Uncertainty Management With Partial Models

Michalis Famelis
Supervisor: Marsha Chechik

University of Toronto
Toronto, Canada
{famelis, chechik}@cs.toronto.edu

Abstract. Model-based software development inevitably involves dealing with artifacts that contain uncertainty. Yet, MDE methodologies rarely, if ever, address uncertainty in a systematic way. We propose an approach for managing uncertainty that focuses on the use of *partial models* as first-class development artifacts throughout the modeling lifecycle. We intend to develop partial modeling support for various common operations, such as reasoning and transformations, as well as methodological and tool support.

1 Problem

Model-driven software development is envisioned as an integrated process of incrementally refining models from requirements to generated code. However, as in any software engineering activity, it is necessary to support decision-making under uncertainty.

Uncertainty can arise in modeling artifacts due to a variety of reasons, such as unclear requirements [26], alternative inconsistency fixes [16,3], multiple stakeholder views [21], etc. Yet existing MDE methodologies do not robustly support the handling of artifacts that contain uncertainty. For example, model transformations can only be applied to concrete models. This renders uncertainty an undesirable characteristic that needs to be removed before resuming development.

In fact, in the face of uncertainty, modelers are forced either (a) to wait until uncertainty is resolved, (b) to handle each alternative model separately or (c) to remove uncertainty entirely before work can continue. The first option can cause inefficiency and resource under-utilization, and in general increase time-to-market. The second option is clearly intractable when the set of alternatives is large. The second option requires modelers to make provisional decisions that artificially remove uncertainty from their artifacts. This increases the risk of having to undo their work when previously unknown information becomes available. Even worse, it can mean committing too early to design decisions that cannot be reversed without significant costs, when in fact the modeler may still find it desirable to keep many alternative options open for consideration [19].

In order to avoid either of these pitfalls, modelers need ways to manage uncertainty in a systematic way; i.e., ways to accomplish modeling and development

tasks without having to remove uncertainty. In particular, modelers should be able to explicate their uncertainty about the content of their models, to reason with models that contain uncertainty, to refine and transform them, as well as have methodological and tooling support.

2 Related work

Uncertainty management has been studied in the past, often with the aim to understand and mitigate its consequences [9] or to manage its associated risks [10]. Other approaches focus on conceptually expressing uncertainty in terms of vagueness [8] or probabilities [17] and support reasoning in its presence. What is common in these techniques is that they present different strategies for the removal of uncertainty. We take a different view, aiming to support *deferring the removal of uncertainty*. In other words, we aim to develop support for a *wide range of modeling activities* that modelers should be able to accomplish regardless of the presence of uncertainty.

In our work, we focus on uncertainty expressed as a space of possibilities. In such a context, managing uncertainty requires working with representations of sets of models. Similar ideas can be found in the field of behavioral specification. For example, Modal Transition Systems (MTSs) [13] allow introduction of uncertainty about transitions on a given event, whereas Disjunctive Modal Transition Systems (DMTSs) [14] add an additional constraint that at least one of the possible transitions must be taken in the refinement. These approaches encode an *over-approximation* of a set of possible Labeled Transition Systems (LTSs) and thus reasoning over them suffers from information loss. As such, they are not well suited for the explicit uncertainty management which we want to create.

Another relevant area is product line software development [20] which captures the potential variabilities of a set of models. For example, similar to MTS and DMTSs, Featured Transition Systems (FTSs) [1] encode a set of products by annotating transitions with specific features from a feature diagram. They differ from MTSs and DMTSs in that they support *precise* representation and reasoning with a set of alternatives. In terms of representing uncertainty, product line approaches usually do not support rich expressive formalisms that can capture the various ways in which uncertainty can manifest itself. Moreover, while product line formalisms can be used to express variability, they are not well suited for the kinds of analysis, manipulation and refinement that is required if we are to fully support deferral of uncertainty resolution.

3 Proposed solution

To tackle the problem described in Section 1, we propose to define a formal artifact, partial models, that allows modelers to systematically account for uncertainty in model-based software engineering. Partial models are first-class artifacts that allow modelers to (a) explicate their uncertainty, (b) do formal reason-

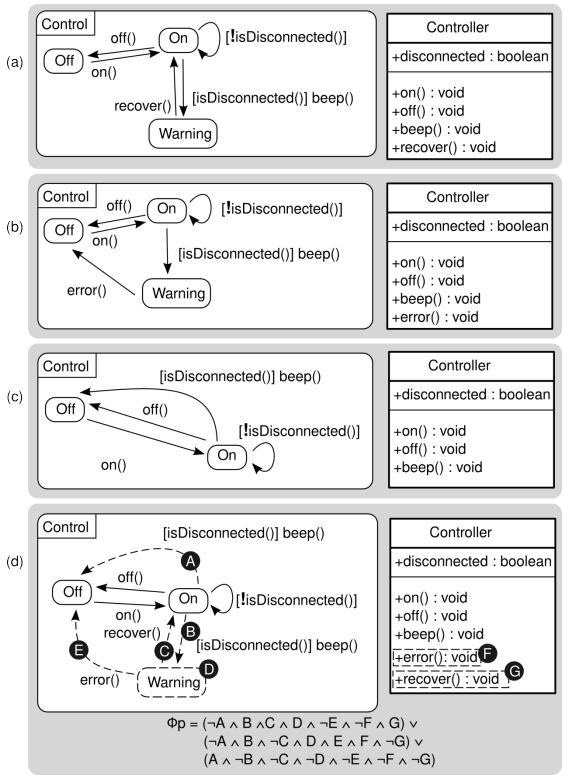

Fig. 1. (a)-(c) Alternative designs for a network controller. (e) A partial model P capturing alternatives in (a)-(c). Dashed elements are optional.

ing without needing to resolve uncertainty, (c) systematically remove uncertainty using refinement, and (d) perform transformations [5]. In addition, partial models enable more systematic treatments of uncertainty in terms of development methodologies and tooling. We explain and motivate our approach using the example of a model of a simple network controller.

Recognizing uncertainty. In our scenario, modelers have come up with three alternative designs for handling disconnects, shown in Figures 1(a)-(c). Due to unclear requirements, the modelers are not yet sure about whether it should attempt recovery, log an error or just beep and power off. They must also keep track of the affected changes to the architectural model of the controller (the relevant class diagram fragment is shown to the right for each alternative). Understanding the kinds of uncertainty that are present in the models is crucial for choosing the appropriate representations and understanding what forms of reasoning are feasible.

Explicating uncertainty. Instead of halting development while uncertainty persists or artificially removing it in an ad-hoc manner, the modelers can use the partial model P, shown in Figure 1(d), to summarize all three alternatives. The dashed elements of P represent optional elements (i.e., ones that differ be-

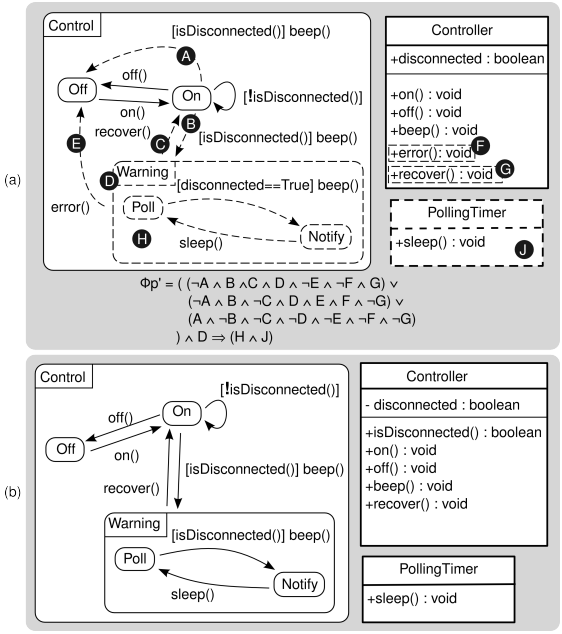

Fig. 2. (a) The model P' after detail-adding transformation of P. (b) Concretization M' of the partial model after uncertainty-removing refinement.

tween alternatives). The additional *may formula* Φ_p, shown below it in the figure, describes the allowable configurations of the optional elements. Each such configuration, called a *concretization*, corresponds to one of the alternatives. Using this representation, modelers can resume development, even in the presence of uncertainty.

Property checking. They can use P to check properties. For example, checking the property C1 *"there are no sink states"* results in the answer True, meaning that "the property holds regardless of how the partial model will be concretized". However, checking the property C2 *"no two transitions have the same source and target states"* results in Maybe, meaning "it depends on how this model will get concretized". For example, C2 holds for the concretization in Figure 1(a) but not the one in Figure 1(c).

Detail-adding transformation. Modelers can also use P as input to transformations. For example, a useful detail-adding (DA) transformation is to elaborate the `Warning` state to periodically check if the controller is still disconnected. Applying this transformation to P results in the model P', shown in Figure 2(a).

Uncertainty-reducing refinement. Once more information becomes available, modelers can refine P' to reduce uncertainty and get a less partial and ultimately concrete model. In our scenario, they decide to keep the `Warning` state and implement periodic polling. The result of the uncertainty-reducing (UR) refinement is the model M' in Figure 2(b). A UR refinement that does not retain any optional elements is called a *concretization refinement*.

This example illustrates the basic aspects of our proposed solution for explicit management of uncertainty: (a) recognizing the kinds of uncertainty in the models during the development process, (b) creating a partial model that explicitly captures this uncertainty, and (c) using the partial model as a first-class artifact that can be checked for properties, refined and transformed.

4 Expected contributions

We expect that the main contribution will be in developing a method for managing uncertainty in model-based software engineering that enables the deferral of uncertainty resolution. In the course of developing this method, we also expect to make the following contributions:

- A formalism, partial models, that allows the explication of uncertainty in modeling.
- A technique for constructing partial models from a set of alternative models.
- Techniques for the analysis of properties of partial models.
- A technique for adapting model transformations such that they can be applied to partial models.
- The identification of *patterns of uncertainty* and their impact in the modeling process.
- The creation of tool support for partial models.

5 Research plan

5.1 Preliminary work

Explicating uncertainty. In [6], we presented an algorithm for the creation of partial models that are exact abstractions of sets of alternative solutions.In [22], we outlined four different kinds of partiality: *May*, *Abs*, *Var* and *OW* (abbreviated *MAVO*)[1].

Property Checking. In [6], we presented the details and experimental evaluation of an approach for reasoning with partial models containing the May partiality. We also presented a method for using the results of property checking to perform a kind of "property-driven" refinement that can be used to provide feedback to the user in lieu of simpler, counter-example-based one. In [18], we presented a further empirical evaluation of the effectiveness of various reasoning formalisms (Alloy [11], CSP [25], SMT [2] and ASP [15]) for property checking of full MAVO models.

Transformations. In [7], we presented our preliminary results on adapting model transformations for partial models. We defined a Correctness Criterion for adapting model transformations and illustrated how to correctly adapt the semantics of transformation application. For this, we used the logical encoding of rewrite rules in *transfer predicates*.

[1] The example in Section 1 only shows the May partiality.

Tooling. We have created prototype tool support for defining partial models and checking their properties using Alloy [11], which we presented in [22]. Moreover, in [6], we built a tool that encodes May models in SAT, whereas in [18], we created translations of partial models expressed in Ecore into various reasoning formalisms.

5.2 Future steps

Recognizing Uncertainty. We are interested in the methodological implications of introducing partial models in software development. For this, we are currently working on the definition, formalization and creation of tool support for *Uncertainty Patterns*. These are methodological patterns that help modelers identify the kind of uncertainty they are faced with and offer methodological guidance, helping understand how to best explicate uncertainty and what forms of reasoning are applicable and/or can be done efficiently. We have identified some common patterns, and we are working on formalizing them. We are currently trying to determine what are the necessary properties that are required to fully define a pattern. Among these, we are trying to define canonical ways in which the patterns can be expressed using MAVO partiality.
Transformations. We want to expand on the preliminary results of our initial study [7] of adapting model transformations. In particular, we want to systematize the creation of transfer predicates using a few reusable, atomic first order logic templates for defining the different parts of graph-rewriting based model transformations. We have identified some building blocks and are putting together a method for constructing transfer predicates. A potential problem could be the efficient translation of models created with our logic-based approach back into a graph-like form. An alternative approach is to formalize partial models as an Adhesive High Level Replacement Category, to be able to reuse existing theoretical results from the field of algebraic graph transformation [4]. .We have made a few unsuccessful attempts to formalize the category of partial models, stumbling mainly on the still open problem of defining the right morphisms.
Tooling. We intend to further expand tooling support for partial models, to create full EMF [24] support for MAVO models. We are investigating the most efficient way to represent the notion of partiality in Ecore, currently focusing on an implementation based on EMF Profiles [12]. Ultimately, we want to implement support for partial models in the Model Management Tool Framework (MMTF) [23]. This would allow us to adapt existing model management techniques so that they take into account the explicit management of uncertainty.

6 Plan for evaluation and validation

In our preliminary work, we have developed experimental support to evaluate the feasibility and scalability of our techniques for reasoning with partial models. More specifically, in [6], we conducted experiments with parametrizable, randomly generated models. The models were parametrized with respect to their

size and degree of uncertainty and combined with randomly populated property templates to simulate property checking. We further refined our random model generation in [18] and created tool support to translate the generated models in various modeling formalisms in order to do comparisons between them. We are currently expanding our capabilities for experimentation with randomly generated models by integrating our generator with MMTF, so that we can easily generate partial models that are random instances of arbitrary metamodels.

To evaluate the adaptation of model transformations for partial models, in [7], we relied on SAT solving technology. However, this required us to manually encode the models. In the follow up work, we want to investigate adopting the technique that we applied in [22], where we used Alloy to verify partial model refinement. A similar strategy to the one used there can be used for verifying the correctness of adapting transformations.

An additional evaluation strategy is the development of case studies. We developed a case study in [6], where we applied our property checking techniques to a real world open source project. This allowed us to triangulate with the results from our experimentation with randomly generated models.

Ultimately, we want to evaluate our approach with a comprehensive case study from the automotive domain. An initial candidate is the case study for the controller of an electric-powered car window. Our case study should demonstrate the explicit management of uncertainty throughout the MDE development process. In it, we should demonstrate the use of partiality as uncertainty appears and/or gets propagated via transformation in models of requirements, specification, implementation and deployment.

7 Conlusion

Existing modeling techniques, tools and methodologies lack a method for supporting the deferring of the resolution of uncertainty. We propose to develop an approach for the explicit and systematic management of uncertainty in MDE. Our approach focuses on the use of partial models as formal, first-class artifacts throughout the engineering life-cycle. For that we propose to create support for various modeling activities, such as reasoning and transformation. By creating methodological and tool support for the management of uncertainty, we aim to improve the quality of produced software, while reducing time-to-market.

References

1. A. Classen, P. Heymans, P.Y. Schobbens, A. Legay, and J.F. Raskin. "Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines". In *Proc. of ICSE'10*, pages 335–344, 2010.
2. L. De Moura and N. Bjørner. "Satisfiability Modulo Theories: Introduction and Applications". *Commun. ACM*, 54(9):69–77, September 2011.
3. A. Egyed, E. Letier, and A. Finkelstein. "Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models". In *Proc. of ASE'08*, pages 99–108. IEEE Computer Society, 2008.

4. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 1 edition, 2006.
5. M. Famelis, S. Ben-David, M. Chechik, and R. Salay. "Partial Models: A Position Paper". In *Proc. of MoDeVVa'11*, pages 1–6, 2011.
6. M. Famelis, M. Chechik, and R. Salay. "Partial Models: Towards Modeling and Reasoning with Uncertainty". In *Proc. of ICSE'12*, 2012.
7. M. Famelis, M. Chechik, and R. Salay. "The Semantics of Partial Model Transformations". In *Proc. of MiSE'12*, 2012.
8. T. Herrmann. *Handbook of Research on Socio-Technical Design and Social Networking Systems*, chapter Systems Design with the Socio-Technical Walkthrough, pages 336–351. 2009.
9. 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.
10. Sh. Islam and S. H. Houmb. "Integrating Risk Management Activities into Requirements Engineering". In *Proc. of RCIS'10*, pages 299–310, 2010.
11. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.
12. Ph. Langer, K. Wieland, M. Wimmer, and J. Cabot. "From UML Profiles to EMF Profiles and Beyond". In *Objects, Models, Components, Patterns*, volume 6705, pages 52–67. Springer Berlin / Heidelberg, 2011.
13. K. G. Larsen and B. Thomsen. "A Modal Process Logic". In *Proc. of LICS'88*, pages 203–210, 1988.
14. P. Larsen. "The Expressive Power of Implicit Specifications". In *Proc. of ICALP'91*, volume 510 of *LNCS*, pages 204–216, 1991.
15. V. W. Marek and M. Truszczynski. "Stable Models and an Alternative Logic Programming Paradigm". *CoRR*, cs.LO/9809032, 1998.
16. C. Nentwich, W. Emmerich, and A. Finkelstein. "Consistency Management with Repair Actions". In *Proc. of ICSE'03*, pages 455–464. ACM Press, 2003.
17. J. Noppen, P. van den Broek, and M. Aksit. Software Development with Imperfect Information. *J. Soft Computing*, 12(1):3–28, 2008.
18. P. SaadatPanah and M. Famelis and J. Gorzny and N. Robinson and M. Chechik and R. Salay. "Comparing the Effectiveness of Reasoning Formalisms for Partial Models", 2012. submitted.
19. Marian Petre. "Insights from Expert Software Design Practice". In *Proc. of FSE'09*, 2009.
20. K. Pohl, G. Böckle, and F. Van Der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer-Verlag New York Inc, 2005.
21. M. Sabetzadeh and S. Easterbrook. "View Merging in the Presence of Incompleteness and Inconsistency". *Requirements Engineering*, 11(3):174–193, 2006.
22. R. Salay, M. Famelis, and M. Chechik. "Language Independent Refinement using Partial Modeling". In *Proc. of FASE'12*, 2012.
23. R. Salay, J. Mylopoulos, and S. M. Easterbrook. "Using Macromodels to Manage Collections of Related Models". In *Proc. of CAiSE'09*, pages 141–155, 2009.
24. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2. edition, 2009.
25. Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London San Diego, 1993.
26. A. van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.