Change Propagation Due to Uncertainty Change

Rick Salay, Jan Gorzny and Marsha Chechik

Department of Computer Science, University of Toronto, Toronto, Canada
{rsalay,jgorzny,chechik}@cs.toronto.edu

Abstract. Uncertainty is ubiquitous in software engineering; however, it has been typically handled in adhoc and informal ways within software models. Automated change propagation is recognized as a key tool for managing the accidental complexity that comes with multiple interrelated models. In this paper, we address change propagation in the context of model uncertainty and consider the case where changes in the level of uncertainty in a model can be propagated to related models. We define such uncertainty change propagation using our earlier formalization and develop automated propagation algorithms using an SMT solver. A preliminary evaluation shows that the approach is feasible.

1 Introduction

Uncertainty is ubiquitous in software engineering. It has been studied in different contexts including requirements engineering [**?**], software processes [**?**] and adaptive systems [**?**]. An area that has not received much attention is the occurrence of uncertainty in software models. Model uncertainty can be the result of incomplete information about the problem domain [**?**], alternative design possibilities [**?**], stakeholder conflicts [**?**], etc.

Despite its importance, uncertainty is typically not treated in a first-class way in modeling languages and as a result, its treatment is adhoc, e.g., including informal notes in the model. To illustrate what we mean by model uncertainty, consider Fig. 1 which shows a UML class and sequence diagram that are part of a hypothetical automotive design model that focuses on the control of the power windows. The sequence diagram shows a scenario in which a security threat is detected and the car responds by closing the windows. However, the modelers are uncertain about various facets of the design, and their points of uncertainty are indicated using the notes attached to the model elements. The top note in the sequence diagram indicates that they are not sure whether to keep the threat detection functionality separate from the car or to put it into the car. The bottom note expresses uncertainty about whether the windows should be disabled after being closed. A corresponding note can be found for the `disable()` operation in the class diagram since if the message is never sent, the operation may not be needed either. Finally, the other note in the class diagram shows that other operations may be needed.

Informal approaches such as the one we have described are adequate for capturing uncertainty information as *documentation* but they do not lend themselves to automation and mechanisms such as change propagation. To help address this problem, in previous work we have proposed a language-independent approach for expressing model uncertainty using model annotations with formal semantics [**?**].

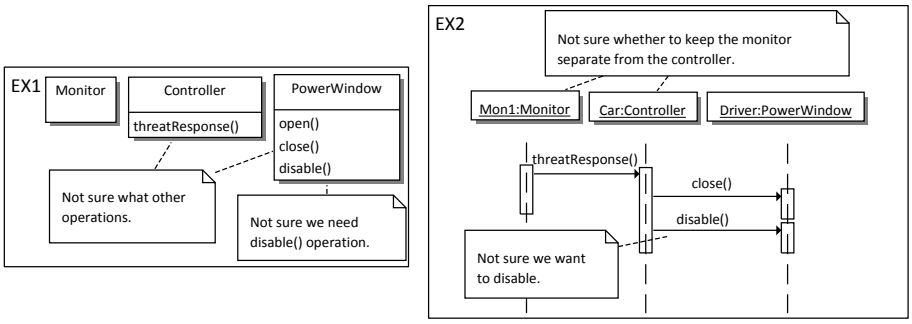


Fig. 1. A pair of related models containing uncertainty.

The *change propagation problem* [**?**] has been defined as follows: given a set of primary changes that have been made to software, what additional, secondary, changes are needed to maintain consistency within the system? Change propagation has been proposed as a mechanism to help manage and automate model evolution. Existing approaches to change propagation focus exclusively on model content changes (e.g., [**?**], [**?**],[**?**]); however, other aspects of a model may be subject to change as well: e.g., comprehensibility, completeness, etc. Model *uncertainty* is one such aspect. Changes that increase or decrease the level of uncertainty as the model evolves can force further model changes, both within the same model and across different related models – thus, uncertainty change is another context in which an automated change propagation mechanism could be used.

The key contribution of this paper is an automated approach to *uncertainty change propagation* within models. More specifically, first we identify and distinguish the problem of model uncertainty change from model content change. Second, we define the conditions for uncertainty reducing and uncertainty increasing change propagation, independently of how the uncertainty is expressed. Third, we instantiate these conditions for our formal annotation method for expressing uncertainty and define generic algorithms and tooling for computing uncertainty change propagations parameterized by a modeling language.

The remainder of the paper is organized as follows. In Sec. 2, we develop and illustrate a general approach for understanding uncertainty change and its propagation. In Sec. 3, we review the foundations of our formal annotation method for expressing model uncertainty and in Sec. 4, we apply the uncertainty change approach from Sec. 2 to this annotation method. In Sec. 5, the algorithms for uncertainty change propagation are described. In Sec. 6, they are evaluated by varying the key problem characteristics with randomly generated models. We discuss related work in Sec. 7 and make concluding remarks in Sec. 8.

2 Uncertainty Change Propagation

In this section, we develop the concept of uncertainty change propagation.

Meaning of Uncertainty. Uncertainty can be expressed as a *set of possibilities*. We can apply this approach to expressing uncertainty within a model by saying that it corresponds to the set of possible models, or *concretizations*, that are admissible given the uncertainty. A natural way to define this set is to indicate *points of uncertainty* within a

model. Although we can do so using informal notes, as in the models EX1 and EX2 of Fig. 1 (this example was described in the introduction), we present a precise, formal approach in Sec. 3. A point of uncertainty can be viewed as a constraint whose satisfaction we are unsure about, so that not all of the concretizations satisfy it. For example, the note "not sure we need the operation disable()" corresponds to the constraint "class PowerWindow has the operation disable()" which may not hold in all possible models corresponding to EX1. The points of uncertainty of model EX1 in Fig. 1 suggest that the set of concretizations of EX1, denoted by [EX1], contains all class diagrams that extend EX1 by adding zero or more operations to classes Controller and/or PowerWindow and may omit the operation disable(). Similarly, the notes attached to EX2 are its points of uncertainty, and they suggest that [EX2] contains the variants in which Mon1 and Car are merged or distinct as well as those in which the message disable() is omitted.

Uncertainty Change. A *model change* typically consists of additions, deletions and changes to the elements of the model. When we consider a model with uncertainty, an additional dimension of change becomes possible: the *level of uncertainty* may change. For example, replacing the name of the object Mon1 to MyMonitor in EX2 changes the model content but does not affect the uncertainty. However, removing the note on the message disable() reduces the uncertainty in the model, because we no longer consider concretizations that omit this message, but does not change the content of the model. Another way to change uncertainty is to increase it. For example, adding a note to say that we are not sure we need the close() message increases uncertainty without changing the content of the model.

These examples suggest that an uncertainty reducing (increasing) change to a model corresponds to reducing (increasing) the number of points of uncertainty. However, when the constraints represented by points of uncertainty depend on each other, a change at one point of uncertainty can force a corresponding change at other points of uncertainty, both *within the same model* and *in related models*. We call this process *uncertainty change propagation*. For example, suppose that EX1 and EX2 are subject to the following well-formedness constraints:

wff1 Every message in a sequence diagram must begin on a lifeline.

wff2 Every message in a sequence diagram must correspond to an operation of the message target object's class.

Assume that we perform an uncertainty reducing change in EX2, denoted by EX2 $\rightarrow$ EX2$'$. Specifically, we remove the note attached to the message disable() (i.e., we become sure that disable() occurs), resulting in the model EX2$'$. This message is contained in every concretization of EX2$'$, and, by wff2, the only well-formed concretizations of EX1 are those containing the operation disable(), i.e., the presence of this operation is *forced* by the change to EX2 and the constraint wff2. However, now the note attached to the operation disable() no longer describes a point of uncertainty because there are no concretizations that omit this operation. To repair this violation without undoing the original change in EX2, we *propagate the change* in EX1, denoted by EX1 $\dashrightarrow$ EX1$'$, by removing this note. Thus, dependencies between points of uncer-

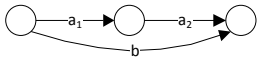


Fig. 2. Example used to show non-uniqueness of uncertainty increasing change propagation.

tainty may mean that the removal of some may force the removal of others. This process is called *uncertainty change propagation due to uncertainty reduction*.

Now assume we make an uncertainty *increasing* change, $\texttt{EX2} \rightarrow \texttt{EX2}'$, for the model EX2 in Fig. 1. The change adds a point of uncertainty as a note saying that we are not sure whether Mon1 is needed. The well-formedness constraint wff1 implies that any concretization that omits Mon1 must also omit the message threatResponse(); however, this is not possible because there is no uncertainty indicated about the presence of this message. That is, unless the presence of this message is also made uncertain, wff1 "invalidates" our newly added point of uncertainty. In order to repair this violation and retain the new point of uncertainty, we make a further, propagated, change, $\texttt{EX2}' \dashrightarrow \texttt{EX2}''$, by adding a note to the message threatResponse() indicating that its presence is now uncertain. Thus, when an added point of uncertainty is invalidated by dependencies on existing constraints (e.g., well-formedness), we may need to relax these constraints by adding (a minimal set of) further points of uncertainty. This process is called *uncertainty change propagation due to uncertainty increase*.

Unlike the case of uncertainty reducing change propagation, here the required propagated change may not be unique, i.e., there may be multiple suitable minimal sets of uncertainty points that can be added and user input is required to decide among these. For example, consider the directed graph in Fig. 2 and let one of the well-formedness constraints for this graph be "if there is a path between two nodes then there is a direct link between them". Assume we add a point of uncertainty to indicate that we are unsure whether edge b exists. The well-formedness constraint forces this edge to exist due to the presence of the path $\texttt{a}_1\texttt{a}_2$, and so this new point of uncertainty is invalidated. This can be repaired minimally in two distinct ways: saying that we are uncertain either about the existence of edge $\texttt{a}_1$ or edge $\texttt{a}_2$, requiring a user decision.

In the next two sections, we instantiate these concepts for a particular type of model with uncertainty called *MAVO*.

3 Background

In this section, we briefly review the concepts of a formal approach, introduced in [**?**], for defining a model with uncertainty called *MAVO*. A *MAVO* model is a conventional model whose elements are marked with special annotations representing points of uncertainty.

Definition 1 *A* MAVO model M *consists of a* base model*, denoted* $bs(M)$*, and a set of annotations on the base model. Let* T *be the metamodel of* $bs(M)$*. Then,* $[M]$ *denotes the set of* T *models called the* concretizations *of* M*.* M *is called consistent iff* $[M] \neq \emptyset$*.*

For example, Fig. 3(a) shows the uncertainty expressed using notes in Fig. 1 via *MAVO* annotations. The base model $bs(\texttt{EX1})$ of model EX1 is the class diagram that remains when the annotations are stripped away.

MAVO provides four types of annotations, each adding support for a different type of uncertainty in a model: Annotating an element with M indicates that we are unsure

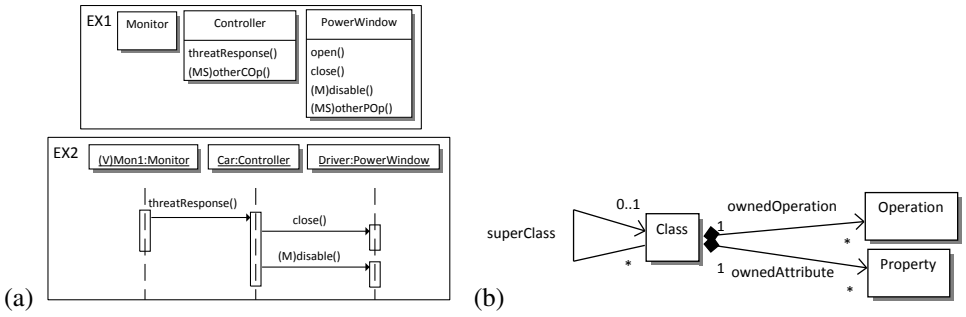


Fig. 3. (a) The models from Fig. 1 expressed using *MAVO* annotations. (b) A simplified metamodel of the UML class diagram language.

about whether it should exist in the model; otherwise, the element does exist. Thus, in `EX1`, the M-annotation on the operation `disable()` indicates that it may or may not exist in a concretization. Annotating an element with S indicates that we are unsure whether it should actually be a collection of elements; otherwise, it is just one element. This is illustrated by the S-annotation on operation `otherCOp()` in `EX1`. This annotation represents a set of operations in a concretization. The fact that it also has an M annotation means that this set could be empty. Annotating an element with V indicates that we are unsure about whether it should actually be merged with other elements; otherwise, it is distinct. Thus, we use the V-annotated object `Mon1` to consider concretizations in which it is merged with other objects such as `Car`. Finally, annotating the entire model with INC indicates that we are unsure about whether it is complete. For our simple example in Fig. 3(a), both models are assumed to be complete, and so we omit this annotation.

Formalizing *MAVO* annotations. A central benefit of using *MAVO* annotations is that they have formal semantics and thus, the set of concretizations for any *MAVO* model is precisely defined. In this section, we describe this semantics.

A metamodel represents a set of models and can be expressed as a First Order Logic (FOL) theory.

Definition 2 (Metamodel) *A* metamodel *is an FOL theory $T = \langle \Sigma, \Phi \rangle$, where Σ is the* signature *with sorts and predicates representing the element types, and Φ is a set of sentences representing the well-formedness constraints. The models that conform to T are the finite FO Σ-structures that satisfy Φ according to the usual FO satisfaction relation. We denote the set of models with metamodel T by* Mod*(T).*

The simple class diagram metamodel in Fig. 3(b) fits this definition if we interpret boxes as sorts and edges as predicates comprising $\Sigma_{\texttt{CD}}$ (where `CD` stands for "class diagram") and take the multiplicity constraints (translated to FOL) as comprising $\Phi_{\texttt{CD}}$.

Like a metamodel, a *MAVO* model represents a set of models (i.e., its concretizations) and thus can also be expressed as an FOL theory. Specifically, for a *MAVO* model M, we construct a theory $FO(M)$ s.t. *Mod*$(FO(M)) = [M]$. We proceed as follows. (1) Let $B = bs(M)$ be the base model of a *MAVO* model M. We define a new *MAVO* model M_B which has B as its base model and its sole concretization, i.e., $bs(M_B) = B$ and $[M_B] = \{B\}$. We call M_B the *ground* model of M. (2) To construct the FOL en-

$\Sigma_{\texttt{B1}}$ has unary predicates $\texttt{Ct}(\texttt{Class}), \texttt{TR}(\texttt{Operation})), \ldots,$
and binary predicates $\texttt{CtOwnsTR}(\texttt{Class}, \texttt{Operation}), \ldots$
$\Phi_{\texttt{B1}}$ contains the following sentences:
(Complete) $(\forall x : \texttt{Class} \cdot \texttt{Ct}(x) \vee \texttt{Mn}(x) \vee \texttt{PW}(x)) \wedge$
$(\forall x : \texttt{Class}, y : \texttt{Operation} \cdot \texttt{ownedOperation}(x, y) \Rightarrow (\texttt{CtOwnsTR}(x, y) \vee \ldots)) \wedge \ldots$
Ct:
(*Exists*$_{\texttt{Ct}}$) $\exists x : \texttt{Class} \cdot \texttt{Ct}(x)$
(*Unique*$_{\texttt{Ct}}$) $\forall x, x' : \texttt{Class} \cdot \texttt{Ct}(x) \wedge \texttt{Ct}(x') \Rightarrow x = x'$
(*Distinct*$_{\texttt{Ct}-\texttt{Mn}}$) $\forall x : \texttt{Class} \cdot \texttt{Ct}(x) \Rightarrow \neg\texttt{Mn}(x)$
(*Distinct*$_{\texttt{Ct}-\texttt{PW}}$) $\forall x : \texttt{Class} \cdot \texttt{Ct}(x) \Rightarrow \neg\texttt{PW}(x)$
similarly for all other element and relation predicates

Fig. 4. The FO encoding of $M_{\texttt{B1}}$.

coding of M_B, $FO(M_B)$, we extend T to include a unary predicate for each element in B and a binary predicate for each relation instance between elements in B. Then, we add constraints to ensure that the only first order structure that satisfies the resulting theory is B itself. (3) We construct $FO(M)$ from $FO(M_B)$ by *removing* constraints corresponding to the annotations in M. This constraint relaxation allows more concretizations and thus represents increasing uncertainty. For example, if an element e in M is annotated with S then the constraint that forces e to occur at most once in every concretization is removed.

We illustrate the above construction using the *MAVO* class diagram EX1 in Fig. 3(a). (1) Let $\texttt{B1} = bs(\texttt{EX1})$ be the base model of EX1 and $M_{\texttt{B1}}$ be the corresponding ground *MAVO* model.

(2) We have: $FO(M_{\texttt{B1}}) = \langle \Sigma_{\texttt{CD}} \cup \Sigma_{\texttt{B1}}, \Phi_{\texttt{CD}} \cup \Phi_{\texttt{B1}} \rangle$ (see Definition 2), where $\Sigma_{\texttt{B1}}$ and $\Phi_{\texttt{B1}}$ are model B1-specific predicates and constraints, defined in Fig. 4. They extend the signature and constraints for CD models described in Fig. 3(b). For conciseness, we abbreviate element names in Fig. 4, e.g., Controller becomes Ct, threatResponse becomes TR, etc. We refer to $\Sigma_{\texttt{B1}}$ and $\Phi_{\texttt{B1}}$ as the *MAVO* predicates and constraints, respectively.

Since $FO(M_{\texttt{B1}})$ extends CD, the FO structures that satisfy $FO(M_{\texttt{B1}})$ are the class diagrams that satisfy the constraint set $\Phi_{\texttt{B1}}$ in Fig. 4. Assume N is such a class diagram. The *MAVO* constraint *Complete* ensures that N contains no more elements or relation instances than B1. Now consider the class Ct in B1. *Exists*$_{\texttt{Ct}}$ says that N contains at least one class Ct, *Unique*$_{\texttt{Ct}}$ – that it contains no more than one class Ct, and the clauses *Distinct*$_{\texttt{Ct}-*}$ – that the class Ct is different from all the other classes. Similar *MAVO* constraints are given for all other elements and relation instances in EX1. These constraints ensure that $FO(M_{\texttt{B1}})$ has exactly one concretization and thus $N = \texttt{B1}$.

(3) Relaxing the *MAVO* constraints $\Phi_{\texttt{B1}}$ allows additional concretizations and represents a type of uncertainty indicated by an annotation. For example, if we use the INC annotation to indicate that B1 is incomplete, we can express this by removing the *Complete* clause from $\Phi_{\texttt{B1}}$ and thereby allow concretizations to be class diagrams that extend B1. Similarly, expressing the effect of the M, S and V annotations for an element E correspond to relaxing $\Phi_{\texttt{B1}}$ by removing *Exists*$_E$, *Unique*$_E$ and *Distinct*$_{E-*}$ clauses, respectively. For example, removing the *Distinct*$_{\texttt{Ct}-*}$ clauses is equivalent to marking the class Ct with V (i.e., Controller may or may not be distinct from another class).

Thus, for each pair (a, e) of model M, where a is a *MAVO* annotation and e is a model element, let $\varphi_{(a,e)}$ be the corresponding *MAVO* constraint that is removed from the FO encoding of M. For the above example, $\varphi_{(\mathtt{V},\mathtt{Ct})} = \mathit{Distinct}_{\mathtt{Ct}-*}$.

4 Formalizing Uncertainty Change Propagation

In this section, we formalize the notion of uncertainty change propagation between models with uncertainty expressed using *MAVO*. We then define algorithms for uncertainty reducing/increasing change propagation based on this formalization.

As discussed in Sec. 3, the points of uncertainty in a *MAVO* model are expressed using annotations on the model elements. In Sec. 2, we argued that a point of uncertainty corresponds to a constraint which we are not sure holds. That is, there must exist a concretization for which it doesn't hold, otherwise we would be certain about the constraint and it couldn't represent a point of uncertainty. Thus, a validity requirement for a point of uncertainty is that there should be some concretization in which it does not hold.

In the FO encoding for a *MAVO* model, we attempt to guarantee this validity requirement by explicitly omitting the constraint $\varphi_{(a,e)}$ corresponding to each annotation a of an element e. However, this may not always be sufficient since the constraint may still be implied by others (e.g., well-formedness), making the annotation an invalid point of uncertainty. When all annotations satisfy the validity requirement, we say that the *MAVO* model is in *reduced normal form* (RNF).

Definition 3 (Reduced Normal Form (RNF)) *Let M be a* MAVO *model with $FO(M) = \langle \Sigma, \Phi \rangle$ and let Φ_A be the set of* MAVO *constraints corresponding to the annotations in M. M is in* reduced normal form *(RNF) iff* $\forall \varphi_{(a,e)} \in \Phi_A \cdot \neg(\Phi \Rightarrow \varphi_{(a,e)})$.

When a model is in RNF, the validity requirement holds for all of its annotations: if the *MAVO* constraint $\varphi_{(a,e)}$ for an annotation a of an element e does not follow from Φ, there must be a concretization that does not satisfy $\varphi_{(a,e)}$. We now use RNF as a way to formally define the notion of uncertainty reducing and increasing change propagation for *MAVO* models.

Definition 4 *Let M and M' be* MAVO *models. $M \dashrightarrow M'$ is an* uncertainty reducing propagated change *if $[M] = [M']$, and M' is obtained by removing annotations so that M' is in RNF.*

Definition 5 *Let M and M' be* MAVO *models. $M \dashrightarrow M'$ is an* uncertainty increasing propagated change *if $[M'] \subset [M]$, and M' is obtained by adding a minimal number of annotations to M so that M' is in RNF.*

To illustrate the application of these definitions, we recast the uncertainty change propagation examples of Sec. 2 in terms of *MAVO* annotations. To be able to express *cross-model* propagation in the FO encoding, we treat both diagrams in Fig. 3(a) as part of a single bigger model. We resolve naming conflicts by appending the model name to the element name, e.g., `disable_EX1`. In the first example, we perform an uncertainty reducing change `EX2` $\rightarrow$ `EX2`$'$ by removing the M annotation attached to the message `disable`() and thus $\Phi_{\mathtt{EX2}'}$ contains the additional *MAVO* constraint $\mathit{Exists}_{\mathtt{disable\_EX2}}$.

Together, $\mathit{Exists}_{\texttt{disable\_EX2}}$ and wff2 imply the constraint $\mathit{Exists}_{\texttt{disable\_EX1}}$, forcing the existence of the operation `disable()` in `EX1`. However, this operation has an M annotation on the operation `disable()`, so `EX1` is not in RNF. We repair the problem by performing a change propagation $\texttt{EX1} \dashrightarrow \texttt{EX1}'$ (Definition 4), removing the annotation on the operation `disable()`.

In the second example, assume that an uncertainty increasing change $\texttt{EX2} \rightarrow \texttt{EX2}'$ is made by adding an M annotation to `Mon1` to indicate that we are not sure whether it exists. However, since the message `threatResponse()` has no M annotation, $\Phi_{\texttt{EX2}'}$ contains the constraint $\mathit{Exists}_{\texttt{threatResponse}}$ and together with wff1, this implies the constraint $\mathit{Exists}_{\texttt{Mon1}}$. Thus, $\texttt{EX2}'$ is not in RNF. Definition 5 says that to repair this, we should propagate the change, $\texttt{EX2}' \dashrightarrow \texttt{EX2}''$, by adding a minimal set of annotations that put $\texttt{EX2}''$ into RNF. In this case, it is sufficient to add an M annotation to the message `threatResponse()` so that the above implication with wff1 does not happen. While this solution is unique and minimal, this is not the case in general (recall the example in Fig. 2).

5 Uncertainty Change Propagation Algorithms

Definitions 4 and 5 provide a specification for uncertainty change propagation. We now describe algorithms for these. Recall that *MAVO* constraints in a FO encoding of a model M correspond to *missing* annotations in M, so adding an annotation a to an element e in M is equivalent to removing the corresponding *MAVO* constraint $\varphi_{(a,e)}$ from the encoding, and vice versa.

5.1 Uncertainty Reducing Change Propagation

Fig. 5(a) shows Algorithm URCP for computing the change propagation due to an uncertainty reducing change. The objective of this algorithm is to put the input model M with $FO(M) = \langle \Sigma, \Phi \rangle$ into RNF (see Definition 3). The main loop in lines 3-16 achieves this by iterating through all annotations (a, e) of M. It then checks whether $\Phi \Rightarrow \varphi_{(a,e)}$, where $\varphi_{(a,e)}$ is the *MAVO* constraint corresponding to this annotation. If so, the annotation can be removed.

First, a satisfiability check is made in line 4 to find a satisfying instance I of $(\Phi \cup \{\neg\varphi_{(a,e)}\})$. If one is *not* found, it means that $\Phi \Rightarrow \varphi_{(a,e)}$ and so the annotation a for an element e in the output model M' can be removed (line 13). Otherwise, I is used to find other annotations that can also be removed (lines 5-11). For each annotation (a', e'), we check whether I is also a counter-example to the corresponding *MAVO* constraint $\varphi_{(a',e')}$. For example, if (a', e') is an M-annotation on some element e', the call $\mathit{NotExists}(e', I)$ checks whether e' is missing in I. If so, then $\Phi \not\Rightarrow \varphi_{(a',e')}$, and the annotation can be removed from further consideration (line 9). Conditions for annotations S and V are checked similarly. The strategy of using a satisfying instance to more quickly eliminate annotations is inspired by a similar strategy used for computing the backbone of a propositional formula (i.e., the set of propositional variables that follow from the formula) with a SAT solver [**?**].

Correctness. Originally, M' is equal to M (line 1). An annotation a is removed from the element e in M' (line 13) only if the condition $\Phi \cup \{\neg\varphi_{(a,e)}\}$ is not satisfiable (line

(a)

Algorithm: URCP
Input: *MAVO* model M with FO encoding $FO(M) = \langle \Sigma, \Phi \rangle$
Output: *MAVO* model M' satisfying Definition 4

1: $M' \leftarrow M$
2: $A \leftarrow$ *Annotations*(M)
3: **for** $(a, e) \in A$ **do**
4: **if** *SAT*$(\langle \Sigma, \Phi \cup \{\neg\varphi_{(e,a)}\}\rangle, I)$ **then**
// I is a satisfying instance
5: **for** $(a', e') \in A$ **do**
6: **if** (a' is M **and** *NotExists*(e', I))
7: **or** (a' is S **and** *NotUnique*(e', I))
8: **or** (a' is V **and** *NotDistinct*(e', I)) **then**
9: $A \leftarrow A \setminus \{(a', e')\}$
10: **endif**
11: **endfor**
12: **else**
13: remove annotation a from e in M'
14: **endif**
15: $A \leftarrow A \setminus \{(a, e)\}$
16: **endfor**
17: **return** M'

(b)

Algorithm: UICP
Input: *MAVO* model M with FO encoding
$FO(M) = \langle \Sigma, \Phi_T \cup \Phi_M \rangle$,
a subset *New* of *Annotations*(M)
identified as new
Output: *MAVO* model M' satisfying Definition 5

1: $\Phi_{soft} \leftarrow \Phi_M$
2: $\Phi_{hard} \leftarrow \Phi_T \cup \{\neg\varphi_{(a,e)} | (a, e) \in New\}$
3: **if** *MAXSAT*$(\Phi_{soft}, \Phi_{hard}, \Phi_{relax})$ **then**
4: **return** $M' \leftarrow M \cup \{(a, e) | \varphi_{(a,e)} \in \Phi_{relax}\}$
5: **else**
6: **return** ERROR
7: **endif**

Fig. 5. Algorithms to compute the change propagation of *MAVO* model $M \dashrightarrow M'$: (a) due to an uncertainty reducing change; (b) due to an uncertainty increasing change.

4). Furthermore, every annotation that passes this condition is removed from consideration (set A) either on line 9 or line 15. Thus, M' is in RNF, and the algorithm correctly implements Definition 4.

Complexity. A *MAVO* model restricted only to M annotations (a.k.a. a *May* model) can be seen as equivalent to a propositional formula where the M-annotated elements are the propositional variables [**?**]. In this case, the RNF corresponds to removing the elements in the backbone of this formula. Thus, the complexity of Algorithm URCP is at least that of computing the backbone of a propositional formula, which is NP-hard [**?**]. Furthermore, all other computations in the algorithm are polynomial time, so we can conclude that URCP is also NP-hard. Algorithm URCP uses a SAT solver (with that complexity). Since the outer loop is bounded by the number of annotations n_A, the SAT solver is not called more than n_A times.

5.2 Uncertainty Increasing Change Propagation

The algorithm utilizes a solver for the partial maximum satisfiability (MAXSAT) problem (e.g., see [**?**]). The partial MAXSAT problem takes a set of *hard* clauses Φ_H and a set of *soft* clauses Φ_S and finds a maximal subset $\Phi_{S'} \subseteq \Phi_S$ such that $\Phi_H \cup \Phi_{S'}$ is satisfiable. Thus, a solution (which may not be unique) represents a minimal relaxation of the soft constraints that with the hard constraints will allow satisfiability.

Fig. 5(b) gives an algorithm for computing the change propagation due to an uncertainty increasing change using partial MAXSAT. The input is a *MAVO* model M with $FO(M) = \langle \Sigma, \Phi_T \cup \Phi_M \rangle$ and a subset *New* of annotations of M identified as new due to an uncertainty increasing change. Φ_T are the well-formedness rules for M from its metamodel T and Φ_M are the *MAVO* constraints for the annotations missing from M.

We assume that M was in RNF prior to adding annotations *New* but now may not be – this assumption is used in the discussion about correctness below.

Since M is not necessarily in RNF, it may be that $\Phi_T \cup \Phi_M \Rightarrow \varphi_{(a,e)}$ for some annotations $(a, e) \in$ *New*. Thus, according to Definition 5, we must add a minimal set of annotations from M so that this implication no longer holds for any *New* annotation. We accomplish this by using MAXSAT to minimally relax Φ_M.

In line 1, the *MAVO* constraints Φ_M are set as the *soft* constraints since our objective is to find a minimal set of these to relax. The *hard* constraints, set in line 2, consist of the well-formedness rules and the negations of the *MAVO* constraints for the *New* annotations. Line 3 makes the MAXSAT call, and the output M' is constructed in line 4 by adding the annotations corresponding to the relaxed clauses. If no possible relaxation exists, the algorithm ends in error (line 6). This means that some of the *MAVO* constraints for new annotations are implied directly by the well-formedness constraints and so removing some new annotations is unavoidable in order for M to be in RNF.

Correctness and complexity. Since MAXSAT is guaranteed to find a relaxation (if one exists), the *MAVO* constraints for the annotations in *New* will not be implied by $FO(M')$. Furthermore since we assumed that M was already in RNF prior to adding the new annotations, none of the *MAVO* constraints for the remaining annotations (i.e., other than the new annotations) will be implied by $FO(M')$. Thus, M' is in RNF, and the UICP correctly produces the result as specified by Definition 5.

UICP consists of single call to a partial MAXSAT algorithm and thus its complexity is equivalent to MAXSAT. For example, the implementation reported in [**?**] calls a SAT solver is called at most $(3n - n_A) + 1$ times, where n is the number of elements in M and n_A is the number of annotations in M.

6 Experiments

We performed a series of experiments to investigate the following research questions related to the scalability of automated uncertainty change propagation:

RQ1 How is uncertainty change propagation affected by how constrained the model is?

RQ2 How is uncertainty change propagation affected by the level of uncertainty in the model?

RQ1 helps us understand the impact of well-formedness constraints while RQ2 is related to the number of annotations.

Experimental Design. We conducted four experiments, to study each of RQ1 and RQ2 with uncertainty reducing or increasing change propagation. In our experiments, we assumed that our models are untyped randomly generated graphs. This is a reasonable simplification since typing information can be seen as a form of constraint. We discretized the space of random models into four size categories defined by the following ranges in the number of elements: $(0, 25], (25, 50], (50, 75], (75, 100]$ (the same categories have been used in experiments in [**?**]).

In the RQ1 experiments, we assumed a fixed graph density[1] of 0.11. We also assumed that a fixed percentage of model elements, 36%, are *MAVO*-annotated. Of the

[1] Graph density is the ratio of the number of edges to the square of the number of nodes.

annotations, 48% were M, 33% were S and 43% were V (the numbers add up to greater than 100% because some elements are multiply annotated). The values of these parameters correspond to the average percentages of the corresponding annotations that we have observed in the existing case studies using *MAVO* [**?**,**?**].

To vary the degree to which the model is constrained, for each randomly generated model we computed a set of constraints that guaranteed that $k\%$ of the annotations would either be removed (for uncertainty reducing change propagation) or added (for uncertainty increasing change propagation). We considered the values of k in the set $\{0, 25, 50, 75, 100\}$.

To understand how the constraints for uncertainty reducing change propagation were generated, assume that a randomly generated *MAVO* model has n annotations. We first choose $n' = n * k/100$ of the annotations arbitrarily. Then we generate a new well-formedness constraint $\varphi_0 \Rightarrow \varphi_1 \Rightarrow \ldots \Rightarrow \varphi_{n'}$, where φ_0 is the *MAVO* constraint for an arbitrarily chosen missing annotation and the remainder are the *MAVO* constraints for the n' annotations. Since φ_0 is the *MAVO* constraint for a missing annotation, it must therefore hold, and so all the remaining sentences are implied. This causes the uncertainty reducing change propagation algorithm to remove the corresponding annotations.

For uncertainty increasing change propagation, assume that the randomly generated *MAVO* model has n missing annotations. We first choose $n' = n * k/100$ of the missing annotations arbitrarily. Then we create a new constraint $(\varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_{n'}) \Rightarrow \varphi_0$, where φ_0 is the *MAVO* constraint for an arbitrarily chosen *new* annotation and the remaining are the *MAVO* constraints for the n' annotations. Since these n' annotations are missing, all of the constraints φ_i, for $1 \leq i \leq n'$, must hold. Furthermore, each implies φ_0. Thus, the uncertainty increasing change propagation algorithm is forced to add all n' annotations in order to achieve RNF.

In the RQ2 experiments, we used the same graph density (0.11) as for RQ1 but varied the total percentage of annotations using values in the set $\{25, 50, 75, 100\}$ [2] while keeping the same relative percentages of each annotation type as for RQ1. In addition, we fixed how the model is constrained, at $k = 50\%$.

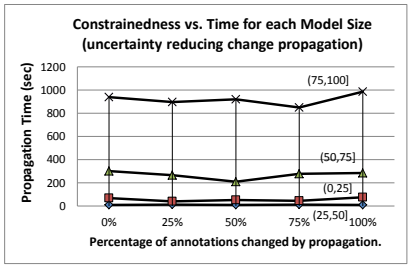
Overall, for each RQ1 experiment, we produced 20 test configurations (four sizes times five constraint levels). For each RQ2 experiment, we had 16 test configurations (four sizes times four uncertainty levels). Each test configuration was run up to 20 times – models were generated randomly until the average time of processing was within a confidence interval of 0.7 or until 20 random models were sampled.

Implementation. We used the Z3 SMT solver[3] for both algorithms. The built-in theory of uninterpreted functions and support for quantifiers made it convenient for expressing the *MAVO* FO encoding. In addition, Z3 provides an implementation of MAXSAT based on Fu and Malik [**?**]. Z3 v.4.1 was used for URCP while Z3 v.3.2 was used for UICP because v.4.1 had bugs in the MAXSAT implementation.

[2] The case of 0% annotations is omitted because no propagation occurs.

[3] http://research.microsoft.com/en-us/um/redmond/projects/z3/

(a)



(b)

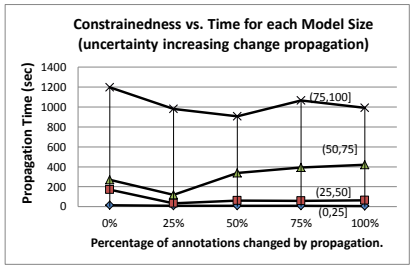


Fig. 6. RQ1 results for (a) uncertainty reducing change propagation, and (b) uncertainty increasing change propagation experiments.

Each randomly generated model was generated in eCore[4] and then translated to SMTLib[5] using Python as input to Z3. All tests were run on a laptop with an Intel Core i7 2.8GHz processor and 8GB of RAM.

Results. Figs. 6 and 7 summarize the obtained results for the RQ1 and RQ2 experiments, respectively. The RQ1 experiment for uncertainty reduction in Fig. 6(a) shows a surprising result: the time does not seem to be significantly affected by how constrained the model is. An analysis of the URCP algorithm in Fig. 5 suggests the reason for this. Changing an annotation is much more expensive than not changing it because the former only happens when the SMT solver returns UNSAT (line 13) which requires it to consider all concretizations. Thus, on the one hand, adding more constraints should reduce the SMT solving time because it has fewer concretizations to consider. On the other hand, our constraints are designed to increase the number of changes to annotations, and so URCP is forced to perform the more expensive processing that offsets the speed gain. The results for uncertainty increase in Fig. 6(b) are similar but slightly more variable. In both uncertainty reducing and increasing cases, there seems to be consistent (but small) dip in propagation time going from 0% to 25% constrainedness. This may suggest that a small amount of constraint is optimal.

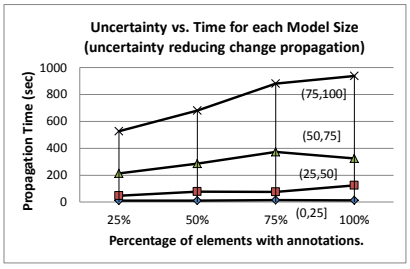
Both of the RQ2 experiments in Fig. 7 exhibit a similar linear increase in propagation time with increasing uncertainty. Adding annotations increases the number of concretizations so an increasing propagation time is to be expected. The linear relationship is a desirable outcome given that the number of concretizations increases exponentially relative to the number of annotations.

In summary, although in all four experiments the propagation time increased exponentially with model size, as is expected when using a SMT solver, the time was relatively unaffected by degree of constrainedness and increases linearly with the degree of uncertainty. Both of these positive results point to the feasibility of tool support for uncertainty change propagation.

Threats to validity. The use of randomly generated models in our experiments is a threat to validity because they may not correctly reflect change propagation behaviour on real models. Another threat to validity is our approach for generating constraints. We

[4] http://www.eclipse.org/modeling/emf/?project=emf
[5] http://www.smtlib.org/

(a)



(b)

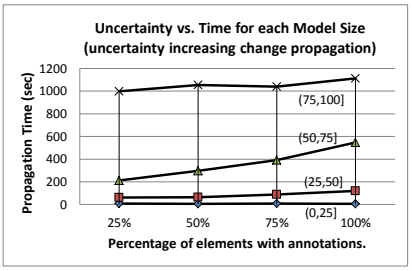


Fig. 7. RQ2 results for (a) uncertainty reducing change propagation, and (b) uncertainty increasing change propagation experiments.

used a method that guarantees particular levels of propagation, but such constraints may not correspond to the actual well-formedness constraints used in modeling languages.

In order to help mitigate the first threat, we tuned the parameters for generating random *MAVO* models so that the graph density and the frequencies of annotations corresponded to those we have observed with *MAVO* models created by hand for different case studies.

7 Related Work

In this section, we review approaches to conventional change propagation and discuss their relation to our method for *MAVO* uncertainty change propagation.

Change propagation can be seen as finding a "repair" to reinstate model consistency after a change has made it inconsistent. The difference between the conventional change propagation studied in the literature, and *MAVO* uncertainty change propagation is the nature of the consistency constraint used. In the former, these are usually well-formedness rules, either within a model or in the traceability relation between models. In the latter, the constraint is that the model be in RNF.

Many approaches [**?**,**?**,**?**,**?**,**?**] focus on attempting to formulate *repair rules* representing various change scenarios where specific repair actions are performed in response to detected changes. Such rules may be expressed in a specialized constraint language, such as Beanbag [**?**] or EVL [**?**], or using logic, such as Blanc et. al [**?**], triple graph grammars [**?**], the xLinkit framework [**?**] or, more recently, Reder et. al. [**?**]. With the exception of xLinkit and Reder, these approaches require that the repair rules be created by hand, and thus are modeling-language specific. These approaches are inappropriate for use with *MAVO* as they go against the language-independent spirit of *MAVO*. On the other hand, xLinkit automatically generates the repair rules from from consistency constraints and thus is language-independent, although the constraints are different from ours. [**?**] takes a similar approach but the rules organize the repairs into trees to simplify user selection. In the future, we intend to investigate the feasibility of automatically generating uncertainty repair rules for each language.

Another approach to conventional change propagation is to use a general constraint solver to find possible repairs. For example, [**?**] expresses the consistency constraints declaratively using Answer Set Programming (ASP) and then finds possible modifications to reinstate consistency using an ASP solver. Analysis of feature models often uses

constraint solvers as well, e.g., [**?**]. Feature models represent a set of products in a manner similar to the *MAVO* model representation of a set of concretizations. Lopez [**?**] uses a SAT solver on feature models to fix inconsistencies that allow feature configurations which yield inconsistent products. Benavides [**?**] uses a solver for *false optional feature* detection – finding cases in which a feature is marked as optional when the constraints actually force it to occur in all products. This is similar to removing an M annotation in a *MAVO* model as part of an RNF computation. Addressing S and V annotations in addition to M , as we do in our work, extends this kind of analysis further.

Since our main focus in this paper is uncertainty change propagation, we consider the work related to *expressing* model uncertainty to be out of scope; see [**?**] for a recent survey.

8 Conclusion

The management of uncertainty and its negative impacts in software engineering is an important problem. In this paper, we extend our earlier work on model uncertainty [**?**] to address the issue of change propagation due to uncertainty change. We identified two general cases of uncertainty change propagation for uncertainty reducing change and for uncertainty increasing change. We then formally specified these cases and defined algorithms for computing the propagation. Although the cases appear to be symmetric, the uncertainty reducing case produces a unique solution while the uncertainty increasing case might not, requiring user interaction. Furthermore, their solutions require different algorithms. We implemented both algorithms on top of the Z3 SMT solver and performed scalability experiments using randomly generated models. Our experiments revealed that although change propagation time increases exponentially with model size, as is expected with the use of SMT solvers, it was unaffected by how constrained the model is and only increases linearly with the degree of uncertainty in the model. These positive results suggest the feasibility of tool support for uncertainty change propagation.

Our experiences with the current work suggest some interesting future directions. The generality of SAT/SMT solving comes at a cost of potentially exponential behaviour, and we found this to be the case for our experiments involving randomly generated models. Of course, it is possible that "real" models avoid this bad behavior – this has been reported to be the case with real feature models [**?**]. We intend to conduct studies with real *MAVO* models to investigate this hypothesis. The appendix describes some promising preliminary results. Another direction to improve performance is to exploit more efficient algorithms in specialized cases. For example, there may be classes of well-formedness constraints for which we can automatically generate efficient change propagation rules in a manner similar to the approach used by xLinkit for conventional change propagation [**?**]. Finally, we intend to investigate how conventional change propagation can be combined with uncertainty change propagation to provide a more comprehensive change propagation solution. This will be an important step toward a general approach for managing model uncertainty across the software development lifecycle.

Acknowledgements. We thank Alessio Di Sandro, Vivien Suen, Michalis Famelis, Pooya SaadatPanah and Nathan Robinson for their help with developing the model gen-

eration framework used in this paper. We would also like to thank Aws Albarghouthi for his help with Z3.

9 APPENDIX

In this section, we describe an uncertainty change propagation case study using models of the open source software project UMLet[6]. UMLet is a simple UML editor consisting of a drawing canvas and a palette of shape templates. A drawing is created by copying template shapes and then configuring them by filling in a textual markup that changes their content and appearance. UMLet has an online issue log[7]. In this case study, we focus on one particular defect referred to as Issue 10, concerning a problem with the copy/paste behaviour: when a shape is copied (whether from the palette or canvas) and then pasted, it does not appear on top of the "z-order", i.e., the stacking order of overlapping shapes. As a result, it is covered by other shapes. We first studied this bug in [**?**] where our objective was to do property checking with models containing uncertainty. In [**?**], we only considered *May* uncertainties. In this study, our purpose is to analyze uncertainty change propagation, and we consider full *MAVO*.

We used the Borland TogetherJ tool[8] to reverse engineer models in order to isolate the problem and then made modifications to fix it. The paste functionality is invoked by instantiating the `Paste` class and calling the `execute` operation. Fig. **??** shows a fragment of the sequence diagram for the `execute` operation and a corresponding class diagram describing both the problem and the fix. The `loop` construct iterates (using index `e`) through the elements in the clipboard and adds them to the canvas represented as `pnl` : `DrawPanel`. The problem occurs because the `AddElement` command object does not set the z-order to 0. The initial version of the fix is to create a "positioner" object `pos` responsible for moving the newly pasted item to the top. This is invoked by calling its `moveToTop` operation which calls `setComponentZOrder` to actually move the item and then notifies other objects.

There are several points of uncertainty in this fix. First, we annotate `pos` with V to indicate that we are not sure whether it should be a new object or part of an existing object. This also means that we are not sure what class `pos` is an instance of (V-annotated placeholder class `Positioner`) and whether it needs to be constructed (the M-annotated message `new`). Second, we are not sure how many notifications will be made and where they will go. Hence, we have the MS-annotated message $\texttt{notifications}_{\texttt{mess}}$ repre-

[6] www.umlet.com
[7] code.google.com/p/umlet/issues/list
[8] www.borland.com/us/products/together/

senting an arbitrary set of messages which are sent to an MSV-annotated object L representing an arbitrary set of objects that may include existing ones. These objects are instances of a correspondingly annotated class Listeners with the annotated operation $\texttt{notifications}_{\texttt{op}}$.

We now consider a series of uncertainty reductions and increases with the associated propagated changes. These are summarized in Table **??**. The well-formedness constraints used for propagation are listed in Table **??** (wff1 and wff2 have been used in Sec. 2 . Change (1) is an uncertainty-reducing change initiated when the modeler decides that, to minimize performance impact, exactly one notification message should be sent. However, she is still unclear exactly where it should be sent. Ensuring that this message always exists (i.e., no M) forces both the object L (due to wff1) and the operation $\texttt{notifications}_{\texttt{op}}$ to exist (due to wff2) as well. In turn, these force the class Listeners to exist (due to wff3, wff4).

Making message $\texttt{notifications}_{\texttt{mess}}$ unique (i.e., no S) does not propagate further because none of the constraints affect uniqueness of the object L or the operation $\texttt{notifications}_{\texttt{op}}$. However, we no longer need multiple occurrences of these. We thus apply change (2) (see Table **??**), forcing them, as well as the class Listeners, to be unique. Note that the annotations after L : refer to the instanceOf relation between L and Listeners and it is made unique by propagation because a single object cannot be an instance of more than one class (due to wff4).

Finally, change (3) is an uncertainty increasing change motivated by the realization that it may be more efficient to incorporate the move to the top of the Z-order directly into the execute operation of the elem : AddElement command object. To express this, we add an M-annotated $\texttt{SetComponentZOrder}_{\texttt{elem}}$ message (from elem) during execute. Then we M-annotate class the Positioner to indicate that it is no longer necessarily needed. This last annotation has the propagated effect of M-annotating all of Positioner's operations since they cannot exist if their class is removed (due to wff3). Similarly, the instance of Positioner and its messages become M-annotated (due to wff1, wff2 and wff4).

Table **??** shows the time to generate the change propagations shown in Table **??** using the tooling described in Sec.6 . In order to compare these times to the results reported in Figs.6-7 , we classify them following the scheme used in the experiments. An analysis of the abstract syntax of the models in Fig. **??** reveals that they have a combined 39 elements and 17 annotations: 6 M annotations, 5 S annotations and 6 V annotations. This data implies that it falls into the $(25, 50]$ size category and the 25% uncertainty category. Also, since (at most) 4 of the 17 annotations get changed in

ID	Change	Change Propagation
(1)	(MS)notifications$_{\text{mess}}$ → notifications$_{\text{mess}}$	(MSV)L : (MSV)Listeners ⇢ (SV)L : (SV)Listeners (MS)notifications$_{\text{op}}$ ⇢ (S)notifications$_{\text{op}}$ (MSV)Listeners ⇢ (SV)Listeners
(2)	(SV)L : (SV)Listeners → (V)L : (SV)Listeners (S)notifications$_{\text{op}}$ → notifications$_{\text{op}}$ (SV)Listeners → (V)Listeners	(V)L : (SV)Listeners ⇢ (V)L : (V)Listeners
(3)	→ (M)setCompZOrder$_{\text{elem}}$ (V)Positioner → (MV)Positioner	moveToTop$_{\text{op}}$ ⇢ (M)moveToTop$_{\text{op}}$ (V)pos ⇢ (MV)pos moveToTop$_{\text{mess}}$ ⇢ (M)moveToTop$_{\text{mess}}$ setCompZOrder$_{\text{pos}}$ ⇢ (M)setCompZOrder$_{\text{pos}}$

Table 1. Case study model changes and propagations.

Name	Constraint
wff1	Every message begins from exactly one object.
wff2	Every message corresponds to exactly one operation of the class of the target object.
wff3	Every operation is in exactly one class.
wff4	Every object is an instance of exactly one class.

Table 2. Well-formedness constraints used in the uncertainty change case study.

Table **??**, the constraindness level is 25%. None of points in Figs.6-7 correspond exactly to this configuration, however, if we average the time for the 25% constrainedness point for $(25, 50]$ in Fig.6 (a) with the 25% uncertainty point for $(25, 50]$ in Fig.7 (a), the result is 44.5 seconds. Comparing this to the times for uncertainty reducing changes (1) and (2) we see that these are faster than the time taken for random graphs. The similar average for Fig. 6 (b) and Fig. 7 (b) yields a time of 47 seconds. Uncertainty increasing change (3) is faster than this average as well. We conclude that the results of case study using real models are consistently faster with the results of the experiments using randomly generated models.

Change	(1)	(2)	(3)
Prop. Time (sec)	7.14	6.69	9.068

Table 3. Change propagation times of case study changes from Table **??** using the tooling described in Sec.6 .

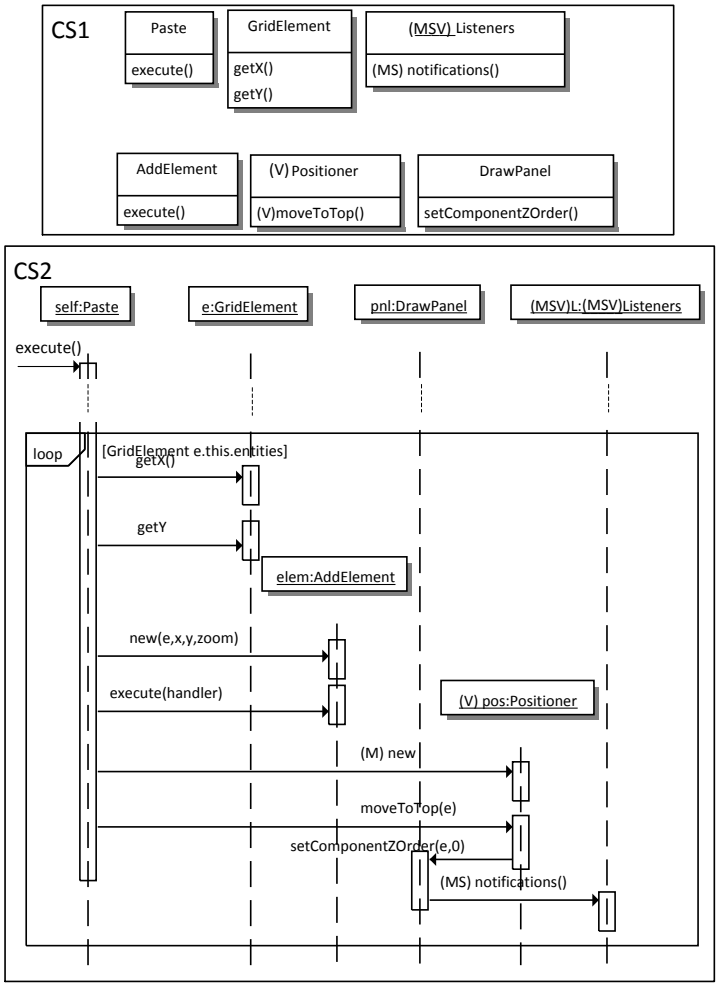


Fig. 8. A fragment of a UML class diagram and sequence diagram reverse engineered from UM-Let code and using *MAVO* annotations to show points of uncertainty in the fix to Issue 10.