. A. Ladd. Semantic Diff: A Tool for Summarizing the Effects of Modifications. In *ICSM*. IEEE Computer Society, 1994.
[14] T. Kehrer, U. Kelter, and G. Taentzer. A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning. In *ASE*, 2011.
[15] T. Kehrer, U. Kelter, and G. Taentzer. Consistency-preserving edit scripts in model versioning. In *ASE*, 2013.
[16] J. M. Küster, C. Gerth, and G. Engels. Dependent and Conflicting Change Operations of Process Models. In *ECMDA-FA*, 2009.
[17] J. M. Küster, C. Gerth, A. Förster, and G. Engels. Detecting and resolving process model differences in the absence of a change log. In *BPM*, 2008.
[18] P. Langer, T. Mayerhofer, and G. Kappel. Semantic Model Differencing Utilizing Behavioral Semantics Specifications. In *MODELS*, 2014.
[19] S. Maoz, J. O. Ringert, and B. Rumpe. ADDiff: Semantic Differencing for Activity Diagrams. In *SIGSOFT FSE*, 2011.
[20] S. Maoz, J. O. Ringert, and B. Rumpe. CD2Alloy: Class Diagrams Analysis Using Alloy Revisited. In *MODELS*, 2011.
[21] S. Maoz, J. O. Ringert, and B. Rumpe. CDDiff: Semantic differencing for class diagrams. In *ECOOP*, 2011.
[22] S. Maoz, J. O. Ringert, and B. Rumpe. A manifesto for semantic model differencing. In *MODELS 2010 Workshops*, volume 6627 of *LNCS*. Springer, 2011.
[23] S. Maoz, J. O. Ringert, and B. Rumpe. An operational semantics for activity diagrams using SMV. Technical Report AIB-2011-07, RWTH Aachen University, July 2011.
[24] S. Maoz, J. O. Ringert, and B. Rumpe. Verifying Component and Connector Models against Crosscutting Structural Views. In *ICSE*, 2014.
[25] M. Rindt, T. Kehrer, U. Kelter, and P. Pietsch. Rules for Edit Operations in Class Diagrams. Technical report, University of Siegen, 2012. Available from http://pi.informatik.uni-siegen.de/qudimo/download/RiKKP2011.pdf.
[26] H. Störrle. Making Sense to Modelers - Presenting UML Class Model Differences in Prose. In *MODELSWARD*. SciTePress, 2013.
[27] G. Taentzer, C. Ermel, P. Langer, and M. Wimmer. A Fundamental Approach to Model Versioning Based on Graph Modifications: from Theory to Implementation. *Software and Systems Modeling*, 13(1):239–272, 2014.
[28] T. Thüm, D. S. Batory, and C. Kästner. Reasoning about edits to feature models. In *ICSE*, 2009.