Requirements Eng (2013) 18:107–128
DOI 10.1007/s00766-013-0170-y

RE 2012

Managing requirements uncertainty with partial models

Rick Salay · Marsha Chechik · Jennifer Horkoff · Alessio Di Sandro

Received: 3 November 2012 / Accepted: 26 March 2013 / Published online: 25 April 2013

Abstract Models are good at expressing information that is known but do not typically have support for representing what information a modeler does not know at a particular phase in the software development process. Partial models address this by being able to precisely represent uncertainty about model content. In previous work, we developed a general approach for defining partial models and using them to reason about design models containing uncertainty. In this paper, we show how to apply our approach to managing uncertainty in requirements by providing support for uncertainty capture, elaboration, and change. In particular, we address the problem of specifying uncertainty within a requirements model, refining a model as uncertainty reduces, providing meaning to traceability relations between models containing uncertainty, and propagating uncertainty-reducing changes between related models. We describe the implementation of uncertainty management within the Model Management Tool Framework and illustrate our approach using two examples.

Keywords Uncertainty · i* models · Partial models · Automation

R. Salay (✉) · M. Chechik · A. D. Sandro
University of Toronto, Toronto, Canada
e-mail: rsalay@cs.toronto.edu

M. Chechik
e-mail: chechik@cs.toronto.edu

A. D. Sandro
e-mail: adisandro@cs.toronto.edu

J. Horkoff
University of Trento, Trento, Italy
e-mail: horkoff@disi.initn.it

1 Introduction

Models are advocated as part of the requirements engineering (RE) process to help with elicitation, recording current understanding, communication, requirements development, and exploration of alternative high-level designs. During the process of creating RE models, it is common to have *uncertainty over the content and structure* of the models. Such uncertainties are due to gaps in domain knowledge, disagreements between stakeholders, or unresolved decisions about their needs. In the modeling process, such uncertainties often remain implicit, making it difficult to ensure that they get resolved and creating the possibility that decisions in later development phases are made using incorrect information. It is useful to explicitly express uncertainty in RE models, to facilitate its resolution through further elicitation or decision making, and to capture such uncertainty-reducing decisions and elicited information as part of the modeling process.

In previous work, the first two authors have developed a language-independent approach for expressing certain types of uncertainty [43] using *partial models*. The approach is based on allowing the modeler to use annotations with formal semantics to express her uncertainty within the model. The language-independence means that we can use it to express uncertainty within models in a uniform way at every phase of the development process. The formal nature of our approach allows the precise expression of uncertainty and provides the basis for automated tool support for activities such as reasoning with models containing uncertainty [6] and verifying that model changes reduce uncertainty [40]. While our approach is formally grounded, the formal details are hidden from the user.

1.1 Contributions

Our goal is *to enable expressing model uncertainties and uncertainty resolution for and across existing RE models, in a systematic, language-independent, formal way*. The benefit is that this allows requirements-related uncertainties to be made explicit, facilitating early resolution, and allowing any remaining uncertainty to be managed as part of later requirements and design. Specifically, in this paper, we describe support for constructing and manipulating RE models with uncertainty, as per the methodology outlined in Fig. 1.

Model capture A requirements process may make use of multiple types of models, for example, to separate "early," high-level, modeling from "later," more detailed-oriented, RE stages [51], as a form of expressive redundancy, to ensure important aspects of the domain are captured (e.g., [11, 26]), or to move toward system design (e.g., [19, 20]). While individual modeling languages may provide ways to capture uncertainty (e.g., in i*, some types of uncertainty can be captured with an "unknown" link), providing uncertainty annotations specific to each possible RE modeling notation would be cumbersome, prone to misinterpretation, and would create a cognitive barrier to learning and applying such a notation in each language. Thus, we ask: **Q1:** *How to express uncertainty over the content and structure of RE models in a language-independent way?* By answering **Q1**, we can express uncertainty over multiple types of models used in the RE process.

Model elaboration As modeling continues, model uncertainty can either increase or reduce. In this paper, we focus on the uncertainty-reducing case. Further rounds of elicitation and discussion can help resolve uncertainties leading to corresponding model refinements. Thus, we ask: **Q2:** *How to record uncertainty-reducing model refinements in RE models?*

Refinements are often caused by the availability of new information. Thus, it would be useful to optionally annotate such decisions with textual rationale. Although methods to capture rationale for RE models have been introduced (e.g., [17, 27]), they are not directly linked to the process of resolving model uncertainty. Thus, we ask: **Q3:** *How to capture rationale specific to uncertainty reduction?*

Requirements traceability is a central concern for requirements engineering and has been well studied (e.g., see [45] for an overview). Traceability relations are used to link the elements of different artifacts (including models) in a development process and can be used to express different relationships such as overlap, dependency, satisfaction, and refinement. Various approaches to RE modeling introduce traceability relations between multiple types of models used in RE (e.g., [3, 26]). Thus, we ask: **Q4:** *What is the meaning of a traceability relation between models containing uncertainty?*

Model change When modifying a model, we may wish to know whether our changes actually make the model less uncertain. Thus, we ask: **Q5:** *How to check whether model changes reduce model uncertainty?*

As models are developed, we make decisions which reduce the uncertainty in one model, and which may have effects on uncertainty in related models. Thus, we ask: **Q6:** *How can we propagate uncertainty-reducing changes to related models in a language-independent way?* Answering this question requires capturing traceability between related models, established using **Q4**.

The machinery for answering **Q1** and **Q2** is taken directly from our earlier work [40]. A conference version of this article [42] introduced the application of partial models to RE models, addressing questions **Q1**–**Q5** in an RE context [42]. This paper offers a definitive version of the previous results and expands them by developing language-independent methods for uncertainty-reducing change propagation (**Q6**), describing the implementation of uncertainty management in the Model Management Tool Framework (MMTF) [39], and reporting on the experience of applying our methodology to another example.

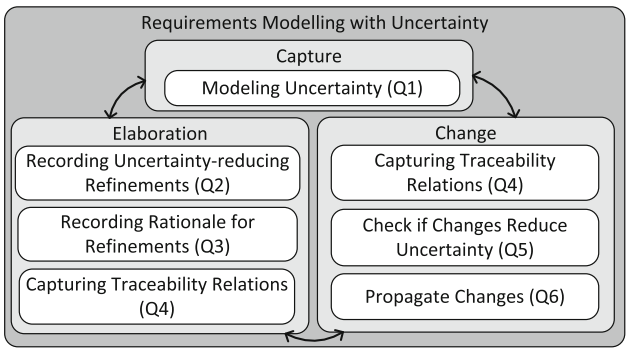


Fig. 1 Methodology for managing models containing uncertainty

1.2 Organization

The rest of this paper is organized as follows: We begin in Sect. 2 by giving a motivating example. In Sect. 3, we introduce partial models and show how to use them to express uncertainty in RE models, illustrating them on i* models and class diagrams (**Q1**). We then give background on the formalization of partial models in Sect. 4. In Sect. 5, we look at applying partial model refinement to RE models, by showing how to construct the refinement mapping (**Q2**) and capture refinement rationale (**Q3**). In Sect. 6, we show how to lift a traceability relation to the uncertain case (**Q4**). In Sect. 7, we discuss how to check whether a change made to a model constitutes a refinement (**Q5**). Section 8 provides methods for propagating uncertainty-reducing refinements (**Q6**). Section 9 describes tool support, whereas Sect. 10 describes our

experience with an extended example. We conclude the paper with a comparison between our approach and related work in Sect. 11 and a summary and discussion in Sect. 12.

2 Motivating scenario

Consider the scenario, summarized in Fig. 2, showing the movement from early to late RE models for an automated meeting scheduler.[1]

Figure 3 shows the detail of an i* model acting as the early RE model P1. The model elaborates the actors, goals, and tasks relating to different approaches to scheduling a meeting. During the construction of this model, the following uncertainties could arise: (a) gaps in the domain knowledge of the modelers ("Are there more alternative ways to organize meetings; are they quick?"); (b) disagreements between stakeholders ("Jack thinks the Meeting Initiator should pick a date, but Anne thinks it should be up to the participants"); (c) modelers wishing to indicate that they are still in the process of adding model detail ("We know the Meeting Participant needs to provide details to the scheduler, but will list these details later"). In Fig. 3, we capture uncertainties using informal text annotations.

After gaining an understanding of the domain and high-level requirements through early RE modeling, a model such as the class diagram in Fig. 4 may be developed, in order to represent relevant details of domain entities. In Fig. 2, this model is denoted by P2 and is connected to P1 via a traceability relation. Since P2 also contains uncertainty (from similar sources as described for P1), the traceability relation is defined in the context of uncertainty.

Elicitation and discussion can help resolve uncertainties leading to corresponding model refinements. *Model refinement*[2] is captured in the left side of Fig. 2, where, after the resolution of some uncertainty, the model P1 is refined into P1′ via a mapping R1. The refinement of P2, represented by P2′ via a mapping R2, can also occur, either due to uncertainty resolutions in P2 or as a result of propagating the refinement of P1 over the traceability relation.

We use this scenario throughout the paper to illustrate our solutions to questions **Q1–Q6**.

3 Expressing uncertainty in RE models

In Figs. 3 and 4, we captured uncertainties using text annotations, an approach which is neither formal nor systematic. We address question **Q1** by presenting a formal, language-independent method called *MAVO* for capturing such uncertainties. In this section, we describe the usage of *MAVO*, and in Sect. 4, we give its formalization. We begin by introducing the concepts of partial modeling.

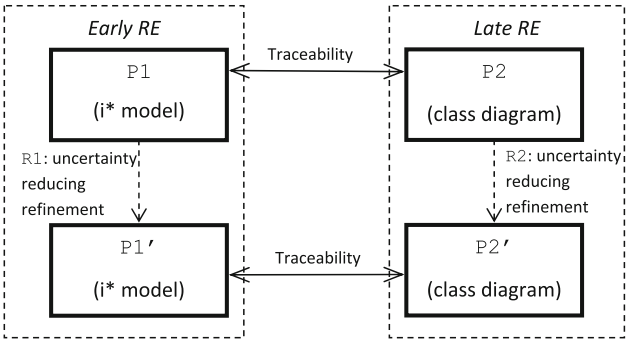


Fig. 2 Overview of a sample RE modeling process considering uncertainty

3.1 Background: partial models

To be able to add uncertainty information to existing modeling languages in a language-independent way, our approach takes as input arbitrary metamodels, referring to them as *model types*. For example, the upper part of Fig. 5 shows a (simplified) metamodel for i* models, and the lower part gives one for class diagrams. Models consist of a finite set of *atoms*, that is, the elements and relation instances of the types defined in its metamodel.

Given a model type, a *partial model of that type* represents the set of different possible concrete (i.e., non-partial) models of that type that would resolve the uncertainty represented by the partiality. More formally:

Definition 1 (*Partial model*) A partial model P consists of a base model, denoted $bs(P)$, and a set of annotations. Let T be the metamodel of $bs(P)$. Then, $[P]$ denotes the (possibly infinite) set of models of type T called the concretizations of P. P is called consistent iff it allows at least one concretization, that is, $[P] \neq \emptyset$.

Partiality is used to express uncertainty about the model until it can be resolved using *uncertainty-reducing refinement*. Refining a partial model means removing partiality as uncertainty is resolved so that the set of concretizations shrinks until, ultimately, it represents a single concrete model. In general, when a partial model P' refines another one, P, there is a mapping from $bs(P')$ to $bs(P)$ that expresses the relationship between them and thus between their concretizations.

3.2 *MAVO* annotations

We achieve language-independence by adding partiality information as annotations of a base model. For example,

[1] Parts of this scenario are adapted from [51] and have appeared in [23, 30, 42].

[2] In this paper, when we say "refinement," we always mean "uncertainty-reducing refinement."

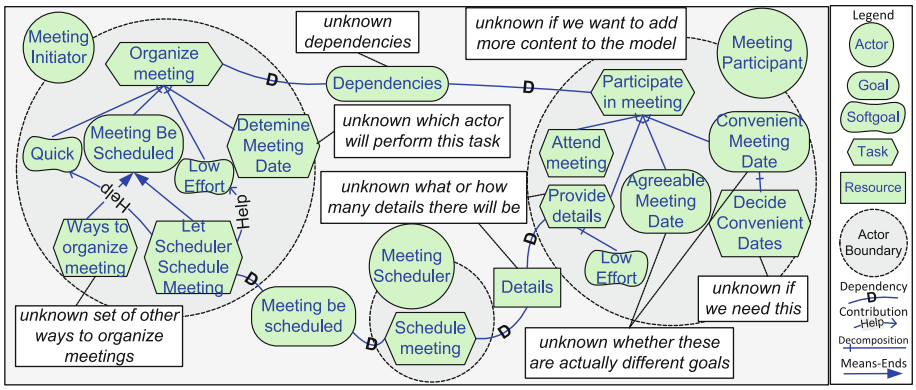


Fig. 3 An early RE diagram with annotated uncertainty for the meeting scheduler example

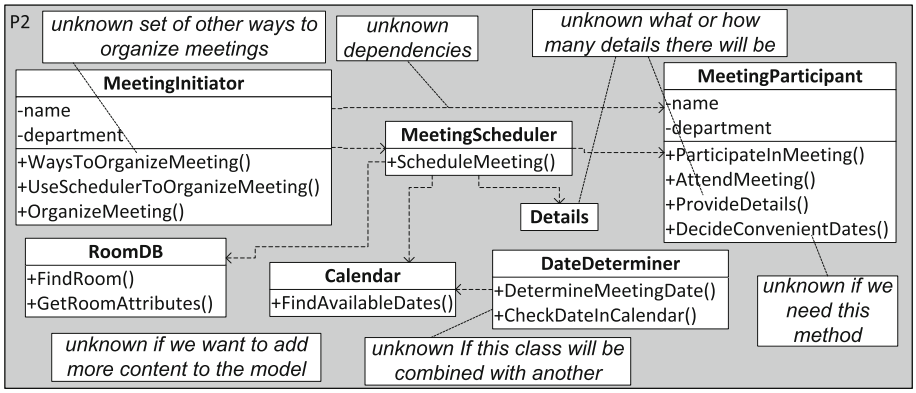


Fig. 4 A late RE diagram with uncertainty for the meeting scheduler example

models P1 and P2 in Fig. 6 show annotations on an i* base model and class diagram base model, respectively. We use four types of partiality annotations, each adding support for a different type of uncertainty in a model, as described below.

Each of the four types of annotations allows refinement as uncertainty is resolved. *May* partiality allows us to express the level of certainty we have about the presence of a particular atom in a model by annotating it with either M , to indicate that it "may exist," or E , to indicate that it "must exist." A *May* annotation is refined by changing an M to E or eliminating the atom altogether. The ground annotation E is the default if an annotation is omitted.

The *Abs partiality* allows a modeler to express uncertainty about the number of atoms in the model by letting her annotate atoms as P , representing a "particular," or S , representing a "set." A refinement of an *Abs* annotation elaborates the content of S atoms by replacing them with a set of S and P atoms. The ground annotation P is the default if an annotation is omitted.

The *Var partiality* allows a modeler to express uncertainty about distinctness of individual atoms in the model by annotating an atom to indicate whether it is a "constant" (C) or a "variable" (V). A refinement of a *Var* annotation involves reducing the set of variables by merging them with constants or other variables. The ground annotation C is the default if an annotation is omitted.

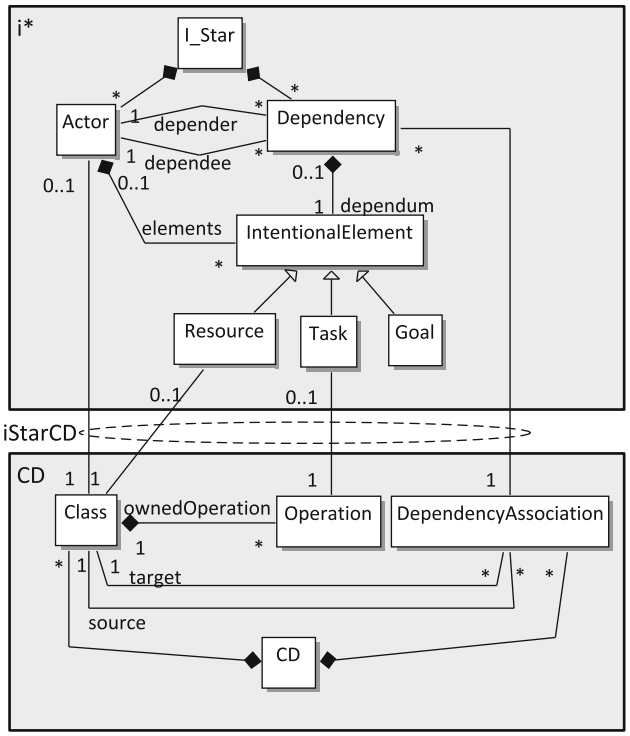
The *OW partiality* allows a modeler to explicitly state whether her model is incomplete (i.e., can be extended) (INC) or complete (COMP). In contrast to the other types of partiality discussed in this paper, here, the annotation is at the level of the entire model rather than at the level of individual atoms. The ground annotation COMP is the default if an annotation is omitted.

When these four types of partiality annotations are used together, we refer to them as *MAVO partiality*.

3.3 Using *MAVO* for expressing uncertainty

In this section, we illustrate how to use *MAVO* for encoding models with uncertainty, thereby answering **Q1**. We do this by applying it to the i* model P1 in Fig. 3 and class diagram P2 in Fig. 4 to express the same points of uncertainty as are given in an informal way using notes. Models P1 and P2 in Fig. 6 show the application of *MAVO* partiality to these models, respectively. The fact that *MAVO* annotations are used both in an i* model and in a class diagram demonstrates the *language-independence* of *MAVO* partiality and its applicability to a variety of models. We use these examples to discuss and illustrate the different situations in which to use *MAVO* annotations.

May partiality is used in P1 for the M-annotated task Decide Convenient Dates in Meeting Participant to express the fact that it is unknown whether the



Additional constraints on the traceability relation iStarCD:
(wff1) Every Task in an Actor must map to an Operation in the Class mapped to the Actor
(wff2) Every DependencyAssociation between Classes for Actors is mapped to a Dependency between the Actors

Fig. 5 The traceability relation `iStarCD` between an i* model and class diagram defined over fragments of the metamodels

task will be needed. In general, the M annotation should be used for any piece of information that we are not sure should be in the model. It can also be used when there is a known small set of alternatives for some fragment of the model, but we are not sure which is the correct one. This situation arises, for example, when multiple stakeholders provide conflicting information. Thus, the M annotation can be used as a language-independent way to *tolerate conflicts* until they are resolved.

Early in the development of a model, we may expect to have collections of atoms representing certain kinds of information but not yet know exactly what those atoms are. For example, in `P2`, the S-annotated operation `Ways to organize meeting` in class `MeetingInitiator` is used to indicate that there are some such ways but they are as yet unknown. Later, when we know more about these ways, we can refine this operation to particular tasks.

May and *Abs* are used together in model `P1` with MS-annotated task `Provide details` and resource `Details` to indicate that there may be no details at all, or there may be several. In general, one of the purposes of `Abs` partiality is to provide a way to create placeholders in the model to indicate that "further elaboration is yet to come."

Var partiality is useful when it is known that a particular fragment should be in the model but it is not yet known where it should go. For example, in Fig. 3, it is known that the task `Determine meeting date` should be in the model but not yet clear which actor should perform it. Yet, in order to achieve i* well formedness, it must be assigned to *some* actor. Without the means to express this type of uncertainty, the modeler would be forced to assign it prematurely (and perhaps, incorrectly) to one of the actors. To solve this problem, in `P1`, we put the task in the V-annotated actor `Date Determiner`, that is, something treated like a "variable" actor that, in a refinement, can potentially be equated (merged) with other variable actors and eventually be assigned to a constant actor.

Finally, it is common, during model development, to make the assumption that the model is still incomplete, that is, that other elements are yet to be added to it. This status typically changes to "complete" (if only temporarily) once some milestone, such as the release of software based on the model, is reached. *OW* partiality is used in `P1` and `P2` with the INC annotation to indicate that the models are still incomplete and can be extended. *OW* partiality defines model completeness in a very specific way: a model is complete iff it should not be extended. However, a complete model can still be changed if it has annotations representing other types of uncertainty. For example, even if `P1` was not marked with INC, it would still be allowable to replace the resource `Details` with more specific sets of detail resources since this is an *Abs* refinement. Yet, it would not be allowable to add a new goal to `Meeting Participant` because this would be a model extension.

Having described the usage of *MAVO* annotations, we next give their formal semantics.

4 Background: formalizing *MAVO*

In this section, we review the formalization of *MAVO* partiality introduced in [43] and reproduced here to help the reader understand the intuition behind the annotations as well as the computational issues of encoding and reasoning with partial models. Readers interested only in the methodological aspects of applying *MAVO* can skip this section.

To formalize *MAVO* partiality, we begin by noting that 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 atom types, and Φ is a set of

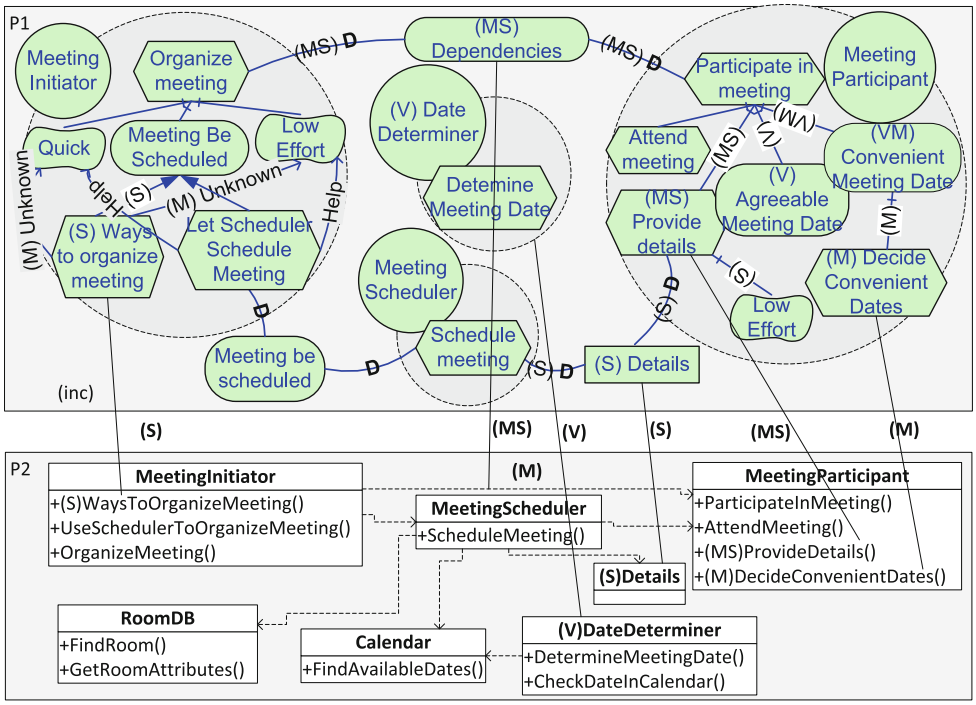


Fig. 6 The use of *MAVO* partiality annotations to express the uncertainty of models in Figs. 3 and 4 as well as the traceability relation `iStarCD(P1,P2)` between them

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 $Mod(T)$.

For example, for the fragment of the i* metamodel in Fig. 5, $\Sigma_{\texttt{iStar}}$, consists of boxes, interpreted as sorts, and associations, interpreted as predicates. $\Phi_{\texttt{iStar}}$ consists of the i* multiplicity constraints, translated to FOL, as well as additional textual well-formedness constraints (e.g., the "wffs" in Fig. 5).

Comparing Definition 2 to a metamodeling language like MOF [34], the sorts, predicates, and sentences correspond to the element classes, associations, and OCL constraints, respectively. We use FOL for representing a metamodel rather than a language like MOF since our aim here is to define a formalization rather than implementation. We discuss implementation issues in Sect. 9.

Like a metamodel, a partial model also represents a set of models and thus can also be expressed as an FOL theory. Specifically, for a partial model P, we construct a theory $FO(P)$ s.t. $Mod(FO(P)) = [P]$. To construct $FO(P)$, we first extend the theory of the metamodel with additional constraints so that it has exactly one satisfying structure—the base model of P. Then, we relax these constraints according to the annotations in P to admit more structures according to the uncertainty the annotations represent. Thus, the satisfying structures of the resulting theory are exactly the set of concretizations of P. More formally, we proceed as follows:

1. Let $M = bs(P)$ be the base model of a partial model P and define a new partial model P_M which has M as its base model and its sole concretization, that is, $bs(P_M) = M$ and $[P_M] = \{M\}$. We call P_M the *ground* model of P.
2. To construct the FOL encoding of P_M, $FO(P_M)$, we extend T to include a unary predicate for each element in M and a binary predicate for each relation instance between elements in M. Then, we add constraints to ensure that the only first-order structure that satisfies the resulting theory is M itself.
3. We construct $FO(P)$ from $FO(P_M)$ by removing constraints corresponding to the annotations in P. This constraint relaxation allows more concretizations and so represents increasing uncertainty. For example, if an atom a in P is annotated with M, then the constraint that enforces that fact that a must occur in every concretization is removed.

Illustration We illustrate the above three-step construction using the partial i* model `P1` in Fig. 6.

1. Let $\texttt{M1} = bs(\texttt{P1})$ be its base model and $P_{\texttt{M1}}$ be the corresponding ground partial model.
2. We have $FO(P_{\texttt{M1}}) = \langle \Sigma_{\texttt{iStar}} \cup \Sigma_{\texttt{M1}}, \Phi_{\texttt{iStar}} \cup \Phi_{\texttt{M1}} \rangle$ (see Definition 2), where $\Sigma_{\texttt{M1}}$ and $\Phi_{\texttt{M1}}$ are model `M1`-specific predicates and constraints, defined in Fig. 7. They extend the signature and constraints for i* models described in Fig. 5. We refer to $\Sigma_{\texttt{M1}}$ and $\Phi_{\texttt{M1}}$ as the *MAVO* predicates and constraints, respectively. The FO structures that satisfy $FO(P_{\texttt{M1}})$ are the i*

Fig. 7 The FO encoding of $P_{\texttt{M1}}$

$\Sigma_{\texttt{M1}}$ has unary predicates $\texttt{MP}(\texttt{Actor}), \texttt{AM}(\texttt{Task})), \ldots,$
and binary predicates $\texttt{MPtaskAM}(\texttt{Actor}, \texttt{Task}), \ldots$
$\Phi_{\texttt{M1}}$ contains the following sentences:
$(Complete)\ (\forall x : \texttt{Actor} \cdot \texttt{MP}(x) \vee \texttt{MS}(x) \vee \texttt{DD}(x) \vee \ldots) \wedge$
$(\forall x : \texttt{Actor}, y : \texttt{Task} \cdot \texttt{task}(x, y)$
$\Rightarrow (\texttt{MPtaskAM}(x, y) \vee \ldots)) \wedge \ldots$
BC:
$(Exists_{\texttt{MP}})\quad \exists x : \texttt{Actor} \cdot \texttt{MP}(x)$
$(Unique_{\texttt{MP}})\quad \forall x, x' : \texttt{Actor} \cdot \texttt{MP}(x) \wedge \texttt{MP}(x') \Rightarrow x = x'$
$(Distinct_{\texttt{MP}-\texttt{MS}})\quad \forall x : \texttt{Actor} \cdot \texttt{MP}(x) \Rightarrow \neg\texttt{MS}(x)$
$(Distinct_{\texttt{MP}-\texttt{DD}})\quad \forall x : \texttt{Actor} \cdot \texttt{MP}(x) \Rightarrow \neg\texttt{DD}(x)$
$(Distinct_{\texttt{MP}-\texttt{MI}})\quad \forall x : \texttt{Actor} \cdot \texttt{MP}(x) \Rightarrow \neg\texttt{MI}(x)$
similarly for all other element and relation predicates

models that satisfy the constraint set $\Phi_{\texttt{M1}}$ in Fig. 7. For conciseness, we abbreviate element names in Fig. 7, for example, `MeetingParticipant` becomes `MP`, etc. Assume N is such an i* model. The *MAVO* constraint *Complete* ensures that N contains no more elements or relation instances than `M1`. Now consider the actor `MP` in `M1`. $Exists_{\texttt{MP}}$ says that N contains at least one actor called `MP`, $Unique_{\texttt{MP}}$—that it contains no more than one actor called `MP`, and the clauses $Distinct_{\texttt{MP}-*}$—that the actor called `MP` is different from all the other actors. Similar *MAVO* constraints are given for all other elements and relation instances in `M1`. These constraints ensure that $FO(P_{\texttt{M1}})$ has exactly one concretization and thus $N = \texttt{M1}$.

3. Relaxing the *MAVO* constraints $\Phi_{\texttt{M1}}$ allows additional concretizations and represents a type of uncertainty indicated by a partiality annotation. For example, if we use the INC annotation to indicate that `M1` is incomplete, we can express this by removing the *Complete* clause from $\Phi_{\texttt{M1}}$ and thereby allow concretizations to be i* models that extend `M1`. Similarly, expressing the effect of the M, S, and V annotations for an element E corresponds to relaxing $\Phi_{\texttt{M1}}$ by removing $Exists_E$, $Unique_E$, and $Distinct_{E\text{-}*}$ clauses, respectively. For example, removing the $Distinct_{\texttt{DD}-*}$ clauses is equivalent to marking the actor `DD` with V (i.e., `DateDeterminer` may or may not be distinct from another actor).

In addition to precisely defining the semantics of a partial model, the FOL encoding provides several capabilities:

1. It allows us to do *property checking*, that is, answer questions such as "does any concretization of P have the property Q?" and "do all concretizations of P have the property Q?," where Q is expressed in FOL. The answer to the former is affirmative iff $FO(P) \wedge Q$ is satisfiable, and to the latter iff $FO(P) \wedge \neg Q$ is not satisfiable.
2. It allows us to *check the consistency* (i.e., whether it has any concretizations) of a partial model as a special case of property checking. P is consistent iff $FO(P)$ is satisfiable.
3. We can verify a refinement mapping between a pair of *MAVO* models (see Sect. 7)
4. Finally, the FO encoding allows us to *extend the expressiveness of MAVO* by augmenting the annotations using FOL to express more complex textual constraints.

5 Capturing uncertainty-reducing refinement

In this section, we address the questions **Q2** and **Q3** from Sect. 1 Specifically, we show how uncertainty reduction in an RE model is done by constructing a refinement (see Sect. 3.1) of the model including optional refinement rationale.

5.1 Constructing the refinement mapping (Q2)

As discussed in Sect. 3.1, an uncertainty-reducing refinement is given by a mapping between the atoms (elements or relationships) of two partial models. Here, we discuss the construction of such a mapping and in Sect. 7—its validation.

Given a model with uncertainty and another model which removes some of it, the refinement mapping is an artifact that expresses the way in which the elements in the two models are mapped to each other and captures the uncertainty resolution decisions made. Figure 8 gives an example of a partial i* model `P1′` refining model `P1`. As

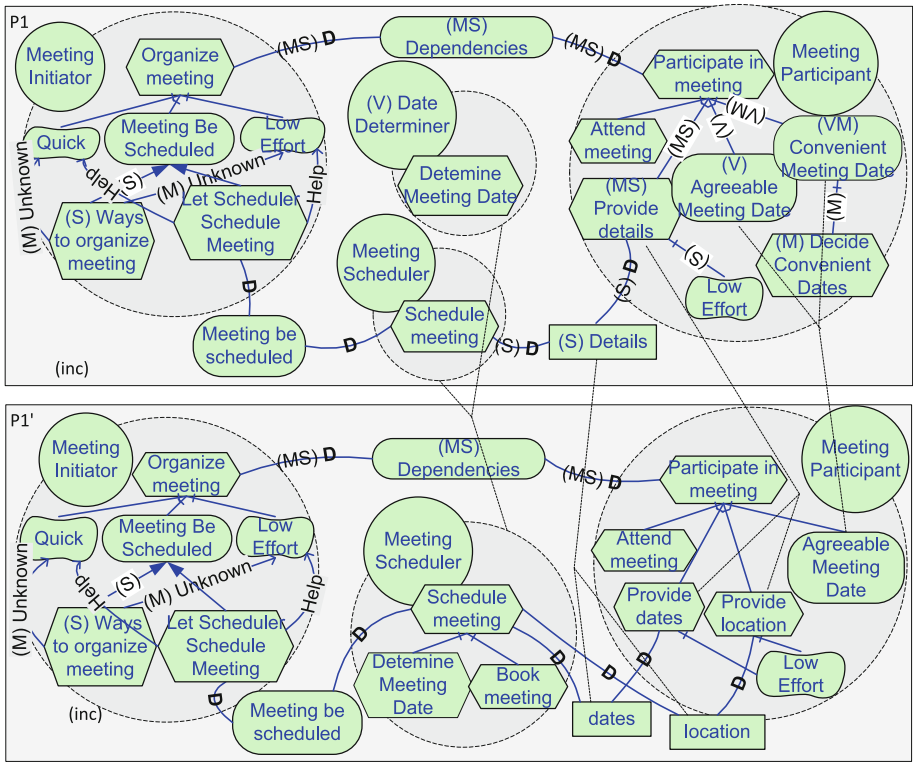


Fig. 8 An example of a uncertainty-reducing refinement of model P1 from Fig. 6

the decisions about reduction in uncertainty are made, they are captured in the refinement mapping R1. To avoid visual clutter, the figure shows only those parts of R1 where the refinement results in changes in the model.

For example, our analyst may decide which of detailed resources should exist and thus the s-annotated resource Details in P1 gets mapped to the two resources, Dates and Location, in P1′. In turn, this means that the s-annotated task Provide details in P1 is mapped to tasks Provide dates and Provide location. The analyst can also decide that the task Determine Meeting Date should be performed by actor Meeting Scheduler, which is reflected by merging the v-annotated actor Date Determiner with Meeting Scheduler so that both are mapped to the actor Meeting Scheduler in P1′. Since P1 is marked INC, more information can be added to the model so Determine Meeting Date can become a subtask of Schedule meeting and the sibling task Book meeting can be added as well. After further discussions with stakeholders, the analyst also determined that the v-annotated goals Agreeable Meeting Date and Convenient Meeting Date are not sufficiently different to keep them distinct, so they are merged and mapped to the goal Agreeable Meeting Date in P2. While doing so, she also realized that the m-annotated task Decide Convenient Dates is not needed after all, and removed it in P2.

5.2 Capturing refinement rationale (Q3)

Each of the decisions that contributed to the refinement mapping R1 constructed above could have a motivating *rationale*. For example, the analyst may decide that task Determine meeting date should be performed by actor Meeting scheduler *because* this will reduce the burden on the meeting initiator and participants. The refinement mapping artifact provides a convenient place to document such statements of rationale as part of the development process. For example, the above rationale for task Determine meeting date would be attached to the corresponding part of the mapping as a textual annotation.

This information can be used in various ways: (1) it can help recall the provenance of refinement decisions, for example, in the case of an audit; (2) it can be combined with the rationale from subsequent refinement steps to build an argument that justifies the state of the model at any point in time in order to support model comprehension; (3) it can be queried to answer questions such as "which decisions involved stakeholder X?"; (4) it can be used to "backtrack" to earlier points in the development of the model to undo decisions and explore other alternatives. For more on using rationale, please see our discussion of future work in Sect. 12

6 Traceability relations and uncertainty

In this section, we consider the problem of traceability in the presence of model uncertainty and address question **Q4** from Sect. 1 Specifically, we show how to "lift" an existing traceability relation to a partial traceability relation by both allowing uncertainty on the traceability links and in the models that are linked.

To illustrate the approach, we begin with a traceability relation `iStarCD` between i* models and class diagrams adapted from [3]. Figure 5 shows the associations between elements of the relevant fragments of the i* and the class diagram metamodels. In this case, `iStarCD` is used to show the overlap between the i* model and the class diagram. Specifically, an i* `Actor` and `Resource` correspond to a `Class`, and an i* `Task` is expressed as an `Operation` in the class corresponding to the actor of the task. These relations are constrained so that every actor, resource, and task is reflected as a class or an operation, but the class diagram can have additional classes and operations not corresponding to anything in the i* model. An i* `Dependency` between actors is expressed as a `DependencyAssociation` between classes. In this case, if there are many dependencies in a particular direction between the same actors, they map to a single dependency association between the corresponding classes.

Our goal is now to use traceability relations between meta-model elements, like `iStarCD`, to define traceability between models containing uncertainty. We can apply *MAVO* annotations to `iStarCD` just like we would to atoms of a model! Fig. 6 shows the partial i* model `P1` from Fig. 8 mapped to the partial class diagram `P2` using the resulting partial traceability relation `iStarCD(P1,P2)`. Note that each trace link and each of its endpoints can be annotated with *MAVO* annotations. To reduce visual clutter, Fig. 6 only shows links with annotations. For example, the actor `MeetingParticipant` is linked to the class `MeetingParticipant` and its M-annotated task `Decide Convenient Dates` is linked via an M-annotated link to an operation by the same name. This says that although the existence of the task in the goal model is uncertain, its presence as an operation in the late requirements is certain.

Furthermore, since the existence of the task is uncertain, the existence of the traceability link should be uncertain as well. This is because the uncertainty of the task is affected by the uncertainty of the link. Making the existence of the link certain, for example, by annotating it with E, means that it occurs in every concretization. This would force the task to also be present in every concretization since a link cannot occur without its endpoints. Thus, if the task is present in every concretization, its existence is made certain despite the fact that it is annotated with M.

To address this problem, we state the following well-formedness rule for traceability relations:

Rule 1 *For each traceability link $\rho(a, b)$, the annotation of ρ should not be more refined than the annotations of a and b.*

The traceability relation in Fig. 6 satisfies this rule.

Recall that refinement of individual *MAVO* annotations was defined in Sect. 4 and constitutes a constant-time check for each of a and b, making checking of the rule efficient.

The partiality annotations on the traceability links occur not only because of Rule 1 but also to express specific uncertainties about the traceability relationship itself. For example, suppose that the analyst is unsure whether to map the actor `Meeting scheduler` in the model in Fig. 6 to the class `Meeting scheduler` or to the class `Calendar` (i.e., to integrate the scheduling functionality into the calendar). This can be expressed by mapping the actor to both classes and annotating the links with M. Note that due to the multiplicity constraints on the actor-class links, it can be mapped only to one class in each concretization.

In summary, *MAVO* annotations can be used with traceability relations by annotating the individual traceability links. Furthermore, the annotations on a link can be both due to the annotations on its endpoint atoms (via Rule 1) or to the intentions of the analyst.

7 Checking uncertainty-reducing refinement

In this section, we address the question **Q5** from Sect. 1 by showing how to verify that a potential refinement mapping, such as the one in Fig. 8, actually reduces uncertainty.

As discussed in Sect. 3.1, we give semantics of uncertainty-reducing refinement of a model in terms of reducing the set of concretizations it has while making sure at least one concretization remains. This is expressed more formally by the following refinement conditions [40]:

Definition 3 (***MAVO* refinement conditions**): Let MAVO models P and P' be given. P' *refines* P iff there exists a mapping R s.t. the following conditions hold

(Ref1) P' must be a consistent partial model.
(Ref2) Every concretization of P' is also one of P.

Condition Ref1 ensures that P' has at least one concretization (see Definition 1). R is then called a *refinement mapping*.

Verifying a *MAVO* refinement requires showing that conditions Ref1 and Ref2 in Definition 3 hold. In general, this can be done using the FO encoding of the two models, and we have implemented prototype tool support [40] on

top of Alloy for performing these checks using SAT solving. The approach can be applied directly for checking refinement of RE models. While this approach is powerful and can be used with general uncertainty-representing models which may include additional textual constraints augmenting *MAVO* annotations, it has computational limitations and typically is not applicable to large (even in the RE sense!) models. Below we discuss ways to improve performance in some typical situations.

Checking Ref1 Checking the condition Ref1 requires showing that P' has at least one concretization, that is, that $FO(P')$ is satisfiable. This can be done using a SAT solver. However, in some cases, the process can be simplified by taking advantage of the following property discussed in [40]: when the base model of P' (i.e., P' with all the annotations removed) is a well-formed model, then it is also a concretization of P', and thus the refinement condition Ref1 is satisfied. Of course, it is a sufficient but not necessary condition: the base model of a *MAVO* model may not be well formed but can still have concretizations. In this case, the FO encoding for checking Ref1 is required.

Checking Ref2 When we limit ourselves to just using *MAVO* annotations, we can simplify checking the refinement condition Ref2 by defining syntactic constraints (i.e., necessary conditions) on the annotations that follow from the FO condition.

Figure 9 summarizes these constraints. Each of the four columns indicates a different case (case number is given at the top) in the refinement mapping, and the sentences in the lower part of each case give the constraints on the *MAVO* annotations for the atoms of that case. A valid refinement must satisfy all of these constraints. The sentences refer to the full set of *MAVO* annotations described in Sect. 3.2 (M/E; S/P; V/C; INC/COMP), including those assumed by default when the annotation for a partiality type is omitted.

For example, case (1) occurs when an atom a of model P is refined to a set of atoms $a'_1,\ldots, a'_n$ of P'. The first sentence says that if a is annotated with E (i.e., it is not M), then at least one of the atoms a'_i must also be annotated with E. Thus, if a exists and it is refined to the set of a'_is then at least one of these should exist. The second sentence says that if a is a particular (i.e., *not* a set), then there can only be one a'_i and it too must be a particular. The third sentence says that if a is a constant (and thus cannot merge with any other atom), then all a'_is it refines must also be constants. Case (2) says that if a is not propagated into the new model, then it must have been annotated with M. Case (3) states that if multiple a_is in P are mapped into a single a' in P', then if any of the a_is had definite information, or were particular, or were a constant, then a' must satisfy these conditions as well. The last sentence in case (3) says

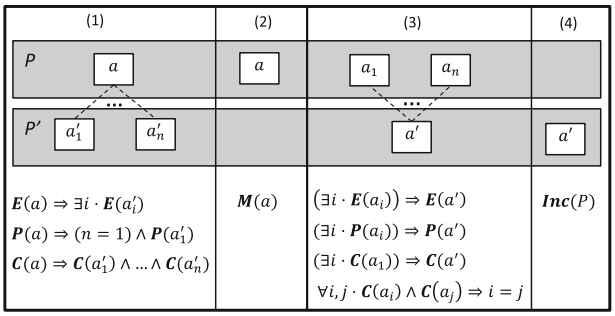


Fig. 9 Summary of the constraints on annotations of model elements across a *MAVO* refinement mapping

that at most one of the a_is could be a constant. Finally, if a new atom, not mapped to anything in P, appears in P' (case (4)), then P could not be complete.

The procedure for applying the constraints in Fig. 9 to verifying condition Ref2 of a potential refinement involves first iterating through the atoms of P and checking cases (1) and (2), and then iterating through the atoms of P' and checking cases (3) and (4). This procedure has complexity $O(|P| \times |P'|)$, which allows us to conclude that this is a scalable strategy for checking the condition Ref2. This contrasts with the SAT-based approach which has exponential complexity.

Illustration We illustrate the above techniques for checking refinement conditions by applying them to the models in Fig. 8. The base model of `P1′` is a well-formed i* model so we can conclude, according to the above discussion, that `P1′` satisfies the refinement condition Ref1.

We can also apply the constraints in Fig. 9 to check Ref2. The refinements of the resource `Details` and the task `Provide details` are examples of case (1). The merges of actors `Meeting scheduler` with `Date Determiner` and goals `Agreeable Meeting Date` with `Convenient Meeting Date` are examples of case (3). The addition of tasks `Schedule meeting` and `Book meeting` is an example of case (4). Finally, the removal of `Decide Convenient Dates` is an example of case (2).

Since both Ref1 and Ref2 have been shown to hold, we conclude that the mapping in Fig. 8 is a valid *MAVO* refinement which reduces uncertainty.

8 Uncertainty-reducing change propagation

Changes to classical models typically affect their content, and *change propagation* [28] deals with determining how this affects related artifacts. In the presence of uncertainty, changes can also affect the level of uncertainty and can then be propagated to related models across a traceability

relationship. Figure 2 illustrates such a scenario: refining a model P1 via an uncertainty-reducing refinement R1 produces a model P1′. This change can be propagated across the traceability relation to force a corresponding refinement P2′ of model P2 via the refinement mapping R2. In this section, we examine the details of this scenario, answering the question **Q6**.

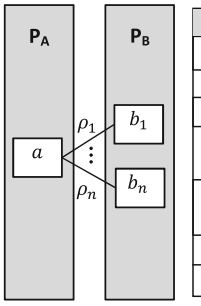
Consider the example in Fig. 6. The s-annotated resource Details in P1 is linked to the s-annotated class Details in P2. Due to the "0..1–1" multiplicity constraint on the resource-class link type in Fig. 5, each "detail" resource in a concretization of P1 must map to one "detail" class in a corresponding concretization of P2. Refining P2 to allow only a single detail class by removing the s annotation from class Details causes the multiplicity constraint to *force* every corresponding concretization of P1 to have a single "detail" resource. This would suggest that the Details resource in P1 should also be refined and lose its s annotation since there is no concretization that can have more than one "detail" resource in it. Thus, we have propagated an uncertainty-reducing refinement of P2 by applying a forced refinement to P1.

In this example, the refinement of P1 was forced because removing the s annotation *did not reduce its set of concretizations*—it was already constrained to a smaller set by P2 via the traceability relation. A potential algorithm for determining such forced refinements would do a breadth-first search through possible refinements until only concretization-reducing ones remain. While such an algorithm is general because it can be applied to any model type regardless of its well-formedness constraints, it is computationally expensive.

A much more tractable approximation of this algorithm can be defined if we focus on the traceability relations and consider the forced refinement rules for particular well-formedness constraints on such relations. For example, Fig. 10 lists rules that apply if we consider only multiplicity constraints and restrict these to "0..1," "1..1," "0..*" or "1..*" on either end of the relation. In this figure, models P_A and P_B are linked so that an atom a in P_A is connected via links $\rho_1, \ldots, \rho_n$ to atoms $b_1, \ldots, b_n$ in P_B. Assume that P_A has just been refined. Rules (R1–R7) on the right-hand side of the figure list some of the ways this change can affect the traceability links and the atoms of P_B. Each rule consists of a rewrite rule with a multiplicity constraint (column Multiplicity), where L_a, U_a, L_b, and U_b are the lower and upper multiplicity values on the "a" and "b" ends of the traceability relation, respectively. A rule means that if the multiplicity constraint holds and so does the left-hand side of the rule, then the traceability relation and the atoms of P_B should be changed to satisfy the right-hand side. For example, rule R1 applies for any of the four multiplicities on either end of the relation and says that if a link ρ_i is E-annotated, so must be b_i. This rule captures the fact that when the link exists in all concretizations—and thus its endpoints must as well since we cannot have a link without its endpoints. Thus, b_i should not be M-annotated.

Rule R4 applies when P_B is incomplete, a is not linked to b_i ($n = 0$) even though it should (since $L_b = 1$). In this case, we must create a new V-annotated link and element of P_B. This forces a to be linked to some atom in P_B in every concretization. Rule R7 applies when the multiplicity constraint states that a can link to at most one b_i ($U_b = 1$), but there are at least two links that exist, one of which is V-annotated. To conform to the multiplicity constraint, we must merge these links. If there are more than two of them, this rule can be applied repeatedly until only one remains.

Termination To compute the propagated change, the rules are invoked repeatedly until they no longer apply. For example, rule R1 is applied to each traceability link that does not have an M annotation by removing any M annotation on its endpoint, etc. To show termination, we observe that Rule R4 applies exactly once for each atom in P_A that is not linked (i.e., an orphan) to an atom in P_B and links it (so it will no longer be applicable). Furthermore, no rule creates orphan atoms. Thus, the action of rule R4 terminates after all such orphan atoms of P_A are linked. All other rules strictly reduce the number of annotations in P_B since each rule application removes an annotation. Thus, since there is only a finite number of annotations in P_B, the application of these rules must terminate.



Id	Multiplicity	Forced Refinement Rule
R1		$\boldsymbol{M}(b_i) \wedge \boldsymbol{E}(\rho_i) \dashrightarrow Set\ \boldsymbol{E}(b_i)$
R2		$\boldsymbol{S}(\rho_i) \wedge \boldsymbol{P}(a) \wedge \boldsymbol{P}(b_i) \dashrightarrow Set\ \boldsymbol{P}(\rho_i)$
R3		$\boldsymbol{V}(\rho_i) \wedge \boldsymbol{C}(a) \wedge \boldsymbol{C}(b_i) \dashrightarrow Set\ \boldsymbol{C}(\rho_i)$
R4	$L_b = 1$, $n = 0$	$\boldsymbol{Inc}(P_B) \dashrightarrow Add(\rho_1), Add(b_1), Set\ \boldsymbol{V}(\rho_1), Set\ \boldsymbol{V}(b_1)$
R5	$L_b = 1$, $n = 1$	$\boldsymbol{M}(\rho_i) \wedge \boldsymbol{C}(\rho_i) \wedge \boldsymbol{E}(a) \dashrightarrow Set\ \boldsymbol{E}(\rho_i)$
R6	$U_b = 1$	$\boldsymbol{S}(\rho_i) \wedge \boldsymbol{P}(a) \dashrightarrow Set\ \boldsymbol{P}(\rho_i)$
R7	$U_b = 1$	$\boldsymbol{E}(\rho_i) \wedge \boldsymbol{E}(\rho_j) \wedge \boldsymbol{V}(\rho_i) \wedge \boldsymbol{V}(b_i) \dashrightarrow Merge(\rho_i, \rho_j)$

Fig. 10 The forced refinement rules for any relation with multiplicity lower bound $L \in \{0, 1\}$ and upper bound $U \in \{1, *\}$ on each end of the relation

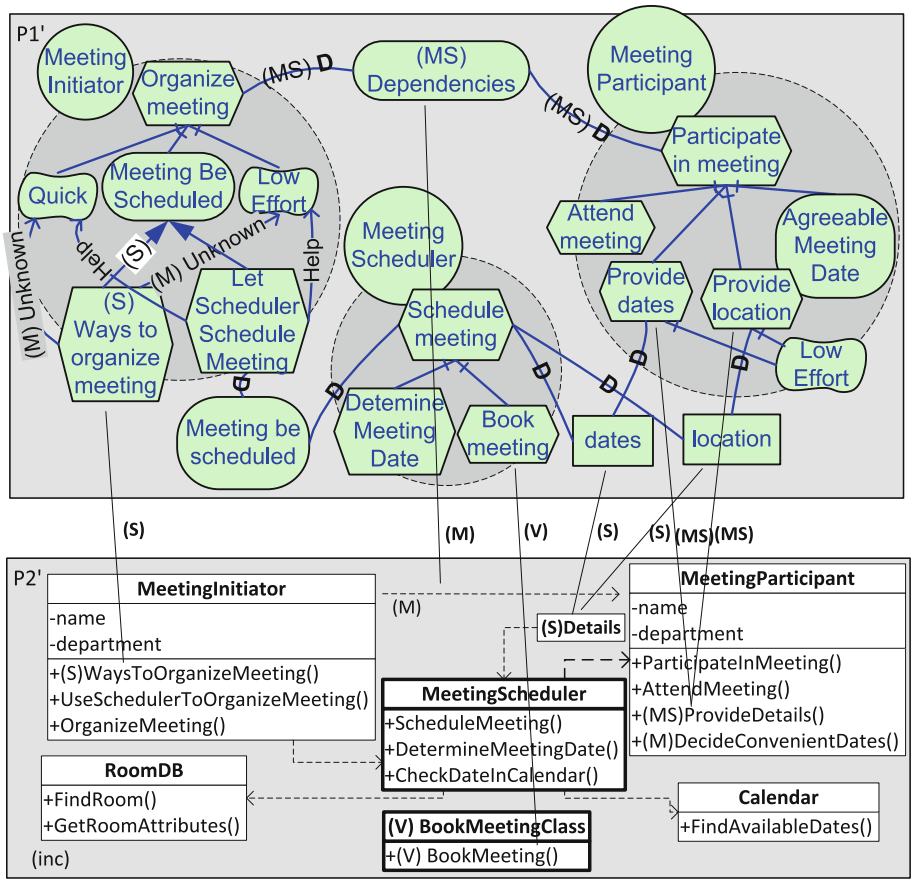


Fig. 11 The result of propagating the refinement in Fig. 8 across the traceability relation in Fig. 6

Illustration We illustrate uncertainty change propagation and the use of rules R1–R7 on the models in Fig. 6. Assume that P1 is refined to P1′ as shown in Fig. 8 and discussed in Sect. 5. Figure 11 shows the result of propagating this refinement to model P2, producing P2′.[3] Such changes are highlighted with a dark black outline.

In P1′, the decision to merge the v-annotated actor DateDeterminer with MeetingScheduler forces a corresponding merge between classes in P2′ (by rule R7). Adding the task BookMeeting in MeetingScheduler forces adding a corresponding v-annotated operation by rule R4. However, the rules do not specify that the new operation should be put into the MeetingScheduler class; instead, a new v-annotated "dummy" class BookMeetingClass is created for it since every operation must be in some class. This shows the imprecision of the propagation rules in Fig. 10 since only multiplicity constraints are taken into account.

Some refinements do not propagate at all. For example, the decision to remove the task Decide Convenient Dates in MeetingParticipant does not force the removal of the corresponding operation in P2′ because the multiplicity constraint for the task-operation link type in Fig. 5 allows the class diagram to have operations that are not in the i* model. Another refinement that is not propagated is the merge of goals Convenient Meeting Date and Agreeable Meeting Date since the given traceability relation does not map goals. Finally, although the s-annotated Details resource is refined into two—Dates and Location—this change is not propagated to split the corresponding class Details.

Figure 12 shows the result of applying the change propagation rules to a further refinement of the i* model P1 in Fig. 11. For example, it is decided that Meeting Initiator depends only on MeetingParticipant for Attends Meeting, refining (MS) Dependencies in the model in Fig. 6 and providing the rationale "the meeting scheduler handles all other dependencies." Similarly, it is determined that the only other way to schedule a meeting in this context is manually (a further alternative may have been to delegate the task to an administrator). This option has negative effects on the soft goals of Low Effort and Quick. The refinement of the dependency uses rule R5, removing the M annotation from the corresponding traceability relation, and then rule R1, removing the M annotation from the dependency association. As with

[3] Note that here we are propagating a refinement of P1 to P2, whereas earlier we gave an example of propagating a refinement of P2 to P1. Thus, the propagation rules can be used in either direction.

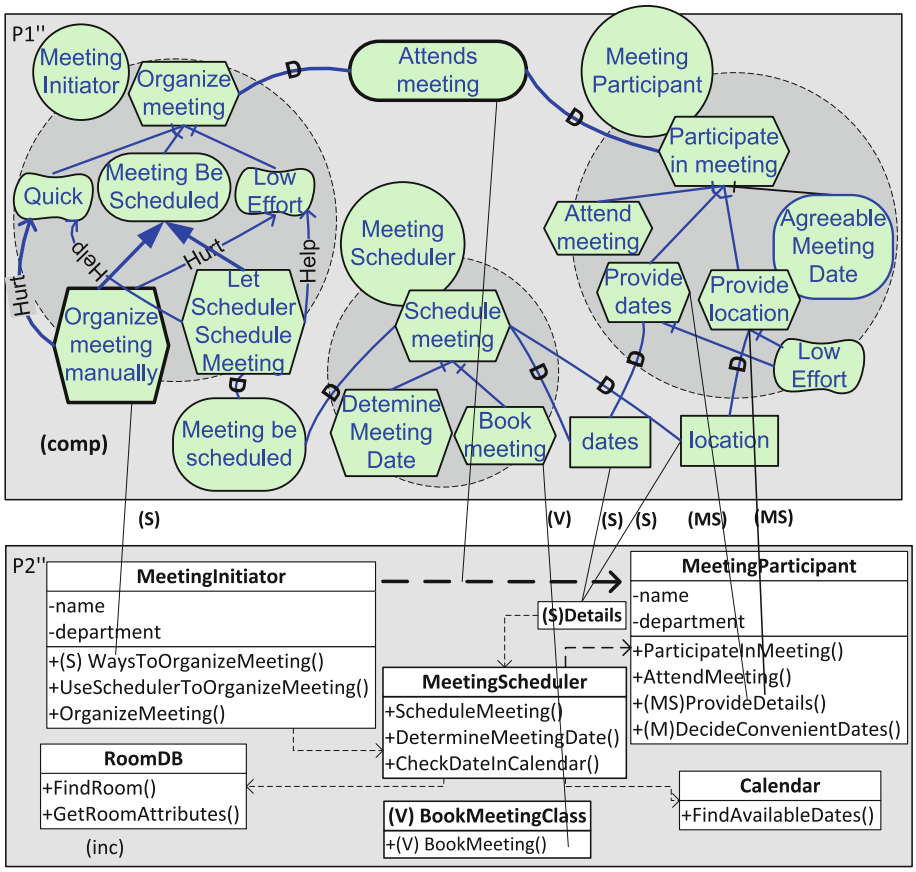


Fig. 12 Second set of refinements for the Meeting Scheduler example with the corresponding class diagram change propagation

the previous refinement, not all changes are propagated. As i* links are not mapped to class diagram elements, refinements to these elements do not affect the class diagram. Thus, while the i* model gets refined to a concrete one, uncertainties in the class diagram remain to be resolved.

9 Tool support

In this section, we describe a prototype tool for supporting uncertainty management within RE described in this paper. Our prototype tool for supporting uncertainty management in RE is built on top of the Model Management Tool Framework (MMTF) [39]—an Eclipse-based plug-in infrastructure for creating model management tools. Figure 13 shows the MMTF architecture. New model types, model relationship types, editors, and operators can be added to MMTF as plug-ins. The functionality of MMTF is built on top of Eclipse modeling services that provide support for visualizing and storing models. MMTF provides a runtime environment in which users can create models, connect them with model relationships, and manipulate them using model operators within a *Model Interconnection Diagram* (MID). A MID is a model with boxes representing models and thick arrows representing

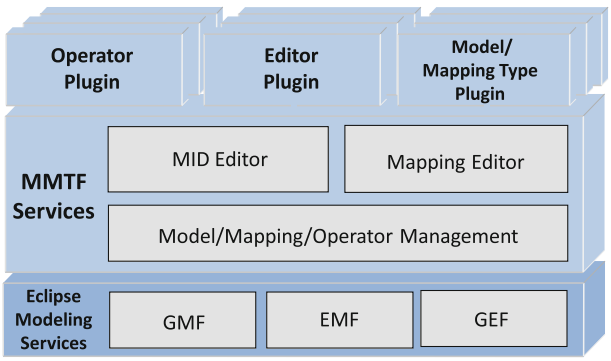


Fig. 13 The MMTF Architecture

relationships between them. The left side of the screenshot in Fig. 15 shows a MID for the meeting scheduler example. Double-clicking on any of the boxes or arrows brings up the corresponding editor to allow the user to change the content of the model or relationship. A MID is also an interface for invoking an operator on models/relationships.

We describe implementing *MAVO* support on top of MMTF below.

Adding annotation support (Q1) *MAVO* is a modeling language-independent approach for expressing uncertainty. While in this paper we exemplify the use of *MAVO* in RE

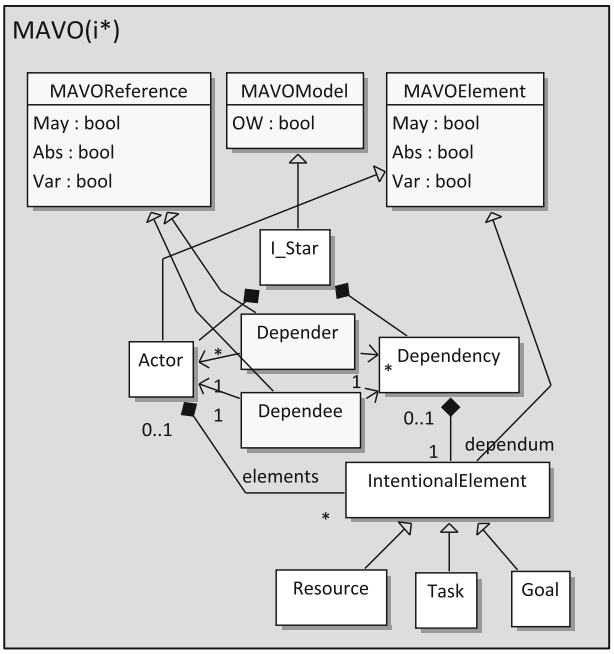


Fig. 14 The *MAVO*-transformed metamodel fragment of i* from Fig. 5

models using i* and class diagrams, we automate its incorporation into *arbitrary* existing modeling languages. Automation not only reduces effort, it also ensures that uncertainty is applied consistently across the different models of the development life cycle.

We begin by describing a transformation of the metamodel of a modeling language to add *MAVO* support, illustrating it using the fragment of the i* metamodel in Fig. 5. Figure 14 shows the result of the transformation. The changed elements are darkened. We add three abstract element classes, `MAVOElement`, `MAVOReference`, and `MAVOModel`, which contain attributes required to store the *MAVO* annotation information. All elements visible in the concrete syntax inherit from `MAVOElement`, and hence all can carry annotation information. Similarly, the element type representing the model as a whole, in this case, `I_Star`, inherits from `MAVOModel` and hence can carry the INC annotation. Finally, the metamodel for relations visible in the concrete syntax (e.g., `depender` and `dependee`) is also changed to carry *MAVO* attributes, by transforming these into new element types that inherit from `MAVOReference`.

This transformation can be applied in a similar fashion to any metamodel, and future work will address integrating these changes with existing model editors.

Traceability relation (Q4) An MMTF relationship type consists of a set of link types and a set of well-formedness constraints they must satisfy. Such constraints can be implemented using a variety of languages, for example, using Java or Object Constraint Language (OCL). Thus, any traceability relationship metamodel can be implemented within MMTF. The one described in Fig. 5 is defined as the type `IStarCDRel`. The content of such relationships can be modified with a generic mapping editor provided. The right side of the screenshot in Fig. 15 shows an instance `Tr` of the relation type `IStarCDRel` used in the meeting scheduler MID on the left side of the figure, opened in the mapping editor. Links `Trace3` and `Trace5` show the *MAVO* annotation support.

Refinement relation (Q2) and checking uncertainty-reducing refinement (Q5) The *MAVO* refinement mapping is a set of links between the atoms of two *MAVO* models and thus it is also implemented as an MMTF relationship type, called `MAVORefinementRel`. As discussed in Sect. 7, the well-formedness constraints in Fig. 9

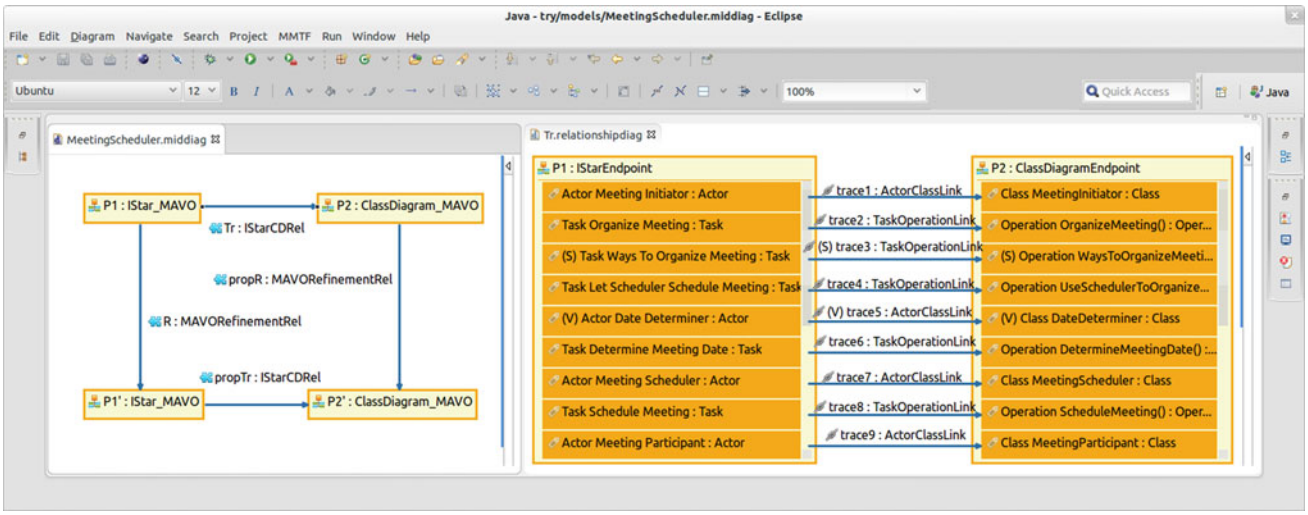


Fig. 15 MMTF screenshot showing the meeting scheduler MID and a detail of a traceability relation between `P1` and `P2`

Fig. 16 A fragment of the OCL used to implement the case (1) constraints in Fig. 9

```
..(
    origModelElemEndpoints->size() = 1 and (
      (not origModelElemEndpoints->at(1).target.may implies
        refinedModelElemEndpoints->exists(not target.may)) and
      (not origModelElemEndpoints->at(1).target.set implies
        (refinedModelElemEndpoints->size() = 1 and
          refinedModelElemEndpoints->exists(not target.set))) and
      (not origModelElemEndpoints->at(1).target.var implies
        refinedModelElemEndpoints->forAll(not target.var))
    )
  ) or ...
```

have to hold for an uncertainty-reducing refinement mapping to be valid. Therefore, the `MAVORefinementRel` relationship type contains an OCL implementation of such constraints. Figure 16 shows the fragment of the OCL used to implement case (1) of Fig. 9 (the case where a single atom is refined to a set of atoms). MMTF provides support for validation checking, that is, checking that the specific refinement relationship instance is conformant to such OCL constraints. Note that this is not a satisfiability problem; instead, the instance of refinement relation provides a finite and decidable context of evaluation for the constraints. The MID on the left side of Fig. 15 shows two instances of the relationship type `MAVORefinementRel`, the refinement `R` from i* model `P1` to `P1′` and the refinement `propR` from class diagram `P2` to `P2′`.

The change propagation operator (Q6) The change propagation functionality described in Sect. 8 is implemented as an MMTF operator that applies the rules in Fig. 10. The MID in Fig. 17 shows a screenshot of the operator being applied to the meeting scheduler scenario. In general, the input to the operator is a pair of models P_1 and P_2 connected by a traceability relation *TT*, a third model P'_1 representing the refined version of P_1, and a refinement mapping *RR* as defined in Sect. 5 The output consists of the propagated model `prop`P_2 as well as the corresponding traceability relationship `prop`*TT* and refinement mapping `prop`*RR*. The MID on the left side of Figure 15 shows the result of applying the change propagation operator in Fig. 17.

10 Applying *MAVO*

In this section, we describe an application of the *MAVO* framework and tool support to a larger, more complex example arising from the requirements analysis of a real system. We report results and then discuss our experiences in light of our research questions. This exercise, along with our experiences in applying *MAVO* to the Meeting Scheduler example, increases our confidence in the utility of *MAVO* in practice. Practical implications and lessons learned (discussed in Sect. 10.3) suggest future framework and implementation improvements.

10.1 Inflo

Inflo is an online modeling application allowing users to create collaborative graphs, for example, to discuss sources and calculations for carbon footprints [24]. As envisioned,

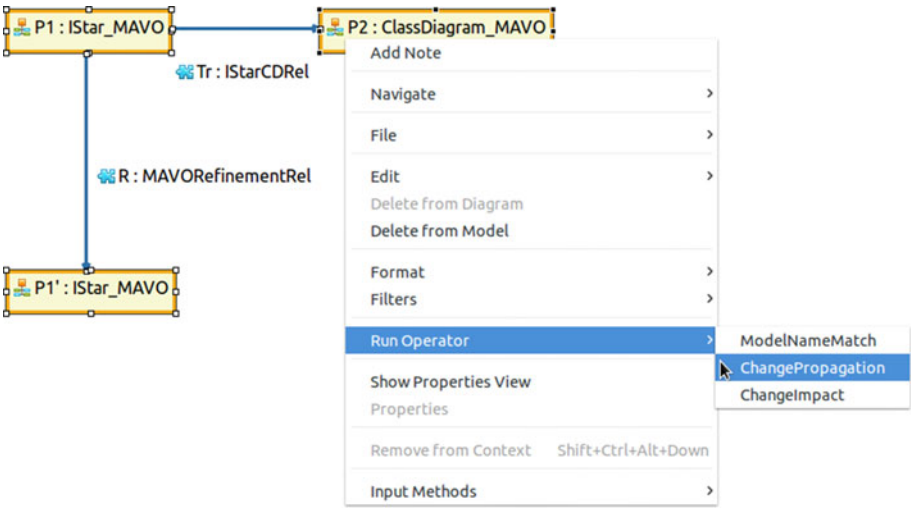


Fig. 17 MMTF screenshot showing the application of the change propagation operator

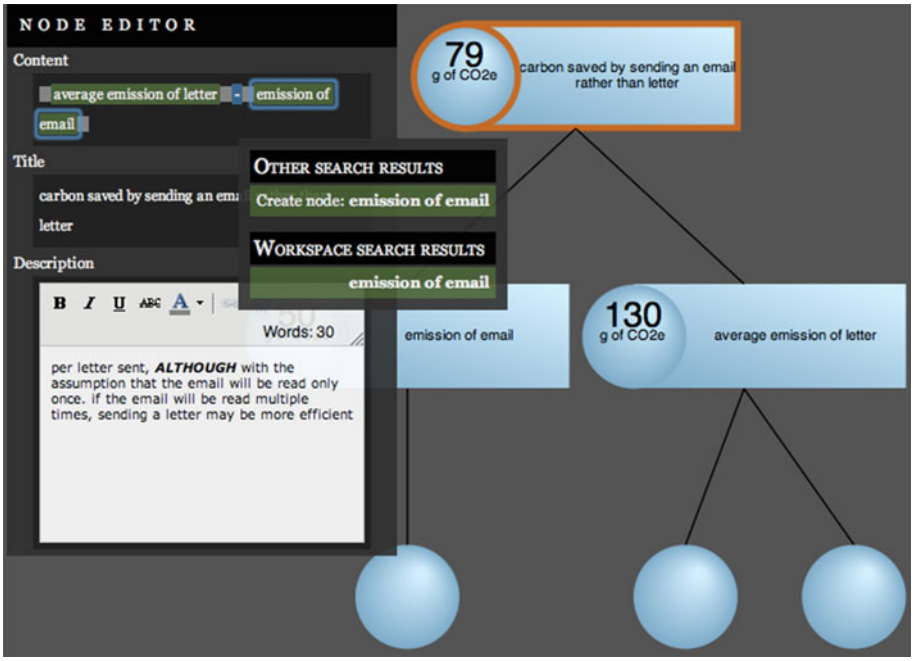


Fig. 18 Screenshot of the Inflo application

users would produce visualizations of calculations, linking them to available data sources, while other users would discuss or dispute calculations or sources. A screenshot of the Inflo application is shown in Fig. 18. An i* model of the Inflo system has been produced by the third author together with Inflo creators as part of a case study for testing interactive i* analysis [13] and consists of 12 actors, 103 elements, and 145 links.

To test the *MAVO* capabilities described in this paper, we have purposely selected a subset of the original i* model which contained some uncertainty, including the major actors and top-level elements in order to create a connected and well-formed submodel. The resulting submodel contains 5 actors, 24 i* elements, and 22 links and captures the dependencies between the `Inflo` system (expressed as a separate agent), `InfloUser`, and `Inflo-Managers/Editors` (see Fig. 19). For example, `Inflo` depends on `Managers/Editors` to moderate online discussions about Inflo graphs.

10.2 Experience

We describe our experience with capturing, analyzing, and refining uncertainty in the Inflo system. The quantitative aspects of our experience are summarized in Table 1. The table also shows similar statistics collected for the Meeting Scheduler example used throughout this paper. Refinement checks and change propagation were executed using the MMTF implementation for both the Inflo and Meeting Scheduler examples.

Q1 To express uncertainty over the Inflo i* model, we needed to add several additional links and elements capturing model alternatives. For example, the relationship between the moderation options of `InfloManagers/Editors` was not clear: was the task `Use Automated reputation system` an alternative to `Extensive moderation` in achieving the `Moderate inflo` goal, or was `Use Automated reputation system` part of an additional option, `less extensive moderation`? To express this uncertainty, we added a new M V-annotated task in `InfloManager/Editors` (see the top actor in Fig. 19). Overall, we added 5 elements and 13 links to our subset model, resulting in 29 element and 35 links, 21 of which are annotated with 24 *MAVO* annotations (see Table 1). We then created a corresponding class diagram, with 10 classes and 12 dependencies, as shown in the lower part of Fig. 19.

Q2 After expressing initial uncertainties, the i* model went through two rounds of refinement, referred to as $P1'$ and $P1''$. For example, in $P1'$ it was decided that `Inflo-Managers/Editors` had the option of extensive or less extensive moderation, with the automated reputation system used as part of the latter. As part of this change, we split the S-annotated task `Moderationtasks` in the `Inflo` agent into several moderation tasks, such as `edit post`, `delete post`. Table 1 summarizes the number of *MAVO* labels refined in each refinement round.

Q3 To document non-obvious reasoning behind model refinements, we provided refinement rationale. For example, when reducing uncertainty concerning moderation, we recorded the rationale statement "Can moderate extensively without an automated system or less extensively, relying on the automated system," explaining the alternative configurations selected in the model. In this case, we

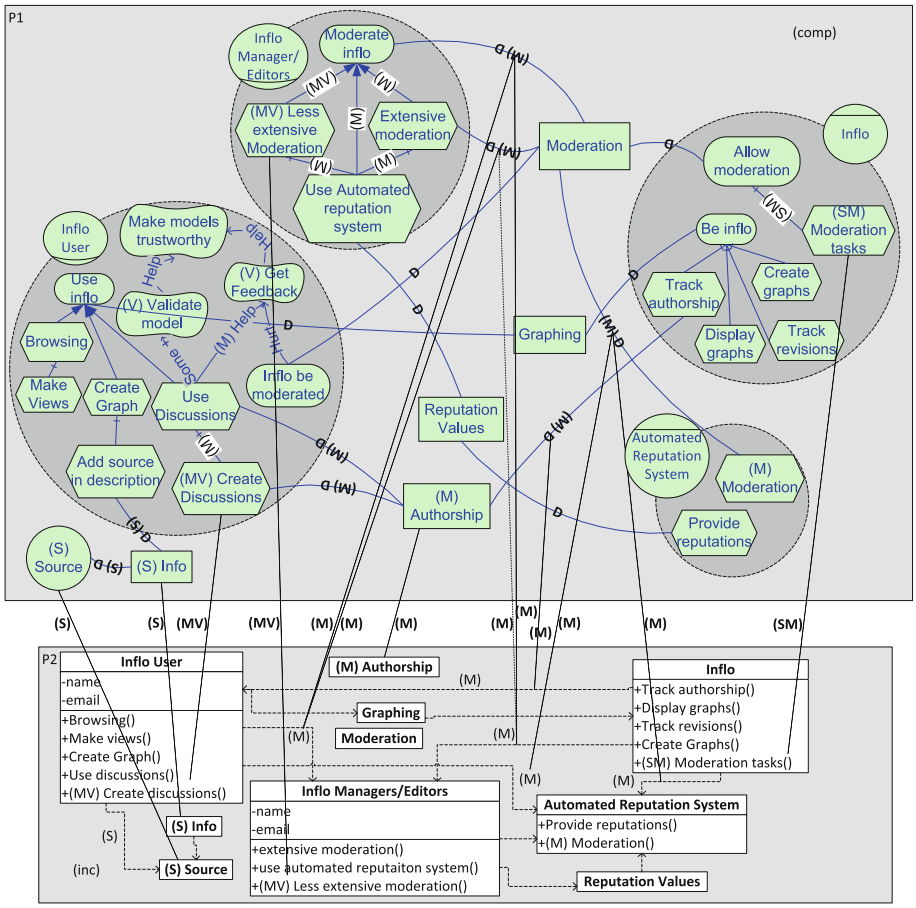


Fig. 19 Inflo model and corresponding class diagram containing *MAVO*-annotated uncertainty

Table 1 Statistics for the Meeting Scheduler and Inflo system

Examples	Initial MAVO–annotated i* model size			Initial i* MAVO labels			Num. MAVO labels refined		Num. rationale statements		Num. changes with rationale		Num. changes should propagate		Num. changes propagated	
	Actors	Elmts.	Links	M	S	V	P1′	P1″	P1′	P1″	P1′	P1″	P1′	P1″	P1′	P1″
Meeting scheduler	4	18	21	8	10	3	11	10	0	2	0	8	4	3	2	1
Inflo	5	29	35	19	6	5	16	14	2	5	13	11	5	4	4	1

found it helpful to apply the rationale to the entire set of changes related to moderation. Table 1 summarizes the number of rationale statements in each refinement as well as the number of changes covered by rationale statements, for example, in $P1'$, 2 rationale statements covered 13 of 16 refined annotations.

Q4–Q5 We checked the refinements made to the model against the rules in Fig. 9. In this particular example, only constraints (1) and (2) were used. We then matched uncertainty in the class diagram to a subset of uncertainty of the i* diagram using traceability links.

Q6 Where possible, the changes to the i* model made in each refinement were propagated to the corresponding class diagram using the rules in Fig. 10. The propagation time for each model took under 10 ms on an Intel Core i7-2600 CPU @ 3.40GHz PC with 8GB ram. Table 1 reports the total number of *MAVO* label refinements (column 4), the number of such changes which should have been propagated based on all the well-formedness constraints of `iStarCD` (column 7) and the number of changes which were actually propagated using the current rules (column 8). For example, in $P1'$, 16 labels were refined, 5 of which should have been propagated, while 4 were actually

propagated via our rules. We discuss the changes which were not propagated, as well as our general experiences applying *MAVO* to the examples, in the next section.

10.3 Discussion and lessons learned

We have successfully applied all aspects of the *MAVO* framework to the Meeting Scheduler and the Inflo system. Although this process provided evidence to support the practical applicability of *MAVO* to an RE context, it has also revealed some areas in which *MAVO* could be improved, allowing us to identify areas of potential future work.

Q1 The *MAVO* approach was not used in the original modeling exercise for the Inflo model, so the starting point of our process reported in Sect. 10 was a model which allowed just one set of possibilities. To construct the version with uncertainty, we "reverse-engineered" uncertainty into a model, based on the third author's recollection of the system and the modeling process, and thus needed to add several additional elements to express various concrete options before annotating the model. Had we used *MAVO* throughout the modeling process, these alternatives would have been added as they arose, making the annotation process more natural.

Although we were able to express all uncertainties encountered in our examples using *MAVO* annotations, the process could be made more intuitive. For example, if there was an uncertainty about an element type (e.g., is a link a `help` or a `make`, in i* terms?), each possible type had to be added to the model with an M annotation. Similarly, if there was an uncertainty involving the source or destination of a link, multiple M-annotated links had to be added to each possible source/destination. In such cases, as well as when adding V-annotated elements, the modeler often had knowledge that these options were mutually exclusive. This experience suggests that we can improve *MAVO* tool support to include explicit-type uncertainty and enforcement of mutual exclusion of alternatives (see Sect. 12.2).

We also note (see Table 1) that M annotations were used far more frequently than V or S. Although this may be inherent in the frequency of certain types of uncertainty, it may also be due to the utility of the individual types of labels. For example, we often wanted to restrict the exact set of atoms with which a V-annotated object can merge. This experience suggests further development of *MAVO* patterns for expressing uncertainty, as well as improvements of concrete syntax for doing so (see Sect. 12.2).

Q3 In capturing refinement rationale, we sometimes did not feel that it was necessary to attach a rationale statement to a particular refinement, and often chose to group several individual refinements (e.g., all related to moderation) to share a single rationale statement. This experience indicated that our tool support should be extended to allow for such possibilities.

Q6 As reported in Sect. 10, many of the refinements made to the i* model of the Inflo system were not propagated to the corresponding class diagram. For example, in the refinement `P1′`, the rules only propagated four out of the 16 label refinements. Many of these exclusions can be accounted for by the relationship between i* and class diagrams—there are many concepts in i* (goals, contributions, decompositions) which are not mapped to concepts in class diagrams; therefore, changes to these elements will not be propagated. Other cases are due to the fact that some of the well-formedness constraints of the traceability relation do not force propagation. For example, when a M-annotated task is deleted in a refinement, the operation that it maps to need not be correspondingly deleted because, by the multiplicity constraints (see Fig. 5), it is mapped to "0..1" tasks. These situations account for 11 out of the 16 label refinements. Of the final five refinements, four were propagated by the rules. The remaining case (that should have been propagated but was not) is due to the limitation of our change propagation technique which uses only multiplicities rather than all well-formedness constraints.

Scalability While it is useful to express all reasonable concretizations in one model, the addition of extra elements and links can increase its complexity. For *MAVO* applications, the complexity increase is linear in the "amount" of uncertainty in the model. Moreover, our approach can directly benefit from all future improvements supporting model scalability, for example, the use of modularization for i* models.

11 Related work

In this section, we survey a number of approaches related to our work on uncertainty.

Uncertainty in RE Several approaches consider uncertainty in requirements, often as part of an overall strategy for managing uncertainty in software development. For example, [14] provides a framework for uncertainty in SE, recognizing requirements uncertainties such as inconsistencies, clarity, and accuracy. While this approach uses existing techniques to model uncertainty in isolation, our approach aims to integrate uncertainty modeling with existing RE modeling approaches. The work in [4] investigates requirements changes and uncertainty in an experimental field study, using data gathered to provide a useful list of root causes for uncertainty, including vague product strategy and missing key stakeholders. Although it is useful

to consider possible root causes for uncertainties, our approach takes a more narrow focus, capturing the manifestation of uncertainties in the structure of RE models.

The approach in [19] argues that much of the uncertainty in SE comes from our inability to precisely measure software and its processes, proposing the use of rough sets to capture software properties and requirements. Although some of the uncertainties captured by *MAVO* might be related to imprecision, our approach aims to support a wider variety of model uncertainty. Similarly, work in [33] studies the problem of "imperfect information" in software development, using fuzzy set theory and probability theory to model imprecise non-functional requirements in order to evaluate design decisions. This approach allows modeling of uncertainty concerned with specific NFRs, while our approach allows for uncertainty over RE models which may capture NFRs. Thus, when it comes to the precision of requirements, our approach inherits the expressiveness of the modeling language to which it is applied.

In another direction, Herrmann [12] studied the value of being able to express *vagueness* within design models. His modeling language SeeMe has notational mechanisms similar to *OW* and *May* partiality; however, there is no formal foundation for these mechanisms.

Much of the investigation of uncertainty in RE is concerned with work on adaptive systems. Such systems aim to respond to uncertainty during run-time by specifying functional adaptations as part of RE (see [44] for overview). Our approach is aimed to represent uncertainty in the content or structure of requirements models arising as part of elicitation, ideally resolved as part of the requirements process, and do not explicitly to handle run-time uncertainty.

Uncertainties in software development are often considered as part of risk management. For example, the approach in [15] advocates the early and explicit consideration of risks as part of RE goal modeling. Although certain risk factors (e.g., an unknown budget) may motivate the presence of model uncertainty, our framework is not explicitly intended to capture these risks.

Argumentation and rationale in RE Previous work has focused on the use of rationale as part of design decisions (e.g., [36]), whereas others added rationale as part of the development of goal models [17, 27] or other requirements models [9]. In contrast, our approach focuses on adding rationale for the reduction in uncertainty, which may or may not involve a design decision.

RE Model mapping and traceability We consider mappings between requirements models as part of managing uncertainty across multiple models. Although we use a particular traceability mapping in our example (adapted from [3]), our approach is meant to apply to any mapping between models (other examples can be found in [19, 20, 26]). Traceability among software artifacts has received much attention (see [45] for an overview of software traceability, including traceability as part of RE), much of it geared toward traceability from models to requirements (e.g., [5, 10]), or from requirements to architecture (e.g., [9, 50]). Some approaches look at traceability between models. For example, [46] focuses on change management for goal models, detecting conflicts and analyzing goal achievement when a goal is added or deleted. Our work is broader in that it considers any change which reduced uncertainty, and can be applied to multiple model types. Other work has a more general focus on software models, often using UML models as examples (e.g., [1, 16, 25]). As our work manages uncertainty on top of existing traceability mappings, including integrity constraints, the traceability links and rules proposed in such work could be integrated with our approach.

Partial modeling `May` partiality in *MAVO* is related to various modal extensions to *behavioral* modeling formalisms. For example, Modal Transition Systems (MTSs) [21] allow introduction of uncertainty about transitions on a given event, whereas Disjunctive Modal Transition Systems (DMTSs) [22] add a constraint that at least one of the possible transitions must be taken in the refinement. Concretizations of these models are Labeled Transition Systems (LTSs). MTSs and DMTSs have been used to capture some forms of uncertainty in early design models [47]. The *MAVO* approach allows specification of more uncertainty types, although it is applicable only to *structural* models.

Change propagation Our work on uncertainty change propagation in Sect. 8 is closely related to research on the propagation of model content changes in order to fix inconsistencies in model-based software development. 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 are usually expressed in a constraint language, such as Beanbag [49] or EVL [18]. Several other approaches use logic, such as in Blanc et. al [2], the xlinkit framework [32], the triple graph grammars approach [29], or, more recently, Reder et. al. [37].

Our approach also follows the repair rule strategy; however, the focus is on the level of uncertainty rather than model content. That is, the propagation rules in Fig. 10 repair the uncertainty information (e.g., annotations) in one model in order to be consistent with another model related via a traceability relation.

In other work [41], we have explored doing *MAVO* uncertainty change propagation with the full FO encoding

of a model using an SMT solver. The change propagation rule approach in the current paper differs from the one in [41] because we are considering changes to both annotations (e.g., rule R1) and the model atoms (e.g., rule R4), while in [41], we considered only changes to annotations. Furthermore, the propagation rules avoid the potential exponential complexity of the SMT approach, albeit with the limitation that they only apply to multiplicity-based well-formedness constraints.

Representing sets of models Work on representing sets of models as a model is closely related to ours. For example, a metamodel is a model that defines the set of models. Such metamodels are typically expressed using a metametamodeling language such as MOF [34] and are intended for representing model types rather than uncertainty within a particular model. The element types in a metamodel correspond to the domain concepts that the modeling language "talks about," whereas in a *MAVO* model, the elements of the base model are particular domain elements.

Another relevant area is product line software development [35], where the variability of a model must be made explicit. This can be understood as representing a set of models (i.e., the set of model variants). Most approaches keep the expressions of variability in a separate model (e.g., a feature model), but some incorporate these directly into the model using notational extensions in the metamodel [31]. Variability modeling is well suited to expressing product variants but would be awkward and limiting if used for expressing uncertainty. The set of variants represented by a variability model are all "desired" possibilities, whereas all but one concretization of a partial model is undesirable. This makes the process of refinement central to partial modeling but not for variability models. In addition, the set of variants is typically finite since each variant must correspond to a well-defined combination of features, while the set of concretizations of a partial model is often infinite because uncertainty can be more open-ended than product variation.

12 Conclusion and future work

12.1 Summary

Our overall goal is to produce a comprehensive strategy for uncertainty management in software engineering. In our previous work [6, 40, 43], we have developed a formal approach called *MAVO* for expressing and reasoning with model uncertainty. We have explored model transformations for such models [7] and done a comparative analysis of tool support for property checking [38].

In this paper, we expanded on previous work by applying *MAVO* to the RE modeling context to answer six methodological and algorithmic questions about uncertainty in RE. Specifically, we made the following contributions: we explicated methodological guidelines for using *MAVO* annotations (**Q1**) and illustrated them through an application to expressing uncertainty in RE models; we applied *MAVO* refinement to expressing uncertainty reduction in RE models (**Q2**); we proposed a methodological approach to documenting the rationale of these refinements (**Q3**); we showed how to apply the *MAVO* uncertainty to traceability relations between RE models (**Q4**); we developed an efficient algorithm for checking the validity of *MAVO* refinements (**Q5**); we developed an approach for using multiplicity constraints to automatically propagate uncertainty-reducing changes (**Q6**); we described an implementation of uncertainty management in the MMTF tool; and we applied the *MAVO* framework to two examples, reporting on experiences and limitations. Through these contributions, we have made progress toward the identification and resolution of uncertainty early in the software development process. Also, although we illustrated our approach in an RE scenario using i* and UML models, it is general enough to be applied as part of any RE modeling approach that has a metamodel.

12.2 Future work

We see this paper as a step toward the development of a comprehensive strategy for uncertainty management across the development life cycle. We intend to conduct additional case studies in order to better evaluate the effectiveness of our approach as well as improve our tool support. We describe other opportunities for follow-on work below.

Undoing refinements The recording of rationale with a *MAVO* refinement provides the opportunity to revisit decisions and explore other alternatives. To do so, we need to be able to change the model to reflect an alternative earlier decision without affecting the decisions that came later. At a high level, this entails increasing uncertainty in the model just enough to "remove" the original decision and then refine it to reflect the new decision. While the formalization of *MAVO* may help do this in a sound way, dependencies between refinements and linking multiple refinements to the same rationale make this problem even more complex.

Trace analysis Having "lifted" traceability relations to the uncertainty setting, we are interested in the corresponding lifting of the different types of analyses that are typically done using traceability relations [46]. This approach has rich potential, for example, in being able to define change impact analysis across related models containing uncertainty. We are currently exploring approaches

for doing this lift by encoding existing traceability analysis techniques into FOL and using these with the *MAVO* FO encoding of partial models.

Uncertainty notation In order to provide a modeling language-independent implementation of *MAVO* within MMTF, we proposed a method for augmenting an arbitrary metamodel to support *MAVO* annotations. This adequately addresses the abstract syntax of a modeling language but does not specify how the annotation support will be integrated with a particular model editor or how this affects the concrete syntax (i.e., notation). For example, in a class diagram, if we are not sure whether an operation belongs to class `A` or class `B`, we could express this in the abstract syntax using *MAVO* by having M-annotated `ownedOperation` relations from each class to the operation. However, in the concrete syntax, the `ownedOperation` relationship is only expressed *implicitly*, by putting the operation inside the class box. Thus, it is not clear where to put the M annotations. Furthermore, it is also unclear how to indicate that the operation may be in class `A` or in class `B`—then it would have two class boxes at the same time. One approach we are exploring for addressing these kinds of issues is to identify problematic concrete syntax patterns and study how uncertainty can be expressed when they occur. We are also investigating extensions to *MAVO* to more conveniently express certain commonly occurring kinds of uncertainty, for example, when a model element type is uncertain or when two M annotations must be mutually exclusive. Preliminary work in this context is reported in [8].

Extending change propagation Currently, the rules we described in Sect. 8 are only used to propagate change *across* a traceability relation; however, once such a change is propagated, it could trigger further propagated changes entirely *within* the model. We are investigating ways to generalize the rules to support such additional propagations. Finally, as the current rules apply only to traceability relations with multiplicity constraints, we are also exploring ways to handle richer well-formedness constraints, for example, by automatically generating the propagation rules directly from the well-formedness constraints.

References

1. Assawamekin N (2010) An ontology-based approach for multiperspective requirements traceability between analysis models. In: Proceedings of ACIS'10, pp 673–678
2. Blanc X, Mougenot A, Mounier I, Mens T (2009) Incremental detection of model inconsistencies based on model operations. In: Advanced information systems engineering, vol 5565. Springer, Berlin, pp. 32–46
3. Cysneiros G, Zisman A, Spanoudakis G (2003) A traceability approach from i* and UML models. In: Proceedings of SELMAS'03, LNCS
4. Ebert C, De Man J (2005) Requirements uncertainty: Influencing factors and concrete improvements. In: Proceedings of ICSE'05, pp 553–560
5. Egyed A (2003) A scenario-driven approach to trace dependency analysis. IEEE TSE 29(2):116–132
6. Famelis M, Chechik M, Salay R (2012) Partial models: Towards modeling and reasoning with uncertainty. In: Proceedings of ICSE'12
7. Famelis M, Salay R, Chechik M (2012) The semantics of partial model transformations. In: Proceedings of MiSE'12
8. Famelis M, Santosa S (2013) MAV-Vis: a notation for model uncertainty. In: Proceedings of MiSE'13. To appear
9. Gilson F, Englebert V (2011) Rationale, Decisions and alternatives—traceability for architecture design. In: Proceedings of ECSA'11, companion vol, pp 4:1–4:9
10. Goknil A, Kurtev I, van den Berg K (2010) Tool support for generation and validation of traces between requirements and architecture. In: Proceedings of ECMFA-TW'10, pp 39–46
11. Gordijn J, Petit M, Wieringa R (2006) Understanding business strategies of networked value constellations using goal- and value modeling. In: Proceedings of ICRE'06, pp 126–135
12. Herrmann T (2009) Systems design with the socio-technical walkthrough. In: Handbook of research on socio-technical design and social networking systems, pp 336–351
13. Horkoff J, Yu E (2010) Interactive goal model analysis applied—systematic procedures versus ad hoc analysis. In: The practice of enterprise modeling, 3rd IFIP WG8.1 (PoEM'10)
14. Ibrahim H, Far BH, Eberlein A, Daradkeh Y (2009) Uncertainty management in software engineering: Past, present, and future. In: Proceedings of CCECE'09, pp 7–12
15. Islam S, Houmb SH (2010) Integrating risk management activities into requirements engineering. In: Proceedings of RCIS'10, pp 299–310
16. Jirapanthong W (2009) Analysis of relationships among software models through traceability activity. In: Advances in information technology, communications in computer and information science vol 55, pp 71–80
17. Kaiya H, Horai H, Saeki M (2002) AGORA: attributed goal-oriented requirements analysis method. In: Proceedings of RE'02, pp 13–22
18. Kolovos D, Paige R, Polack F (2008) Detecting and repairing inconsistencies across heterogeneous models. In: Proceedings of ICSTVV'08, pp 356–364
19. Laplante PA, Neill CJ (2005) Modeling uncertainty in software engineering using rough sets. J Innov Syst Soft Eng 1:71–78
20. Lapouchnian A, Liaskos S, Mylopoulos J, Yu Y (2005) Towards requirements-driven autonomic systems design. ACM SIGSOFT SEN 30(4):1
21. Larsen KG, Thomsen B (1988) A Modal process logic. In: Proceedings of LICS'88, pp 203–210
22. Larsen P (1991) The expressive power of implicit specifications. In: Proceedings of ICALP'91, *LNCS*, vol 510, pp 204–216
23. Lin L, Zhi J (2006) Integrating goals and problem frames in requirements analysis. In: Proceedings of RE'06, pp 349–350
24. Lung J, Easterbrook SM (2012) Inflo: collaborative reasoning via open calculation graphs. In: Proceedings of CSCW'12, pp 1199–1202
25. Mader P, Gotel O, Philippow I (2008) Enabling automated traceability maintenance by recognizing development activities applied to models. In: Proceedings of ASE'08, pp 49–58
26. Maiden N, Jones S, Manning S, Greenwood J, Renou L (2004) Model-driven requirements engineering: synchronising models in an air traffic management case study. In: Proceedings of CAiSE'04, *LNCS*, vol 3084, pp 3–21
27. Maiden N, Lockerbie J, Randall D, Jones S, Bush D (2007) Using satisfaction arguments to enhance i* modelling of an air traffic management system. In: Proceedings of RE'07, pp 49–52

28. Mens T (2008) Introduction and roadmap: history and challenges of software evolution. In: Software evolution, pp 1–11
29. Mens T, Straeten RVD (2007) Incremental resolution of model inconsistencies. In: Proc. of WADT'06, pp 111–126. Springer, Berlin
30. Moisiadis F (2000) Prioritising scenario evolution. In: Proceedimgs of RE'00, pp 85–94
31. Morin B, Perrouin G, Lahire P, Barais O, Vanwormhoudt G, Jézéquel J (2009) Weaving variability into domain metamodels. In: Model driven engineering languages and systems pp 690–705
32. Nentwich C, Emmerich W, Finkelstein A (2003) Consistency management with repair actions. In: Proceedimgs of ICSE'03, ACM Press, New York, pp 455–464.
33. Noppen J, van den Broek P, Aksit M (2008) Software development with imperfect information. J Soft Comput 12(1):3–28
34. OMG: Meta Object Facility (MOF) Core Specification Version 2.0 (2006). http://www.omg.org/cgi-bin/doc?formal/2006-01-01
35. Pohl K, Böckle G, Van Der Linden F (2005) Software product line engineering: foundations, principles, and techniques. Springer, New York
36. Potts C, Bruns G (1988) Recording the reasons for design decisions. In: Proceedings of ICSE'88, pp 418–427
37. Reder A, Egyed A (2012) Computing Repair trees for resolving inconsistencies in design models. In: Proceedings of ASE'12, pp 220–229
38. Saadatpanah P, Famelis M, Gorzny J, Robinson N, Chechik M, Salay R (2012) Comparing the effectiveness of reasoning formalisms for partial models. In: Proceedings of MoDeVVa'12
39. Salay R, Chechik M, Easterbrook S, Diskin Z, McCormick P, Nejati S, Sabetzadeh M, Viriyakattiyaporn P (2007) An eclipse-based tool framework for software model management. In: Proceedings of OOPSLA eclipse workshop, pp 55–59
40. Salay R, Chechik M, Gorzny J (2012) Towards a methodology for verifying partial model refinements. In: Proceedings of VOLT'12
41. Salay R, Chechik M, Gorzny J (2013) Change propagation due to uncertainty change. In: Proceedings of FASE'13. LNCS, vol 7793, pp 21–36
42. Salay R, Chechik M, Horkoff J (2012) Managing requirements uncertainty with partial models. In: Proceedings of RE'12
43. Salay R, Famelis M, Chechik M (2012) Language independent refinement using partial modeling. In: Proceedings of FASE'12, *LNCS*, vol 7212, pp 224–239
44. Sawyer P, Bencomo N, Whittle J, Letier E, Finkelstein A (2010) Requirements-aware systems: a research agenda for RE for self-adaptive systems. In: Proceedings of RE'10, pp 95–103
45. Spanoudakis G, Zisman A (2005) Software traceability: a roadmap. J Softw Eng Knowl III:1–35
46. Tanabe D, Uno K, Akemine K, Yoshikawa T, Kaiya H, Saeki M (2008) Supporting requirements change management in goal oriented analysis. In: Proceedings of RE'08, pp 3–12
47. Uchitel S, Chechik M (2004) Merging Partial behavioural models. In: Proceedings of SIGSOFT FSE'04, pp 43–52
48. van Lamsweerde A (2009) Requirements engineering: from system goals to UML models to software specifications. Wiley, New York
49. Xiong Y, Hu Z, Zhao H, Song H, Takeichi M, Mei H (2009) Supporting automatic model inconsistency fixing. In: Proceedings of ESEC/FSE'09, pp 315–324
50. Yrjönen A, Merilinna J (2010) Tooling for the full traceability of non-functional requirements within model-driven development. In: Proceedings of ECMFA-TW'10, pp 15–22
51. Yu E (1997) Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of RE'97, pp 226–235