Managing Requirements Uncertainty with Partial Models

Rick Salay, Marsha Chechik, and Jennifer Horkoff
University of Toronto
Toronto, Canada
{rsalay,chechik,jenhork}@cs.toronto.edu

***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 applied it to capturing uncertainty, including reasoning over design models containing uncertainty. In this paper, we show how to apply our approach to managing requirements uncertainty. In particular, we address the problem of specifying uncertainty within a requirements model, refining a model as uncertainty reduces and reasoning with traceability relations between models containing uncertainty. We illustrate our approach using the meeting scheduler example.**

I. Introduction

In the requirements phase of the software lifecycle, unresolved decisions about needs, incomplete understanding of the problem domain and disagreements among stakeholders can all produce uncertainty. If uncertainties in the requirements phase remain unresolved, corresponding uncertainties are forced into the design phase alongside design-specific uncertainties such as undetermined architectural strategies or the presence of alternative design possibilities. In a similar way, design uncertainty impacts implementation uncertainty. Since the uncertainty at each phase is linked to the next, any increase or reduction at one phase must be appropriately propagated to the next. Furthermore, since the requirements phase is the first one in this sequence, as with defects, requirements uncertainty has the highest potential impact on the entire software life cycle. To facilitate the resolution of uncertainty early in the software development process, it is important to provide techniques which explicitly capture uncertainty as part of Requirements Engineering (RE).

Models are advocated as part of the 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 uncover uncertainty over the contents and structure of the model. Such uncertainties involve gaps in domain knowledge, disagreements between stakeholders, or uncertainty over model details. The modeling process often implicitly involves identifying model uncertainties and then resolving them through further elicitation or decision making. It is useful to explicitly express uncertainty in RE models, and to capture such uncertainty-reducing decisions and elicited information as part of the modeling process.

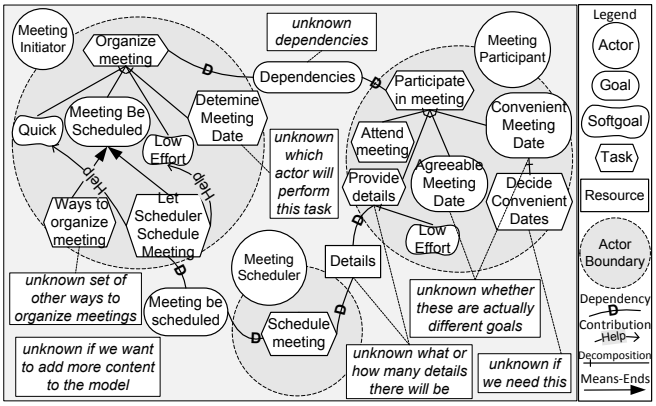

Figure 1. An early RE diagram with annotated uncertainty for the meeting scheduler example.

A. Motivating Scenario

A requirements process may make use of multiple types of models, for example, to separate "early", high-level modeling from "later" RE stages ([32]), as a form of expressive redundancy, to ensure important aspects of the domain are captured (e.g., [20] [8]), or to move towards system design (e.g., [30], [15]). We illustrate uncertainty representation and mapping on a scenario moving from early to late RE models (see Figure 2).

Expressing uncertainty in RE models. Figure 1 shows an i* model, adapted from [32][1], created in the early stages of determining the requirements for an automated meeting scheduler. We summarize our motivating scenario in Figure 2, calling this early RE model P1. 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

[1]Parts of this scenario are adapted from [2] and have appeared in [18], [22].

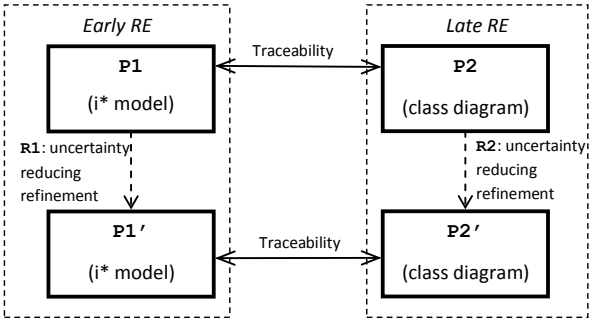

Figure 2. Overview of a sample RE modeling process considering uncertainty.

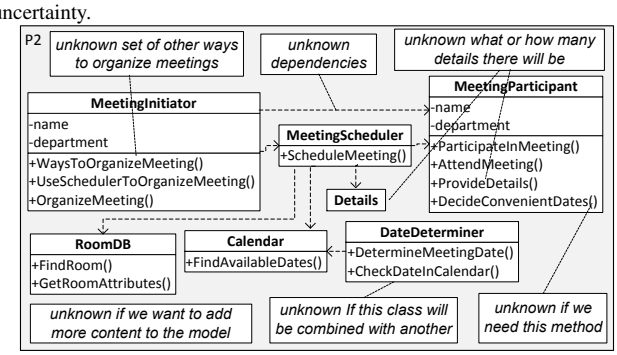

Figure 3. A late RE diagram with uncertainty for the meeting scheduler example.

Participant needs to provide details to the scheduler, but will list these details later"). It may be especially useful to record uncertainty if the process of creating the model is long and may be interrupted, i.e., via successive modeling workshops [20].

In Figure 1, we captured uncertainties using text annotations — an approach which is neither formal nor systematic. 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?*

After gaining an understanding of the domain and high-level requirements through early RE modeling, a model such as the class diagram in Figure 3 may be developed, in order to represent relevant details of domain entities. In Figure 2, we call the resulting model `P2`. Although, ideally, uncertainties would be resolved in early RE stages, uncertainties may remain when creating later RE models (shown in Figure 3 via text annotations). By answering **Q1**, we can express uncertainty over multiple types of models used in an RE process.

Uncertainty reducing changes. 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. For example, elicitation may reveal that meetings are often scheduled via in person communication, and we may wish to add this to our model in Figure 1. *Model refinement*[2] is captured in the left side of Figure 2, where, after the resolution of some uncertainty, the model `P1` is refined into `P1′` via a mapping `R1`. Thus, we ask: **Q2:** *How to record uncertainty-reducing model refinements in RE models?*

Capturing refinement rationale. Refinements are often caused by the availability of new information. Thus, it would be useful to optionally annotate such decisions with textual rationale, e.g., "Spoke to Jack, March 1. He said they typically organize meetings in person. This is quick, as colleagues are collocated, but it is often quite difficult to find an agreeable date". Although methods to capture rationale for RE models have been introduced (e.g., [13], [21]), they are not directly linked to the process of resolving model uncertainty. Thus, we ask: **Q3:** *How to capture rationale specific to uncertainty reduction?*

Model change and model uncertainty. When modifying a model, we may wish to know whether our changes actually make the model less uncertain. For example, in Figure 1, after further elicitation we may decide that the `MeetingParticipant` needs to provide available dates along with a meeting location. Does this change make the model less uncertain? Thus, we ask: **Q4:** *How to check whether model changes reduce model uncertainty?*

Traceability relations and uncertainty. Requirements traceability is a central concern for requirements engineering and has been well studied (e.g., see [28] 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, refinement, etc. Various approaches to RE modeling introduce traceability relations between multiple types of models used in RE (e.g., [2], [20]). For example, Figure 4 shows traceability between i* and class diagram meta-models. Thus, we ask: **Q5:** *What is the meaning of a traceability relation between models containing uncertainty?*

B. Contributions and Organization

Our goal is *to enable expressing model uncertainties and uncertainty resolution for and across existing RE models, in a systematic, language-independent and formal way*.

In previous work, the first two authors have developed a language-independent approach for expressing certain types of uncertainty [26] using *partial models*. The approach is

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

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 a 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 [5] and verifying that model changes reduce uncertainty [25]. Despite the fact that our approach is formally grounded, the formal details can be hidden from the user, making it appropriate for use with stakeholders in early RE. We give background on partial models and uncertainty-reducing refinement in Section II.

While our previous work defined individual components of our approach, in this paper, we present these components as part of a methodology for managing uncertainty in requirements engineering by addressing questions **Q1**-**Q5**. Specifically, in Section III, we show how to use partial models to express uncertainty in RE models, illustrating them on i* models and class diagrams (**Q1**). In Section IV, we look at applying partial model refinement to RE models. Specifically, we show how to construct the refinement mapping (**Q2**) and capture refinement rationale (**Q3**). In Section V, we discuss how to check whether a change made to a model constitutes a refinement (**Q4**). In Section VI, we show how to lift a traceability relation to the uncertain case. The machinery for answering **Q1** and **Q2** is taken directly from our earlier work. Novel contributions include: the notion of documenting refinement rationale (**Q3**), a new computational procedure for checking uncertainty reducing refinement (**Q4**) and an expansion our partial modeling approach to address traceability relations with uncertainty (**Q5**). We conclude the paper with a comparison between our approach and related work in Section VII and a summary and discussion in Section VIII.

II. Background

In this section, we briefly review the formal concepts of language-independent partial modeling introduced in [26]. We use it to capture models with uncertainty. When a model contains partiality information, we call it a *partial* model.

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 Figure 4 shows a (simplified) metamodel for i* models, and the lower part gives one for class diagrams. 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

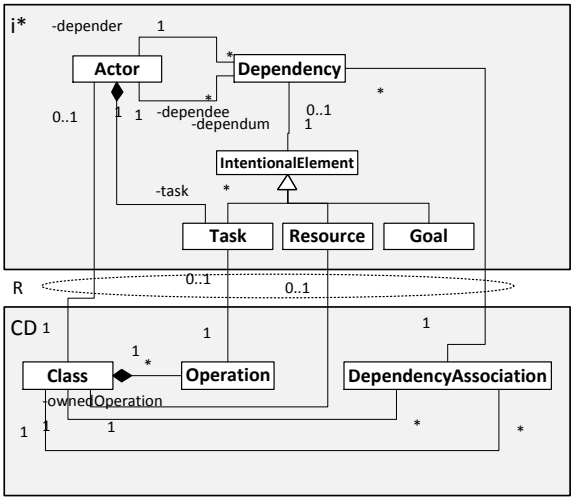

Figure 4. The traceability relation `Tr` between an i* model and class diagram defined over fragments of the metamodels.

T be the metamodel of $bs(P)$. Then, $[P]$ denotes the set of T models called the *concretizations* of P. P is called *consistent* iff it allows at least one concretization, i.e., $[P] \neq \emptyset$.

Models consist of a set of *atoms*, i.e., the elements and relation instances of the types defined in its metamodel.

Partiality is used to express uncertainty about the model until it can be resolved using *partiality refinement*. Refining a partial model means removing partiality 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.

***MAVO* annotations.** We achieve language-independence by adding partiality information as annotations of a base model. For example, models `P1` and `P1′` in Figure 6 show annotations on an i* base model. We use four types of partiality annotations, each adding support for a different type of uncertainty in a model, as described below.

The *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 a 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*.

MAVO refinement. When the annotations are used together, the refinement mapping combines the mappings of all four types into a single relation between the atoms of the two models.

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.

Definition 2 (MAVO Refinement): 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*.

Formalizing *MAVO*. In this section we describe how *MAVO* partiality formally characterizes model uncertainty and is given to help the reader understand the computational issues of encoding and reasoning with partial models. Readers interested in 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 3 (Metamodel): A *metamodel* is an FOL theory $T = \langle \Sigma, \Phi \rangle$, where Σ is the *signature* with sorts and predicates representing the element types, and Φ is a set of sentences representing the well-formedness constraints. The models that conform to T are the finite FO Σ-structures that satisfy Φ according to the usual FO satisfaction relation. We denote the set of models with metamodel T by $Mod(T)$.

For example, for the fragment of the i* metamodel in Figure 4, $\Sigma_{\texttt{iStar}}$, 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 constraints.

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

Figure 5. The FO encoding of $P_{\texttt{M1}}$.

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. $Mod(FO(P)) = [P]$. 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, i.e., $bs(P_M) = M$ 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 the fact that a must occur in every concretization is removed.

We illustrate the above 3-step construction using the partial i* model P1 in Figure 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 3), where $\Sigma_{\texttt{M1}}$ and $\Phi_{\texttt{M1}}$ are model M1-specific predicates and constraints, defined in Figure 5. They extend the signature and constraints for i* models described in Figure 4. 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* models that satisfy the constraint set $\Phi_{\texttt{M1}}$ in Figure 5. For conciseness, we abbreviate element names in Figure 5, e.g., 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 $Distinct_{E-*}$ 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. First, it allows us to do *property checking*, i.e., 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. Second, 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. Third, we can verify that a given mapping R is a refinement mapping between a pair *MAVO* models by checking the conditions in Definition 2. Checking condition Ref1 is the consistency check. Checking condition Ref2 can be cast as a special case of property checking by checking if $FO(P') \Rightarrow \mathsf{R}(\Phi_P)$, where R translates Φ_P according to the refinement mapping. The theory and the tooling for refinement are described in [25]. Finally, the FO encoding allows us to *extend the expressiveness of MAVO* by allowing the specification of more complex textual constraints using FOL to augment the annotations.

III. Expressing Uncertainty in RE Models

In this section, we address question **Q1** from Section I and show how to use *MAVO* partiality to express uncertainty in RE models.

Model P1 in Figure 8 shows the application of *MAVO* partiality to our i* model, describing the same points of uncertainty as given in the ad-hoc way in Figure 1. Consider also the class diagram P2 in Figure 8, which is a MAVO-annotated model capturing the same information as done in an ad-hoc way in Figure 3. Both P1 and P2 use the same partiality annotations, demonstrating *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 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 arises, for example, when multiple stakeholders provide conflicting information. Thus, the M annotation provides a language-independent way to *tolerate conflicts* until they can be 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 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 or several. In general, Abs partiality provides a way to create placeholders in the modeler to indicate that "further elaboration is yet to come".

Early in a modeling process, we may not be sure whether two atoms are distinct or should be the same, i.e., we may be uncertain about atom *identity*. This annotation is used 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 Figure 1, it is known that 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 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 a temporary "variable" actor, i.e., something treated like an actor but, in a refinement, potentially equated (merged) with other variable actors and eventually assigned to a constant actor.

Finally, it is common, during model development, to make the assumption that the model is still incomplete, i.e., 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 defines model completeness in a very specific way: a model is complete iff it should not be extended. However, a model marked as complete with COMP can still be changed if it has annotations representing other types of uncertainty. For example, even if P1 is marked as COMP , it would still be possible to replace the resource Details to more specific sets of detail resources since this is an *Abs* refinement. Yet, it would not be possible to add a new goal to Meeting Participant because this would

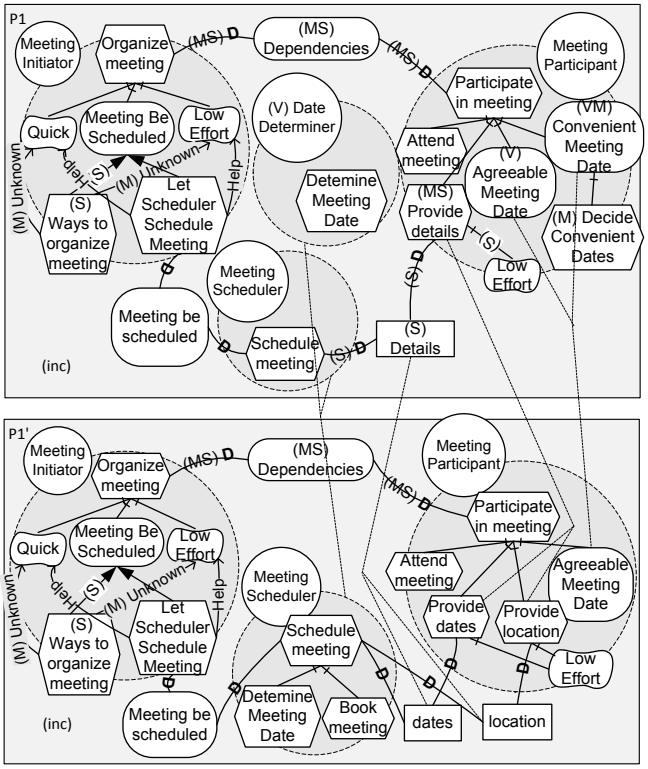

Figure 6. The use of *MAVO* partiality annotations to express the uncertainty in the model in Figure 1 and an example of a refinement that reduces uncertainty.

be a model extension. *OW* partiality is used in P1 and P2 with the INC annotation to indicate that the model is still incomplete and can be extended.

IV. Capturing Uncertainty Reducing Refinement

In this section, we address the questions **Q2** and **Q3** from Section I. Specifically, we show how uncertainty reduction in an RE model is done by constructing a *partial model refinement* of the model including optional refinement rationale.

Constructing the Refinement Mapping (Q2). 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 6 gives an example of a partial i* model P1′ refining model P1. As the decisions about reduction of 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 actor Meeting Scheduler in P1′. Since P1 is marked INC, more information can be added to the model so Determine Meeting Date can be made 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 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 goal Agreeable Meeting Date in P2. Connected with this, she also realized that the M-annotated task Decide Convenient Dates is not needed after all, and so it is removed in P2.

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, e.g., in 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 Section VIII.

V. Checking Uncertainty Reducing Refinement

In this section, we address the question **Q4** from Section I. Specifically, we show how to verify that a potential refinement actually reduces uncertainty.

Verifying a *MAVO* refinement requires showing that the two refinement conditions in Definition 2 hold. In general, this can be done using the FO encoding of the two models, and we have implemented prototype tool support [25] 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

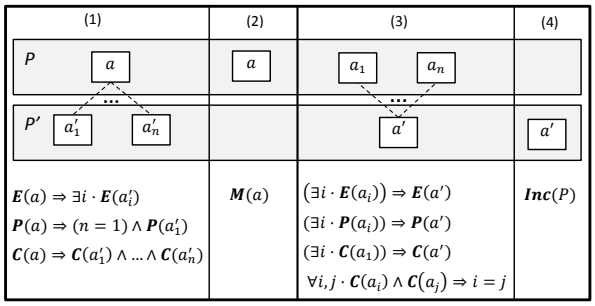

Figure 7. Summary of the constraints on annotations of model elements across a *MAVO* refinement mapping.

additional textual constraints augmenting *MAVO* annotations, it has computational limitations and typically is not applicable to large (even in the RE sense!) models. However, when we limit ourselves to just using *MAVO* annotations, we can simplify checking refinement condition Ref2 by defining syntactic constraints (i.e., necessary conditions) on the annotations that follow from the FO condition. We now describe this approach and discuss its scalability relative to the Alloy implementation.

Figure 7 summarizes these constraints. Each of the four columns indicates a different case (case number at 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 make reference to the full set of *MAVO* annotations described in Section II (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) is when an atom a of model P is refined to a set of atoms $a_1, ..., a_n$ 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 it can't merge with any other atom then neither can any of the a_is it refines to and so they too must 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 so is a'. The last sentence in case (3) says that at most one of the a_is could be a variable. Finally, if a new atom, not mapped to anything in P appears in P' (case (4)), then P could not be complete.

We illustrate the application of these rules for checking the refinement in Figure 6. The refinements of resource `Details` and task `Provide details` are examples of case (1). The merges of actor `Meeting Scheduler` with actor `Date Determiner` and goal `Agreeable Meeting Date` with

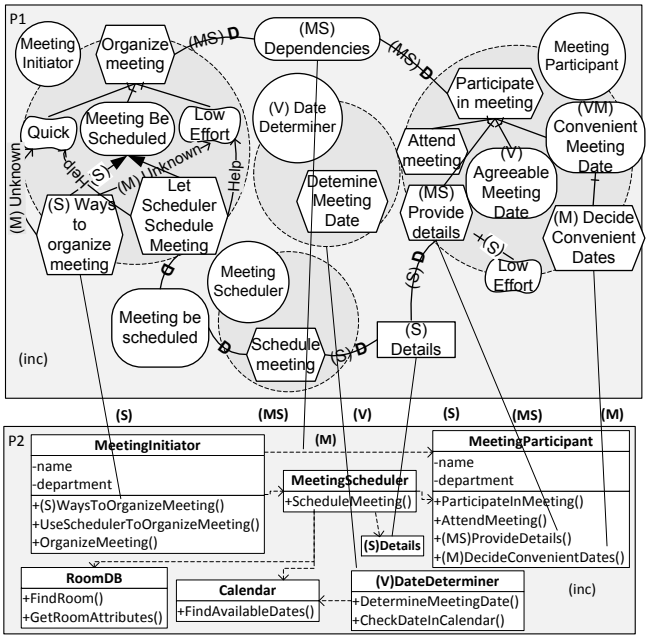

Figure 8. The i* model `P1` from Figure 6 mapped to the partial class diagram `P2` using the traceability relation `Tr(P1, P2)`.

`Convenient Meeting Date` are examples of case (3). The addition of task `Book meeting` is an example of case (4). Finally, the removal of `Decide Convenient Dates` is an example of case (2). Furthermore, the annotations in models `P` and `P′` in Figure 6 satisfy the conditions for their respective cases. For example, resource `Details` (with annotations ESC) refines to resources `Dates` (with annotations EPC) and `Location` (with annotations EPC). It is straightforward to see that these annotations satisfy the three constraints for case (1).

The procedure for applying Figure 7 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 time complexity $O(|P| \times |P'|)$, which allows us to conclude that this is a scalable strategy for checking condition Ref2 of refinement.

Checking the first refinement condition, Ref1, requires showing that P' has at least one concretization, i.e., that $FO(P')$ is satisfiable. This can be done using a SAT-solver. However, the process can be simplified for RE models by taking advantage of the following property discussed in [25]: when the base model of P' (i.e., P' with all the annotations removed) is a well-formed i* (resp., class diagram) model, then it is also a concretization of P', and thus refinement condition Ref1 is satisfied. Of course, this 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 to check Ref1 is required.

The base model of `P1′` in Figure 6 is a well-formed i* model so we can conclude that `P1′` satisfies refinement condition Ref1. Furthermore, following the above illustra-

tion, the refinement in Figure 6 satisfies the constraints in Figure 7 and so it satisfies refinement condition Ref2. Thus, we conclude that the mapping in Figure 6 shows a valid *MAVO* refinement and reduces uncertainty.

VI. Traceability Relations and Uncertainty

In this section, we consider the problem of traceability in the presence of model uncertainty and address question **Q5** from Section I. Specifically, we show how to "lift" an existing traceability relation to a partial traceability relation in order to incorporate uncertainty and relate models containing it.

To illustrate the approach, we begin with a traceability relation `Tr` between i* models and class diagrams adapted from [2]. Figure 4 shows the associations between elements of the relevant fragments of i* and the class diagram metamodels. In this case, `Tr` is used to show the overlap between the i* model and the class diagram. Specifically, 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 classes and operations, 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 metamodel elements, like `Tr`, to define traceability between models containing uncertainty. We can apply *MAVO* annotations to `Tr` just like we would to atoms of a model! Figure 8 shows the partial i* model `P1` from Figure 6 mapped to the partial class diagram `P2` using the resulting partial traceability relation $\texttt{Tr}(\texttt{P1}, \texttt{P2})$. Note that each trace link and each of its endpoints can be annotated with *MAVO* annotations. To reduce visual clutter, Figure 8 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 a M-annotated operation by the same name.

Furthermore, since the existence of the task and operation is uncertain, the existence of the traceability link should be uncertain as well. This is because the uncertainty of the endpoints is affected by the uncertainty of a link. Making the existence of the link certain, e.g., by annotating it with E, means that it occurs in every concretization and this would force the task and operation to also be in every concretization since a link can't occur without its endpoints. Thus, if the task and operation is present in every concretization, their existence is made certain despite the fact that they are 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.

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

The traceability relation in Figure 8 satisfies this rule.

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 Figure 8 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 will actually 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) as well as to the intentions of the analyst.

VII. 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, [10] 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. [3] 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 [14] 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 [23] 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 [9] 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 [27] 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 [11] 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., [24]), whereas others added rationale as part of the development of goal models [13], [21] or other requirements models [6]. In contrast, our approach focuses on adding rationale for the reduction of 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 [2]), our approach is meant to apply to any mapping between models (other examples can be found in [20], [15], [30]). Traceability amongst software artifacts has received much attention (see [28] for an overview of software traceability, including traceability as part of RE), much of it geared towards traceability from models to requirements (e.g., [4], [7]), or from requirements to architecture (e.g., [31], [6]). Other work has a more general focus on software models, often using UML models as examples (e.g., [1], [12], [19]). 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 *behavioural* modeling formalisms. For example, Modal Transition Systems (MTSs) [16] allow introduction of uncertainty about transitions on a given event, whereas Disjunctive Modal Transition Systems (DMTSs) [17] add a constraint that at least one of the possible transitions must be taken in the refinement. Concretizations of these models are Labelled Transition Systems (LTSs). MTSs and DMTSs have been used to capture some forms of uncertainty in early design models [29]. The *MAVO* approach generalizes `May` partiality to arbitrary model types and allows specification of more uncertainty types.

VIII. Conclusion and Future Work

In our previous work [26], [25], [5], we have developed a formal approach called *MAVO* for expressing and reasoning with model uncertainty and have applied it to design models. In this paper, we expanded *MAVO* and applied it to the RE modeling context, answering five methodological and/or algorithmic questions about uncertainty in RE. Specifically, we made the following contributions: (1) we explicated methodological guidelines for using *MAVO* annotations (**Q1**) and illustrated them through an application to expressing uncertainty in RE models; (2) we applied *MAVO* refinement to expressing uncertainty reduction in RE models (**Q2**); (3) we proposed a methodological approach to documenting the rationale of these refinements (**Q3**); (4) we developed an efficient algorithm for checking the validity of *MAVO* refinements (**Q4**) ; and (5) we showed how to apply the *MAVO* uncertainty to traceability relations between RE models (**Q5**). Through these contributions, we have made progress towards 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 where the model can be represented via a metamodel.

Future Work. We see the current paper as a step towards the development of a comprehensive strategy for uncertainty management across the development lifecycle, and many of the topics addressed in this paper have natural follow-up work. For example, the recording of rationale with a *MAVO* refinement provides the opportunity to revisit decisions and explore other alternatives, but how can the model then be changed to reflect an alternative earlier decision without affecting the decisions that came after? At a high level, we want to increase uncertainty in the model just enough to "remove" the original decision and then refine it to reflect the new decision. The formalization of *MAVO* may help do this in a sound way.

Now that we have "lifted" traceability relations to the uncertainty setting, another area that has rich potential is the corresponding lifting of the different types of analyses

that are typically done using traceability relations [28]. For example, how is change impact analysis across related models containing uncertainty conducted? How are model changes propagated across a traceability relation containing uncertainty? Furthermore, how are actual uncertainty reductions or increases propagated to related models? Since our approach expresses uncertainty as sets of concrete models, this suggests we need to "lift" the existing analyses that apply to individual models to work on sets of models. We are currently exploring approaches for doing this lift by encoding existing analysis techniques into FOL and using these with the *MAVO* FO encoding of partial models.

Finally, we intend to further evaluate our technique by applying it to industrial RE case studies.

REFERENCES

[1] N. Assawamekin. An Ontology-Based Approach for Multiperspective Requirements Traceability between Analysis Models. In *Proc. of ACIS'10*, pages 673–678, 2010.

[2] G. Cysneiros, A. Zisman, and G. Spanoudakis. A Traceability Approach from i* and UML Models. In *Proc. of SELMAS'03*, LNCS, 2003.

[3] C. Ebert and J. De Man. Requirements Uncertainty: Influencing Factors and Concrete Improvements. In *Proc. of ICSE'05*, pages 553–560, 2005.

[4] A. Egyed. A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE TSE*, 29(2):116–132, 2003.

[5] M. Famelis, M. Chechik, and R. Salay. Partial Models: Towards Modeling and Reasoning with Uncertainty. In *Proc. of ICSE'12*, 2012.

[6] F. Gilson and V. Englebert. Rationale, Decisions and Alternatives – Traceability for Architecture Design. In *Proc. of ECSA'11, Companion Vol.*, pages 4:1–4:9, 2011.

[7] A. Goknil, I. Kurtev, and K. van den Berg. Tool Support for Generation and Validation of Traces between Requirements and Architecture. In *Proc. of ECMFA-TW'10*, pages 39–46, 2010.

[8] J. Gordijn, M. Petit, and R. Wieringa. Understanding Business Strategies of Networked Value Constellations Using Goal- and Value Modeling. In *Proc. of ICRE'06*, pages 126–135, 2006.

[9] T. Herrmann. *Systems Design with the Socio-Technical Walkthrough*, pages 336–351. 2009.

[10] H. Ibrahim, B. H. Far, A. Eberlein, and Y. Daradkeh. Uncertainty Management in Software Engineering: Past, Present, and Future. In *Proc. of CCECE'09*, pages 7–12, 2009.

[11] S. Islam and S. H. Houmb. Integrating Risk Management Activities into Requirements Engineering. In *Proc. of RCIS'10*, pages 299–310, 2010.

[12] W. Jirapanthong. *Analysis of Relationships among Software Models through Traceability Activity*, volume 55 of *Communications in Computer and Information Science*, pages 71–80. 2009.

[13] H. Kaiya, H. Horai, and M. Saeki. AGORA: Attributed Goal-Oriented Requirements Analysis Method. In *Proc. of RE'02*, pages 13–22, 2002.

[14] P. A. Laplante and C. J. Neill. Modeling Uncertainty in Software Engineering using Rough Sets. *J. Innovations in Systems and Soft. Eng.*, 1:71–78, 2005.

[15] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu. Towards Requirements-Driven Autonomic Systems Design. *ACM SIGSOFT SEN*, 30(4):1, 2005.

[16] K. G. Larsen and B. Thomsen. A Modal Process Logic. In *Proc. of LICS'88*, pages 203–210, 1988.

[17] P. Larsen. The Expressive Power of Implicit Specifications. In *Proc. of ICALP'91*, volume 510 of *LNCS*, pages 204–216, 1991.

[18] L. Lin and J. Zhi. Integrating Goals and Problem Frames in Requirements Analysis. In *Proc. of RE'06*, pages 349–350, 2006.

[19] P. Mader, O. Gotel, and I. Philippow. Enabling Automated Traceability Maintenance by Recognizing Development Activities Applied to Models. In *Proc. of ASE'08*, pages 49 –58, 2008.

[20] N. Maiden, S. Jones, S. Manning, J. Greenwood, and L. Renou. Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study. In *Proc. of CAiSE'04*, volume 3084 of *LNCS*, pages 3–21, 2004.

[21] N. Maiden, J. Lockerbie, D. Randall, S. Jones, and D. Bush. Using Satisfaction Arguments to Enhance i* Modelling of an Air Traffic Management System. In *Proc. of RE'07*, pages 49–52, 2007.

[22] F. Moisiadis. Prioritising Scenario Evolution. In *Proc. of RE'00*, pages 85–94, 2000.

[23] J. Noppen, P. van den Broek, and M. Aksit. Software Development with Imperfect Information. *J. Soft Computing*, 12(1):3–28, 2008.

[24] C. Potts and G. Bruns. Recording the Reasons for Design Decisions. In *Proc. of ICSE'88*, pages 418–427, 1988.

[25] R. Salay, M. Chechik, and J. Gorzny. Towards a Methodology for Verifying Partial Model Refinements. In *Proc. of VOLT'12*, 2012.

[26] R. Salay, M. Famelis, and M. Chechik. Language Independent Refinement Using Partial Modeling. In *Proc. of FASE'12*, volume 7212 of *LNCS*, pages 224–239, 2012.

[27] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *Proc. of RE'10*, pages 95–103, 2010.

[28] G. Spanoudakis and A. Zisman. Software Traceability: a Roadmap. *J. of Software Engineering and Knowledge*, III:1–35, 2005.

[29] S. Uchitel and M. Chechik. Merging Partial Behavioural Models. In *Proc. of SIGSOFT FSE'04*, pages 43–52, 2004.

[30] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[31] A. Yrjönen and J. Merilinna. Tooling for the Full Traceability of Non-Functional Requirements within Model-Driven Development. In *Proc. of ECMFA-TW'10*, pages 15–22, 2010.

[32] E. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proc. of RE'97*, pages 226–235, 1997.