# From Requirements to Partial Behavior Models: an Iterative Approach to Incremental Specification Refinement

Ivo Krka
Department of Computer Science
University of Southern California
Los Angeles, CA, USA
krka@usc.edu

## ABSTRACT

In this thesis, I will improve the state-of-the-art for capturing, analyzing, and refining functional requirements by providing support for synthesizing, analyzing, and refining Modal Transition Systems.

## Categories and Subject Descriptors

D.2 [**Software Engineering**]: Requirements/Specifications;
D.2 [**Software Engineering**]: Software Architectures

## Keywords

modal transition systems, synthesis, elicitation, refinement

## 1. INTRODUCTION

During early stages of a software system's life cycle, the requirements collected from the stakeholders are transformed to more formal yet straightforward and intuitive specifications (e.g., use cases, scenarios, and goal models [19]). Such specifications have proven useful for different development activities including preliminary architectural design, communication with stakeholders, and capturing different viewpoints. These early specifications provide only a partial description of a system-to-be; they focus on particularly interesting aspects of a system, allow deferring some decisions, and omit unnecessary details. Conversely, more detailed and rigorous approaches to architectural design increasingly depend on comprehensive behavioral models (e.g., LTS [11]) that, at a chosen level of abstraction, comprehensively specify how a component or a system should behave. These models can then be leveraged for automated verification, simulation and code generation; they can also serve as a "blueprint" for the subsequent implementation.

With my thesis, I intend to bridge the gap between the early, inherently partial specifications and comprehensive behavioral models. To accomplish this objective, three main problems need to be tackled:

1. **Problem 1**: How to *synthesize* behavioral models that precisely capture the partial requirements specifications (i.e., support those behaviors that are explicitly required in the requirements and prohibit the behaviors that conflict the requirements)?
2. **Problem 2**: How to *elicit* new and *elaborate* existing requirements based on the synthesized models, in order to obtain a more complete behavioral specification?
3. **Problem 3**: How to support *refinement* of the synthesized models when the additional elicited requirements have different scope (component vs. subsystem vs. system model) or level of abstraction (detailed vs. abstract model)?

Several solutions have been proposed for similar and related problems, but they overlook some of the challenges that arise when working with partial specifications. For example, researchers have worked on prospective solutions to Problem 1 – synthesizing behavioral models from requirements (surveyed by Liang et al. [10]). Most of the existing techniques focus on synthesizing closed behavioral models that model only required behavior (notable exceptions are [14, 18]). Hence, these techniques overlook the inherent partiality of the requirements, which, often intentionally, leave some behavior as neither required nor prohibited (i.e., *maybe*). Without considering the maybe behavior, incremental requirements elaboration and refinement becomes limited to analyzing the required behavior [16].

Problem 2 has been considered for different types of requirements specifications. For example, existing techniques support elaboration, analysis, and refinement of goal models [19]. These and similar techniques provide guidelines, patterns, and strategies for specifying correct requirements of a certain type. Conversely, my work is concerned with the prospect of extracting and proposing new requirements based on the unknowns in the existing requirements. Lastly, Problem 3 is related to techniques for merging and refining behavioral models of different viewpoints (e.g., [15, 3]). However, these techniques provide no support for incorporating requirements that relate to different parts of the system (e.g., a subsystem and a component in the subsystem).

Modal Transitions Systems (MTS) have been proposed as a suitable formalism for capturing different types of partial requirements specifications (scenarios and properties) in a single behavioral model [18]. Scenarios are explicitly incorporated into an MTS as *required* behavior, while the behavior that is not proscribed by properties is captured as *maybe* behavior. I believe that this dichotomy is especially suitable in the context of requirements elaboration where the maybe behavior can be used to elicit new and elaborate existing requirements. Furthermore, by analyzing the required behav-

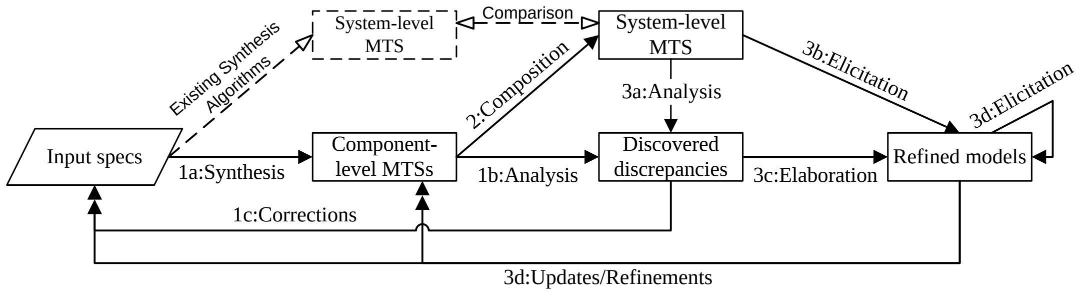

**Figure 1: Outline of the envisioned iterative approach.**

ior we can detect undesirable, currently supported behavior (e.g., implied scenarios) and explore high-level goal satisfaction. The existing MTS synthesis approaches [14, 18] do not appropriately solve the aforementioned research problems because: (1) they synthesize system-level, as opposed to component-level, models, hence suffering from potential scalability issues; (2) they only informally discuss support for incremental requirements elaboration; and (3) provide limited support for model refinement based on requirements of different scope and levels of abstraction.

My thesis is going to be based on the following three hypotheses, which, granted they are valid, mitigate the identified research problems:

1. **Hypothesis 1**: *Precise component-level MTSs* can be *synthesized* in a scalable manner from system requirements specifications.
2. **Hypothesis 2**: *Requirements elicitation and elaboration* can be effectively supported by *analyzing MTS models* using (1) heuristics that help to refine the maybe behavior (e.g, property patterns [2]); and (2) heuristics that discover undesirable required behavior (e.g., implied scenarios [17]).
3. **Hypothesis 3**: Requirements of different scope and levels of abstraction can be incorporated into an *existing MTS specification* by *semi-automatically propagating the refinements* from (1) subsystem to component models; and (2) abstract to detailed models.

In Section 2, I describe the techniques that will test the hypotheses. I outline the evaluation plans in Section 3, and describe the progress made so far in Section 4.

## 2. APPROACH DETAILS

In this section, I will describe the intended approach, which is depicted in Figure 1. My approach will take positive and negative scenarios, goals (properties), and operational requirements (pre- and postcondition constraints on system operations) as inputs. The assumptions are twofold: (1) the scenarios are given in terms of event sequences, and (2) goals and constraints are given as logical formulas on system state variables (e.g., FLTL fluents [4]). Neither should be restrictive because such specifications are widely used [19]. In the following three sections, I describe solutions that attempt to mitigate the research problems outlined in Section 1.

### 2.1 Component-Level MTS Synthesis

As the first step of my approach, I propose an algorithm for synthesizing MTS models of system components (*task 1a* in Figure 1). I choose to create component-level models as they capture the properties arising from the system decomposition more precisely than the system-level models produced using existing techniques. Additionally, directly synthesizing component-level models makes the algorithm more scalable to large specifications. Synthesizing component-level models also helps to expose certain types of specification discrepancies (*tasks 1b-c*) that are overlooked when using only the system-level viewpoint (details in [7]).

The resulting component MTSs comprise required behavior from the positive scenarios and maybe behavior which does not conflict the negative scenarios and operational requirements. For each system component, the synthesis algorithm first creates an MTS that captures all the behavior that does not conflict the operational requirements (similar to [1]). The states in a component model correspond to the different values of system state variables observable by that component. A transition labeled with a particular operation is created between two states when that operation's precondition is satisfiable in the source state, and the postcondition is satisfiable in the destination state. These steps assure that the created model does not conflict the operational requirements. Next, the resulting MTS is refined with required behavior according to the positive scenarios. The algorithm automatically annotates the scenarios with state information and queries the architect for their validity. These annotations are used to find and refine to required those MTS maybe transitions that are traversed during scenario execution. Finally, all the undesirable transitions according to negative scenarios are removed from the component models.

Figure 2a illustrates how the synthesis works on a simple Client-Server system. The top frame in Figure 2a contains the initial requirements, while the bottom frame depicts the synthesized MTS *Client* (transitions ending with '?' are maybe transitions). *Client* has states annotated with the value of *requestPending* state variable. The transition $S_0 \overset{responseData}{\longrightarrow_m} S_0$ is a maybe transition because it does not violate *responseData*'s constraints, but is also not explicitly required in a scenario. Conversely, $S_0 \overset{requestData}{\longrightarrow_r} S_1$ became required to support the given scenario. Subsequent requirements elicitation should then determine whether the only maybe transition should be required or removed.

As part of my thesis, I will identify and classify the types of requirements flaws that can be discovered by analyzing component MTSs (*task 1b*). I will also define guidelines for solving these flaws (*task 1c*). For example, analysis of component MTSs can detect (1) overly constraining requirements (e.g., operational requirements disallow a scenario), (2) wrong scenarios,and (3) synchronization problems (e.g., the internal component states are inconsistent). Example resolution strategies include (1) relaxation of constraints,

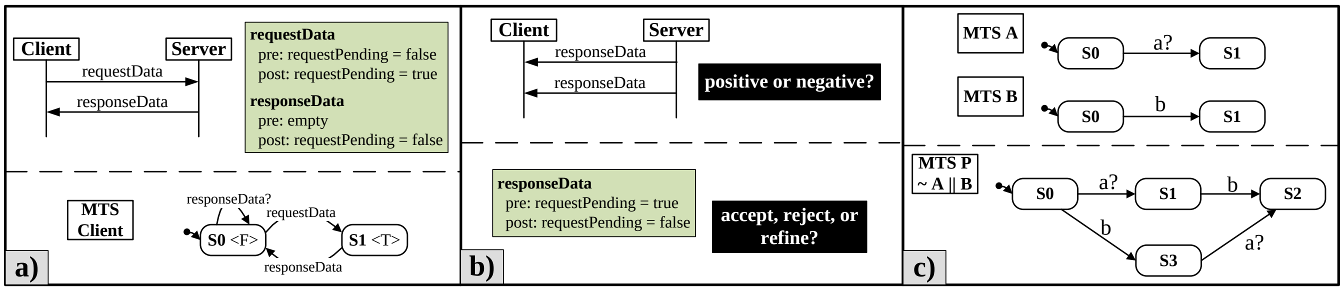

**Figure 2: Illustration of the different facets of my approach.**

(2) removal of incorrect scenarios, and (3) addition of synchronization events.

## 2.2 Requirements Elaboration

As a second step, my thesis will contribute a technique for eliciting new and elaborating existing requirements based on the synthesized MTS models (*tasks 3a-3d* from Figure 1). The synthesized component MTSs, their composition into subsystem MTSs (*task 2*), and their abstract models (i.e., models with hidden irrelevant actions), can contain a number of maybe transitions. These transitions correspond to behavior that is (1) not required in any positive scenario, but also (2) not prohibited by other requirements.

A trivial approach to eliciting new requirements from an MTS model would be manual inspection and per-case decisions whether a maybe transition should be removed or become required; this would obviously be a tedious and an error-prone process. Therefore, I will investigate ways of guiding the refinement process with focused model exploration, learning techniques (e.g., [1]), and property pattern queries (e.g., [2]); the idea is to apply techniques that can reduce the number of maybe transitions. Figure 2b depicts some of the requirements that could be proposed to refine the *Client* MTS. For example, my technique would try to resolve the maybe transition on *responseData* (Figure 2a) by proposing a new scenario that has two adjacent *responseData* invocations. In case that the scenario is confirmed as positive, the technique would further ask whether a longer sequence (or an unbounded loop) of *responseData* invocations should be allowed to happen.

Based on the *Client* MTS, my technique will be able to propose a more elaborate precondition for *responseData*; this is done by only considering the state variable values before required transitions on *responseData* (Figure 2). Furthermore, analysis of state variable values that are reachable through required and some subset of maybe transitions may suggest changes to the system goals or specification of new goals. The role of the architect in this process would be to answer whether the proposed requirements are valid or not (*tasks 3b-d*). In case of a positive answer, maybe behavior that violates the refined specification can be eliminated. When an engineer's answer to a query is negative or undecided, the maybe behavior in question would be displayed as a scenario that should be classified either as a legal or an illegal behavioral sequence.

I also plan to work on detecting flaws discoverable using the synthesized MTSs (*task 3a*). An example flaw is an implied scenario (i.e., undesirable required behavior) that can appear as a consequence of imprecise specification. An implied scenario can be avoided, for example, by eliciting an additional state variable and then elaborating the operational requirements in a way that prevents the undesirable behavior. As part of my research I will explore resolution strategies for implied scenarios in partial behavior models that will complement the state-of-the-art for detecting implied scenarios using LTS models [17].

## 2.3 Refinement Support

In Section 2.2, I mentioned that requirements can vary in their scope (e.g., a component or a subsystem) and level of abstraction (e.g., high-level system behavior considering only a couple of important operations). Hence, elicitation of new requirements calls for suitable MTS models. A subsystem model is obtained as a parallel composition [11] of component models. Similarly, an abstract model can be obtained by hiding a set of actions from an MTS. Once an appropriate MTS is obtained, new requirements can be elicited, and the MTS further refined using the technique from Section 2.2. Refining an MTS model that is related to other models (e.g., a subsystem MTS is related to component MTSs) should be reflected by appropriately refining the related models. However, refinement propagation between related partial behavior models is, to a large extent, an unsolved problem. In this thesis, I intend to provide support for propagating simple refinements (single transition and state refinements) of a composite or an abstract MTS to its related MTSs. This is an important part of my thesis because it demonstrates the feasibility of applying the proposed iterative synthesis–refinement cycle to realistic system specifications.

To illustrate problems that arise when a subsystem model is refined without refining the component models, consider MTSs $A$ and $B$ and their composition $P$ depicted in Figure 2c. Let $P'$ be a refinement of $P$ such that $P'$ is obtained by modifying $P.S_3 \overset{a}{\rightarrow}_m P.S_2$ to required and removing $P.S_0 \overset{a}{\rightarrow}_m P.S_1$. Notably, $P'$ is an invalid model as it is not a legal composition of any of $A$'s and $B$'s refinements. $B$ cannot be further refined since it does not have maybe transitions, while $A$ has only two refinements depending on whether the maybe transition is removed or becomes required. If $A$'s transition becomes required, the composite behavior of $A$ and $B$ is an arbitrary interleaving of $a$ and $b$, while $P'$ is not possible. If such refinement conflicts are not prevented or detected in a timely manner, a significant modeling effort may be lost (e.g., either $P'$ or both $A$ and $B$ would be invalidated).

My technique will propagate an MTS refinement to the related models by constructing and utilizing a relation between the states in a subsystem MTS and individual component MTSs. Whenever a new requirement is incorporated by refining a subsystem MTS, the related component MTSs will be refined accordingly. For example, my technique will capture the relation between $A.S_0$ and two states in the subsystem model $P$ – $P.S_0$ and $P.S_3$. This relation is then used to propagate the refinement of $P.S_3 \overset{a}{\rightarrow}_m P.S_2$ from maybe to required by requiring the the corresponding $A$'s transition $A.S_0 \overset{a}{\rightarrow}_r A.S_1$ and, in turn, requiring $P.S_0 \overset{a}{\rightarrow}_r P.S_1$. By performing the refinement propagation, we avoid the previously described conflict.

## 3. EVALUATION PLAN

In my thesis evaluation, I will formally analyze the correctness of the proposed approach and examine different practical considerations related to the produced artifacts. The initial empirical assessment will be done on common requirements exemplars. Subsequently, I will apply the approach to the specification of two mid-sized applications (10,000–100,000 SLOC) developed at USC in collaboration with an external organization: (1) a sensor network application MIDAS [12], and (2) a mobile robotics application [13].

To validate the synthesis algorithm, I intend to prove its correctness and determine its worst-case complexity. To test the practical performance of the synthesis algorithm, I will measure the implementation's running times on the different available specifications and test scalability on large, randomly generated specifications. I will assess the correctness of the synthesized MTSs through manual inspection, as well as compare them with MTSs produced using existing techniques. Specifically, I will test whether the component MTSs omit desirable system-level behavior and whether they successfully expose system-level behavior that is not feasible, but present in the system-level model (dashed lines in Figure 1). I will evaluate the requirements discrepancies discovered using component MTSs across two dimensions: relevance and precision.

I will validate the usefulness of the requirements elicitation/elaboration technique with a qualitative analysis of the proposed requirements. The question I will ask is whether the new requirements are non-trivial and whether they help to explore real concerns for a system at hand. Finally, I will measure the number of positive and negative answers in the elaboration process to see how much additional effort is invested by an engineer using the technique.

## 4. CURRENT STATUS

In an early paper about my ongoing work [6], I described the high-level steps of the component-level MTS synthesis algorithm (*task 1* in Figure 1). In that paper, I also outlined promising ways of utilizing component MTSs, including requirements elaboration. I have described my planned thesis in a recent two page, initial stage Doctoral Symposium paper [5]. This proposal contains significant expansions of initially presented ideas and a more elaborate evaluation plan.

Subsequently, I have more precisely specified the synthesis algorithm (*task 1*) and implemented the algorithm in a tool that accepts positive scenarios and operational requirements as inputs. This work was reported along with the analysis of the algorithm's correctness in a conference publication [7]; the algorithm was subsequently utilized to derive MTSs for source code artifacts [8]. Most recently, I have been working on the refinement support (Section 2.3); I have devised a technique that helps to propagate atomic MTS refinements to its related models. This work has been published as a technical report [9], while a more comprehensive journal submission is in preparation.

## 5. ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Nenad Medvidovic for his insightful advice and generous support on planning, conducting, and reporting my thesis research. I would also like to thank the reviewers for their helpful comments. This work was sponsored in part by the National Science Foundation, award numbers 0720612 and 0820170.

## 6. REFERENCES

[1] D. Alarjeh et al. Learning operational requirements from goal models. In *Proc. of ICSE*, 2009.
[2] M. Dwyer et al. Patterns in property specifications for finite-state verification. In *Proc. of ICSE*, 1999.
[3] D. Fischbein and S. Uchitel. On correct and complete strong merging of partial behaviour models. In *Proc. of FSE*, 2008.
[4] D. Giannakopoulou and J. Magee. Fluent model checking for event-based systems. In *Proc. of FSE*, pages 257–266, 2003.
[5] I. Krka. Synthesizing and utilizing partial behavior models during requirements elicitation. In *Proc. of ESEC/FSE Doctoral Symposium*, 2009.
[6] I. Krka et al. From system specification to component behavioral models. In *Proc. of ICSE NIER*, 2009.
[7] I. Krka et al. Synthesizing partial component-level behavior models from system specifications. In *Proc. of ESEC/FSE*, 2009.
[8] I. Krka et al. Using dynamic execution traces and program invariants to enhance behavioral model inference. In *Proc. of ICSE NIER*, 2010.
[9] I. Krka and N. Medvidovic. Supporting refinement of partial behavior models under model composition and abstraction. *Tech. Rep. USC-CSSE-2010-510*, 2010.
[10] H. Liang et al. A comparative survey of scenario-based to state-based model synthesis approaches. In *Proc. of Work. on Scen. and State Mach.*, 2006.
[11] J. Magee and J. Kramer. *Concurrency: State Models and Java Programming.* John Wiley & Sons, 2006.
[12] S. Malek et al. Reconceptualizing a family of heterogeneous embedded systems via explicit architectural support. In *Proc. of ICSE*, 2007.
[13] S. Malek et al. An Architecture-Driven software mobility framework. *JSS*, 2010.
[14] G. Sibay et al. Existential live sequence charts revisited. In *Proc. of ICSE*, 2008.
[15] S. Uchitel and M. Chechik. Merging partial behavioural models. 2004.
[16] S. Uchitel et al. Behaviour model elaboration using partial labelled transition systems. In *Proc. of ESEC/FSE*, 2003.
[17] S. Uchitel et al. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM TOSEM*, 13(1), 2004.
[18] S. Uchitel et al. Synthesis of partial behavior models from properties and scenarios. *IEEE TSE*, 35(3), 2009.
[19] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley & Sons, 2009.