# MU-MMINT: an IDE for Model Uncertainty

Michalis Famelis, Naama Ben-David, Alessio Di Sandro, Rick Salay, and Marsha Chechik
{famelis, naama, adisandro, rsalay, chechik}@cs.toronto.edu
University of Toronto

***Abstract*—Developers have to work with ever-present design-time uncertainty, i.e., uncertainty about selecting among alternative design decisions. However, existing tools do not support working in the presence of uncertainty, forcing developers to either make provisional, premature decisions, or to avoid using the tools altogether until uncertainty is resolved. In this paper, we present a tool, called MU-MMINT, that allows developers to express their uncertainty within software artifacts and perform a variety of model management tasks such as reasoning, transformation and refinement in an interactive environment. In turn, this allows developers to defer the resolution of uncertainty, thus avoiding having to undo provisional decisions. See the companion video: http://youtu.be/kAWUm-iFatM**

## I. INTRODUCTION

Software developers often face uncertainty (also called *design-time uncertainty*) about selecting among alternative design decisions. The multitude of possible designs requires them to make (and keep track of) many provisional decisions. In fact, the skillful management of such provisionality is a defining characteristic of expert software designers [13]. However, existing tools do not support working in the presence of uncertainty. This forces developers to either refrain from using their tools until uncertainty is resolved, or to make provisional decisions and attempt to keep track of them in case they need to be undone. These options lead to either under-utilization of resources or potentially costly re-engineering attempts. In previous work [16], [7], [8], we have demonstrated that there exists a viable third approach, centered around explicit uncertainty modeling. Specifically, the different alternative design decisions are encoded in a *partial model* which can be used to perform tasks such as reasoning, refinement and transformation early in the design process. This allows deferring the resolution of uncertainty for as long as necessary, while still being able to work in its presence.

In this paper, we present a tool, called MU-MMINT[1], that realizes this vision of decision deferral through comprehensive management of models with uncertainty. The rest of this paper is organized as follows: Sec. II gives a motivating scenario, Sec. III describes the tool's features and Sec. IV its implementation details, Sec. V describes how we evaluated MU-MMINT. The paper concludes with discussion in Sec. VI.

## II. MOTIVATING EXAMPLE

We assume that a team is developing a protocol for peer-to-peer downloads, called PtPP. The current state of the protocol is shown in Fig. 1(a). The state machine fragment on the left captures the parts of the design they have already agreed upon, expressed using the simplified metamodel in Fig. 1(c). Specifically, there is a state `Idle`, a state `Leeching` (sharing and downloading an incomplete file), and a state `Seeding` (sharing a complete file). Sharing can be canceled, triggering the action `cancel()`, while when `Leeching` is initiated, the action `start()` is executed. On the right of Fig. 1(a) we list the things the team is uncertain about, i.e., the three design decisions that they have yet to make: (a) Can users initiate seeding? (b) Can downloads be restarted? (c) What policy is followed when `Leeching` ends? We then assume the team made a series of provisional choices regarding the three design decisions, as shown in Fig. 1(b). Specifically: (a) Users can initiate seeding (indicated by the transition with action `share()`). (b) Downloads are not restarted (no effect on the model). (c) The "selfish" policy is adopted: at the end of `Leeching`, PtPP goes directly to `Idle`, not letting other connected peers to complete their work (indicated by the transition with action `completed()`).

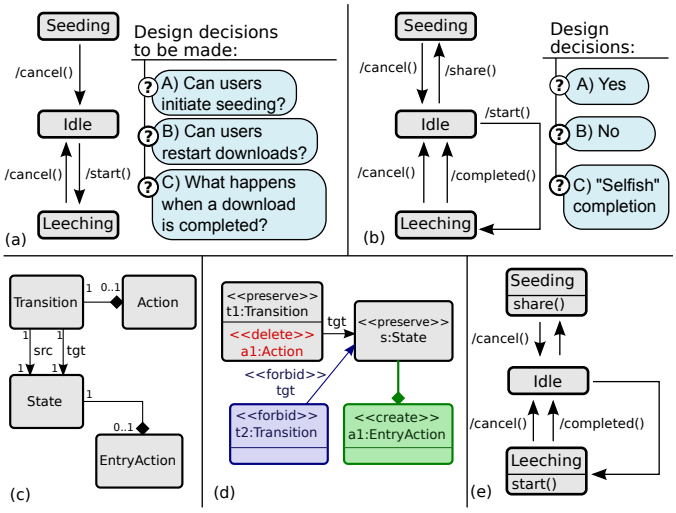



Fig. 1. (a) Developing PtPP. Left: what is known, right: design decisions. (b) P2P model after making design decisions. (c) Simplified state machine metamodel. (d) "Fold Single Entry Action" refactoring transformation. (e) The PtPP model after refactoring.

In the course of developing PtPP, the team may do additional work on the model. *Refactoring* is a common task done during development. For example, the team could refactor PtPP with a transformation such as the rewriting rule "Fold Single Entry Action" (FSEA) shown in Fig. 1(d). FSEA is expressed in the notation used by the Henshin model transformation engine [1]. The rule tries to match a `State s` that has

[1]Available at: http://github.com/adisandro/mmint

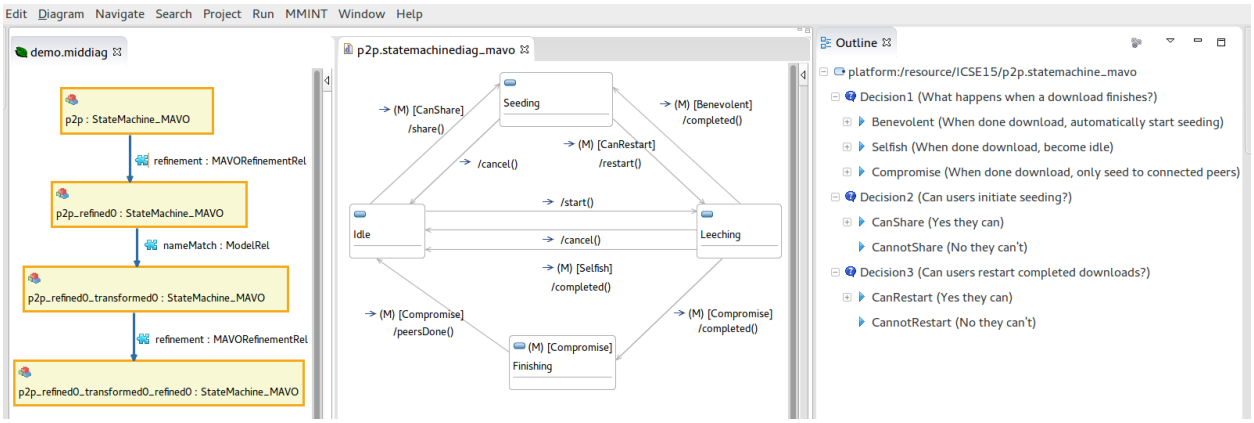



Fig. 2. Screenshot of MU-MMINT. Left: interactive workspace showing different versions of PtPP. Middle: Graphical partial model $M_{p2p}$ for PtPP. Model elements that reify solutions in the uncertainty tree are annotated with (M). Right: Decision tree for $M_{p2p}$.

only one incoming `Transition t1`. The "negative application condition" (indicated by $<<$ `forbid` $>>$) stops the refactoring from being applied if there exists a second incoming `Transition t2`. Otherwise, it deletes `Action a1` of `t1` (indicated by $<<$ `delete` $>>$) and adds a new `EntryAction` with the same name (indicated by $<<$ `create` $>>$), associating it with `s`. The result of refactoring PtPP with FSEA is shown in Fig. 1(e), where the actions `share()` and `start()` have been respectively folded into the states `Seeding` and `Leeching`.

After making design decisions and further modifying the model, the team may decide to perform some analysis on their model. In our example, they check the property $P_1$: *"no two transitions enabled in the same state can lead to the same target state"*. The PtPP model that the team has created after making design decisions and subsequent modifications (shown in Fig. 1(e)) does not satisfy this property due to the two outgoing transitions from `Leeching`, with `cancel()` and `completed()` respectively. In other words, some of the design decisions taken early on proved to be *premature*, ultimately forcing the team to undo a lot of work.

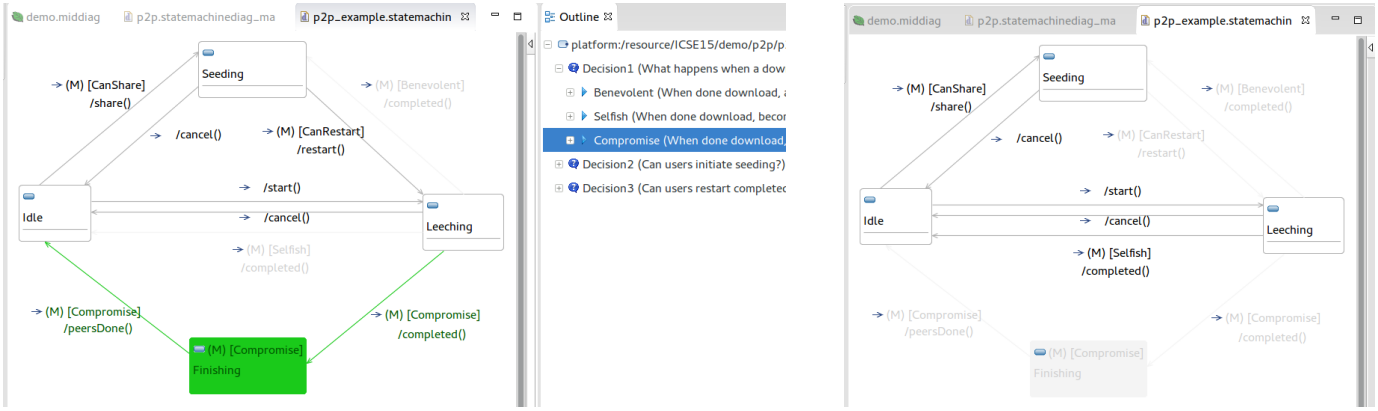
Our tool, MU-MMINT, facilitates a different way of doing development. Using it, developers can check properties and remove unwanted possibilities early in the process. By additionally providing support for common tasks such as transformations, we can allow developers to defer the resolution of uncertainty for as long as they need.

## III. TOOL DESCRIPTION

MU-MMINT is a tool for managing models [2] that contain uncertainty about how to address a set of design questions. Such models are called *partial models* [6]. They succinctly encode a set of *concretizations*, i.e., ways to resolve uncertainty. In previous work, we have studied how to effectively articulate [11], verify [7], refine [16], and transform partial models [8]. MU-MMINT was created as an Integrated Development Environment (IDE) that combines these techniques in one coherent unit. The workspace of MU-MMINT is an interactive megamodel [3], shown in the left panel of Fig. 2, which allows modelers to create, manipulate and interact with models using the operations described below.

**Articulating Uncertainty.** At the start of the PtPP scenario, the team's design may be separated into known and unknown parts, as shown in Fig. 1(a). In MU-MMINT, this information is captured in a single partial model $M_{p2p}$, shown in the middle and right panels of Fig. 2. The right panel contains the partial model's *uncertainty tree*. It consists of a list of *decision* elements, each of which can have any number of children representing mutually exclusive *alternative solutions*. For $M_{p2p}$, the tree contains the three decisions listed in Fig. 1(a). For each decision, the team explicates the possible solutions. E.g., the decision about the policy when a download completes involves selecting among three alternative solutions: (a) the "benevolent" option which automatically starts `Seeding`, (b) the "selfish" option described earlier, and (c) a "compromise" option where no new connections are accepted while waiting for existing peers to complete their copies. The middle panel shows the graphical part of $M_{p2p}$. It consists of a model expressed in the same concrete syntax as the original PtPP model fragment, with the addition of annotations to some elements. These are called *May elements*; they reify the various alternative solutions and are included in the final version of the model only if their respective solution is selected. E.g., the state `Finishing` and its associated transitions are annotated with (M) and [`Compromise`] to indicate that they are part of that particular solution to the policy decision. MU-MMINT supports highlighting the elements reifying a particular solution, as shown in Fig. 3(a). This allows developers to quickly examine the various possibilities in the partial model.

**Checking and Enforcing Properties.** MU-MMINT supports checking of syntactic properties, such as the property $P_1$ in the PtPP example. This kind of property checking is done using the technique described in [7], where the objective is to determine whether a property $\Phi$ holds for all, some, or none of the partial model's concretizations. To check $\Phi$, the partial model's uncertainty tree is first translated into a propositional *May formula* using propositional variables to encode the presence of May elements. The May formula and the model's well-formedness constraints are then combined




(a) (b)

Fig. 3. (a) Highlighting the elements that reify the "compromise" alternative solution. (b) Visualizing a counterexample to property $P_1$.

with $\Phi$ and $\neg\Phi$ to create two queries to the Z3 SMT solver [5]. Depending on the satisfiability of these two queries, the result of the property checking can be True ("the property holds for all concretizations"), False ("it does not hold for any concretization") or Maybe ("it only holds for some"). For the last two cases, MU-MMINT can display a counterexample concretization. E.g., Fig. 3(b) shows that the model in Fig. 1(e) is a counterexample for $P_1$. Modelers can choose to restrict the possible concretizations of the partial model by enforcing the property $\Phi$ thus eliminating some design alternatives. This is called *property-based refinement* [7] and is done by conjoining $\Phi$ with the May formula. MU-MMINT automatically recalculates the uncertainty tree and updates the graphical part of the partial model, while keeping traceability between the different versions in its workspace, as shown in the left panel in Fig. 2.

Partial models are a language-agnostic approach for expressing uncertainty; checking properties that require awareness of language semantics is thus not supported out-of-the-box in MU-MMINT. Instead, it provides an interface for adding custom plugins implementing "lifted" [8] versions of semantics-aware analysis techniques. E.g., we have created such plugins for supporting early requirements analysis [10].

**Applying Transformations.** Transformations like the FSEA rule in Fig. 1(d) are typically expected to work on models that do not contain uncertainty. This makes it difficult to transform a partial model: consider for example the state `Seeding` in $M_{p2p}$, shown in the middle panel of Fig. 2. Both its incoming transitions are May elements, and the uncertainty tree allows creating concretizations with none, one and both of them. Should FSEA be applied to this state? To correctly answer this question, we must ensure that the transformation is "lifted", i.e., adapted for uncertainty [8]. In MU-MMINT, when a lifted transformation is applied to a partial model $M_1$ that encodes a set $S_1$ of concretizations, a new partial model $M_2$ is created such that its set $S_2$ of concretizations is the same as if we had applied the non-lifted transformation to each concretization of $M_1$ separately [8]. MU-MMINT uses the Henshin graph transformation engine [1]. Henshin transformations do not need to be rewritten or manually adapted for uncertainty. Instead, lifting of transformations is handled internally by MU-MMINT. Thus, modelers can fully leverage model transformations early in the lifecycle.

**Making Decisions.** In addition to supporting decision deferral, when the time is right MU-MMINT also supports making design decisions in a systematic way. Once the modeler has enough information to make a decision, she can simply select the desired alternative solution in the uncertainty tree thus triggering a "decision-based" refinement [16] of her partial model: (a) May elements that reify her chosen solution are turned into regular model elements (b) May elements reifying alternative solutions are removed from the model, and (c) the resolved decision is removed from her uncertainty tree. As with property-based refinement, MU-MMINT keeps explicit traceability between versions of the model in its workspace.

## IV. Implementation

MU-MMINT was developed in Java by extending MMINT[2], an interactive environment for model management [2] developed at the University of Toronto [14]. MMINT consists of 140 KLOC, 80% of which is automatically generated. MU-MMINT has an additional 10 KLOC, 60% of which is generated. MMINT uses the Eclipse Modeling Framework (EMF) [17] to express models and the Eclipse Graphical Modeling Framework (GMF) [9] for creating graphical editors. MU-MMINT extends MMINT's data model with EMF structures for uncertainty-related constructs. It also hooks specialized GMF diagram elements and views to represent partial models. MU-MMINT uses the Z3 SMT solver [5] for performing reasoning tasks, and adapts parts of the Henshin [1] engine for lifted graph transformations [8].

## V. Evaluation

MU-MMINT integrates previously developed (and evaluated) techniques for analyzing and manipulating partial models. The feasibility and scalability of property checking and

[2]Previously known as *Model Management Tool Framework* (MMTF).

TABLE I
CASE STUDIES CONDUCTED WITH MU-MMINT.

| | Language | #Elements | #Decisions | Average #Solutions/Decision | #Mays | $T_{PC}$ | $T_{PBR}$ | $T_{CE}$ | $T_{DBR}$ |
|---|---|---|---|---|---|---|---|---|---|
| PtPP | State machine | 13 | 3 | 2.33 | 7 | 0.27 | 0.73 | 0.58 | 0.50 |
| Ecore Metamodel | Class diagram | 86 | 13 | 2 | 15 | 0.25 | 1.34 | 0.56 | 0.79 |
| UMLet | Sequence diagram | 101 | 1 | 12 | 52 | 0.38 | 2.45 | 0.88 | 1.34 |

**Legend.** $T_{PC}$: time to check a property (msec), $T_{PBR}$: time to do property-based refinement (msec), $T_{CE}$: time to generate a counterexample (msec), $T_{DBR}$: time to do decision-based refinement (msec)

property-based refinement were studied in [7] both experimentally and with a case study; the scalability of lifted transformation was studied in [8] with a case study; the performance of refinement transformations was studied in [15]. To evaluate MU-MMINT as an integrated whole we deployed it in three modeling settings: (a) the PtPP example, (b) a case study described in [7] where a bug fix in the UMLet open source project results in several alternative repairs, and (c) a case study described in [8] where we manually introduced uncertainty to the Ecore metamodel. For each case study, we modeled all artifacts in MU-MMINT, explicating the uncertainty in the uncertainty tree. While modeling each one, we recorded the size of each partial model (number of elements), the number of decision nodes in the uncertainty tree, the average number of alternative solutions per decision, and the number of model elements that have a (M) annotation. In addition, we computed time it took for the basic analysis operations supported by MU-MMINT: an average time to check a property ($T_{PC}$), an average time to perform property-based refinement ($T_{PBR}$), an average time to produce and visualize a counterexample ($T_{CE}$), and an average time to refine the partial model based on a modeler decision ($T_{DBR}$). Table I summarizes the results. Despite the additional language constructs added by MU-MMINT (tree of decisions and alternative solutions), the results are consistent with previous observations in [7], [8].

## VI. DISCUSSION AND CONCLUSION

The main limitation of MU-MMINT is the expressiveness of the visual syntax used to represent uncertainty in partial models. Our original intent was to realize the syntax MAVVIS [11] which was designed using the theory of visual notations developed by D. Moody [12]. This was not possible because of our reliance on GMF. While GMF allows easy integration of existing model editors to MU-MMINT, it only supports a limited visual vocabulary. To overcome this, we introduced uncertainty trees, as shown in the right panel of Fig. 2. This approach attempts to ameliorate the limitations of GMF by using a separate dialog, exclusively dedicated to modeling uncertainty at a higher level of abstraction. This idea followed from a preliminary empirical study with human participants [11] that pointed us to the need to elevate decisions and alternative solutions to first class concepts in partial modeling. The uncertainty tree approach is a simplification compared to our earlier work with partial models where we used the propositional May formula described in Sec. III. While the uncertainty tree approach is less expressive than fully fledged propositional logic, it has nonetheless been shown that humans can graphically create and manipulate many useful formulas [4].

MU-MMINT was created with the aim to integrate existing partial modeling techniques in an IDE for uncertainty. MU-MMINT also introduces new features such as the uncertainty tree, alternative solution highlighting, and refinement based on single-click decision making, which help enhance the usability of partial modeling. Using MU-MMINT, modelers can articulate their uncertainty about design decisions in partial models. By supporting tasks such as property checking, refinement and transformations, it allows them to work in the presence of uncertainty, while postponing the resolution of design decisions.

## REFERENCES


[1] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer. Henshin: Advanced Concepts and Tools for In-place EMF Model Transformations. In *Proc. of MODELS'10*, pages 121–135, 2010.

[2] P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proc. of CIDR'03*, 2003.

[3] J. Bézivin, F. Jouault, and P. Valduriez. On the need for megamodels. In *In Proc. of OOPSLA/GPCE'04*, 2004.

[4] K. Czarnecki and A. Wasowski. Feature Diagrams and Logics: There and Back Again. In *In Proc. of SPLC 2007*, pages 23–34, Sept 2007.

[5] L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Proc. of TACAS'08*, LNCS, pages 337–340, 2008.

[6] M. Famelis, S. Ben-David, M. Chechik, and R. Salay. Partial Models: A Position Paper. In *Proc. of MoDeVVa'11*, pages 1–6, 2011.

[7] M. Famelis, M. Chechik, and R. Salay. Partial Models: Towards Modeling and Reasoning with Uncertainty. In *Proc. of ICSE'12*, pages 573–583, 2012.

[8] M. Famelis, R. Salay, A. Di Sandro, and M. Chechik. Transformation of Models Containing Uncertainty. In *Proc. of MODELS'13*, pages 673–689, 2013.

[9] R. Gronback. *Eclipse Modeling Project*. Addison Wesley, 2009.

[10] J. Horkoff, R. Salay, M. Chechik, and A. Di Sandro. Supporting Early Decision-Making in the Presence of Uncertainty. In *Proc. of RE'14*, pages 33–42, 2014.

[11] M. Famelis and S. Santosa. "MAV-Vis: a Notation for Model Uncertainty". In *Proc. of MiSE'13*, 2013.

[12] D. Moody. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *TSE*, 35(6):756–779, 2009.

[13] M. Petre. Insights from Expert Software Design Practice. In *Proc. of ESEC/FSE'09*, pages 233–242, 2009.

[14] R. Salay, M. Chechik, S. Easterbrook, Z. Diskin, P. McCormick, S. Nejati, M. Sabetzadeh, and P. Viriyakattiyaporn. An Eclipse-Based Tool Framework for Software Model Management. In *Proc. of eTX Wrksp. at OOPSLA'07*, pages 55–59, 2007.

[15] R. Salay, M. Chechik, M. Famelis, and J. Gorzny. A Methodology for Verifying Refinements of Partial Models. *J. of Object Technology*, 2015.

[16] R. Salay, M. Famelis, and M. Chechik. Language Independent Refinement using Partial Modeling. In *Proc. of FASE'12*, pages 224–239, 2012.

[17] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.