

ULRR

Towards behavior elaboration and synthesis using modes

Item Type	Meetings and Proceedings
Authors	Shokry, Hesham
Citation	FSE '10 Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of Software Engineering.;2010
Publisher	Association for Computing Machinery
Download date	2026-03-07 18:00:35
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/2067

Towards Behavior Elaboration and Synthesis Using Modes

Hesham Shokry

Lero – the Irish Software Engineering Research Centre
University of Limerick, Castletroy, Limerick, Ireland

hesham.shokry@lero.ie

ABSTRACT

Early system requirements are often captured by declarative and property-based artifacts, such as scenarios and goals. While such artifacts are intuitive and useful, they are partial and typically lack an overarching structure to allow systematic elaboration of the fragmented behaviors they denote. I aim to develop a design technique for structuring the partial specifications by partitioning the state-space based on Parnas' notions of 'modes' and 'mode-classes'. A mode is set of states, characterized by a predicate. A mode-class is a set of disjoint modes completely covering the state space. The structuring framework supports early elaboration of partial specification, and facilitates improved synthesis of integrated system behavioral prototype.

Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements/Specifications;
D.2.10 [Software Engineering]: Design – Methodologies.

General Terms

Design.

Keywords

Modes, Behavior synthesis, State-space partitioning.

1. RESEARCH PROBLEM

At early stage of development, designers often have little and vague requirements specifications. The vagueness about the system behavior space, combined with the varieties of requirements sources, results in specifications that are of partial and fragmented nature. These specifications are commonly captured via intuitive artifacts such *scenarios* [14, 15]. Because of the resulting partiality, designers will be unable to ensure behavior space coverage of the prospective system, and this leads to undesirable iteration between the development phases. Moreover, the requirements fragmentation impedes early reasoning about system properties that span the behavior space.

A general approach for overcoming these problems is to synthesize an integrated behavioral model, from those specifications, for early reasoning and elaboration. Such synthesis technique must maximize coverage of space and link those fragmented behaviors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FSE-18, November 7–11, 2010, Santa Fe, New Mexico, USA.

Copyright 2010 ACM 978-1-60558-791-2/10/11...\$10.00

Several approaches have attempted to synthesize a system behavior model from partial specifications [1-6]. A common characteristic of these approaches is that they address the *symptom* of the problem: partiality, instead of its cause: the lack of a proper overarching framework for specifying system behavior. Partiality refers to the extent to which a specification, such as a scenario, captures system behaviors. The scenario is more partial if its scope is assumed to span the whole behavior space, but yet it captures a limited part of behavior.

Defining a context (i.e. part of the behavior space), within which a scenario is specified, minimizes the partiality of this scenario with respect to that context. Structuring the behavior space into a set of distinct contexts, and specifying the scenarios within these contexts, maximizes coverage of the whole behavior space by the total set of scenarios, as a direct result of the maximized coverage of individual contexts. One way to structure the behavior space is to partition it into manageable sub-spaces parts, each of which identifies a distinct system context. These contexts should be related together so as to link the individual scenarios specified in each context, resulting in integrated specifications. I argue that the provision of a structured behavior space guides the task of specifying the requirements and improves synthesis of a system model out of those requirements.

The aim of this thesis is to propose such a framework using the concept of *mode-classes*, early introduced by Parnas [7]. The framework facilitates early elaboration of partial specifications by scoping different system contexts using modes, and organizes these contexts in mode-classes so as to enable a disciplined partitioning of the behavior space from different views. Based on this framework I also propose an improved technique for synthesis of an integrated automata-based model of the system from a set of given scenarios scoped by modes.

2. PROPOSED APPROACH

The intended overarching framework is based on state space abstraction using the notion of *modes*. This section defines the basic concepts of *modes* and then describes how to use them to scope scenarios and synthesize a system model.

Modes and Mode-Classes. Let V be a set of system variables, ranging over a set of values D and defining the system's state space $T = \langle V, D \rangle$. A *state* is a valuation of system variables, sufficient to derive the future system responses to any given input. The state is determined by the function $t : V \rightarrow D$, and the *state space* $T = \langle V, D \rangle$ is the set of observable states, defined as $v_1 : D_1, \dots, v_n : D_n$. We assume a transition system between states as

the concrete representation of behavior space, relative to which we describe more abstract representations using *modes*.

DEFINITION 1 (Mode M). Let $T=\langle V, D \rangle$ be a space and let Q be a predicate over V . A *mode* M is a subset of states $M \subset T$ such that $\exists t \in T, Q(t) \Rightarrow t \in M$, where Q is said to be *characterizing* M . ■

That is, a mode M is a subset of states that are satisfying some predicate Q . Unlike hierarchical state models, a mode itself is not a higher-level state, but it is a group of states. We can say that if two identical systems are in the same state, their future responses, to the same input, are indistinguishable. However, this is not necessarily true for two identical systems in the same mode. In the later case, the two identical systems might be in two different states that are belonging to the same mode.

To be able to represent a space T using modes, there must be a number of modes that cover the space. These modes must be *disjoint* so as to be able to use them in an abstract transition system. Such a collection of modes is referred to as a *mode-class*.

DEFINITION 2 (Mode-Class MC). For a space $T=\langle V, D \rangle$, a list of modes $MC = \langle M_1, M_2, \dots, M_m \rangle$, characterized by a corresponding list of predicates $Q_{MC} = \langle Q_1, Q_2, \dots, Q_m \rangle$, is called a *Mode-Class* MC iff each state $t \in T$ is in exactly one mode $M_i \in MC$. That is,

$$\forall t \in T, \bigoplus_{i=1..m} Q_i(t). \blacksquare$$

That is, a mode-class MC is a set of disjoint modes specified such that they cover the system state-space. A MC is a *mathematical partitioning* of the space T , induced by the Equivalence Relation given by the formula in Definition 2. There may be several MC s specified for the same system, such that each MC partitions the space from a different dimension. As a symbolic example, consider a system with two integer variables x and y that take values in $[0,9]$. The space can be partitioned by two modes characterized by the predicates $Q_a \Rightarrow (x \geq 0 \wedge x \leq 4)$ and $Q_b \Rightarrow (x \geq 5 \wedge x \leq 9)$. Another MC can partition the system where y varying over the same ranges $[0,4]$ and $[5,9]$. A third MC could be $Q_c \Rightarrow (x > y)$, $Q_d \Rightarrow (x = y)$ and $Q_e \Rightarrow (x < y)$. Every system state belongs to exactly one mode from each of those mode-classes. For e.g., a state t that satisfies the predicate Q_a must belong to exactly one of the modes in the mode-class characterized by Q_c, Q_d or Q_e .

Scenarios' Contexts Structuring. There is a synergy between mode-classes and scenarios (or partial specifications in general), and it can be illustrated in terms of the *scope of a mode* M and the *context of a scenario* SD notions. By the *scope of* M we mean the set of states that belong to M . By the *context of* SD we mean the total set of states that are formed by the *pre-* and *post-*conditions at each message in the SD . By a SD fits under the scope of M we mean that the SD 's context is a subset of M 's scope.

Since the same system's space can be partitioned from several different dimensions, designers may specify as several mode-

classes as they see fit. On the other hand, scenarios are generally perceived as several (possibly disjoint or overlapping) descriptions of the same system. This suggests that mode-classes allows to define a variety of contexts where adding a new mode-class exposes emerging contexts. This provides a fertile environment to uncover possible gaps within which scenarios can be specified. At the same time, mode-classes organize those contexts in *classes* that facilitate independent elaboration and maintenance of partial specifications.

The disjointness property of modes (in the same mode-class) enables designers to draw a transition relation between modes in a mode-class, in the same way they specify transitions between states in a state machine but at a higher level of abstraction. This relation establishes the *inter-context* transitions that relate the fragmented specifications scoped by the modes, leaving away the intra-context transitions specified in the scenarios themselves. A mode-class with a transition relation is called a *Mode-Machine*.

Synthesis from Structured Scenarios Specifications. Given a specification of the system as a collection of scoped scenarios and their scoping mode machines, together are referred to as *structured scenarios specifications*, we synthesize a standard FSM out of those specifications.

Synthesis of automata models from partial specifications involves two major phases: *translating* the individual specifications to FSM models, then *merging* these isolated models. The translation phase is typically straightforward with some proprietary techniques for loops detection. The merge phase is more challenging due to difficulties in finding a *common refinement* model that exhibits all the individual behaviors.

The basic hypothesis of our synthesis process is that *an integrated system model can be obtained by a successive refinement of the state space*. Every mode machine represents a view of the state space partitioned into a set of contexts. The scoped scenarios specified within these contexts are *partially refining* the state space (because a scenario is still likely to be partial even within the sub-space of its scoping mode). So, with several mode machines—each with its own scoped scenarios—we will have several partial refinements of the system state space.

Next, what we need to do is to merge these partial refinements such that to find a common refinement model exhibiting all their behaviors. Here comes again the power of mode-classes. Since the mode-classes are partitioning the same space, then every mode in a mode-class is completely overlaps with at least one mode (or possibly all modes) of any other mode-class. This can directly deduced from the aforementioned assertion that 'every state belongs to *exactly* one mode from each mode-class.'

Fig. 1 illustrates a coarse-grained version of the synthesis process using a symbolic example of a system with two mode-machines scoping a set of scenarios. The mode-machines are depicted as a standard transition diagram with each node represents a mode. For visual illustration, the modes in the two mode-machines are distinguished by *bright-* and *dark-*blue colors.

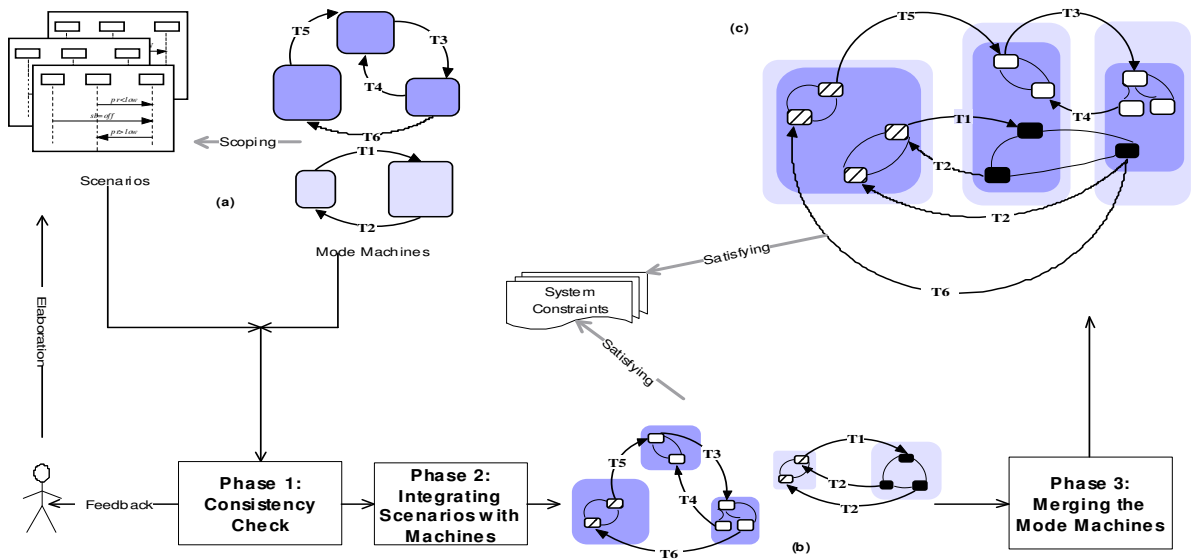


Figure 1: Behavior synthesis process. (a) The input specifications. (b) The Mode Machines with scenarios are translated and mode transitions are elaborated. (c) The final integrated system model after merging the mode-machines. (LEGEND: thin lines indicate data flows and bold-gray lines indicate data relations)

Phase 1 performs consistency check on the input (Fig. 1(a)) to ensure disjointness and coverage of mode-classes, and to ensure that a scenario does not have *pre-* or *post-*conditions violating the predicate of the scenario’s scoping mode. This phase provides feedback to the designer so as to adjust the specifications.

Phase 2 starts the synthesis by, first, translating the scenarios into *local* FSMs (depicted as small machines in Fig. 1(b) within the areas of the modes) and assigned under the corresponding modes that scope these scenarios. Then it follows by an elaboration of the mode transitions (T1 to T6 in both mode-machines). The transition elaboration step attempts to relate the individual local machines based on the fact that *a transition appearing at the mode-level is actually a transition between states belonging to the source and destination modes*. For e.g., the mode transition T2 (Fig. 1(b)) is found to be elaborated to two state transitions. This step must also respect the system constraints, so that the generated machines are satisfying them.

Phase 3, finally, merges the isolated machines into one model (Fig. 1(c)) where the overlapping modes, from different mode-classes, are detected and possibly new modes are created. For e.g., the states with shaded states, from different mode machines, have happened to co-exist under the same space overlap. Another transitions elaboration step is made to find potential transitions between states that happen to co-exist in the overlapping areas (transitions shown in dotted lines in Fig.1(c)). This phase must also respect the system constraints.

It is important to note that the mode-classes in Fig.1 are a special case where one of them (the darker one) is actually a refinement of the other (the brighter mode-class). In more general cases, a mode from one mode-class overlaps with possibly all modes from the other, thus creating new modes in the intersection areas and yields a more refined model. As a final note, the example in Fig. 1 is a special case where the dark mode-class is a *space refinement* of the brighter one. More *orthogonality* between mode-classes can happen such that the merge process results in

newly created modes [16] and, in turn, provides a finer output model.

3. RELATED WORK

A wide range of techniques have been proposed to automate the construction of behavioral models out of partial specifications. To the best of my knowledge, no prior approach addresses the structuring of contexts in partial specifications in terms of providing coverage and separation of concerns. However, the extraction of contextual information from a given specifications is often done (cf. [2, 3]). Below I summarize the state of the art work, classified by different aspects of this research area.

On modes. In the computing literature, there are two fundamentally different usages of the term ‘mode’: in Modal Logic [8] and in Hybrid Systems [9]. The former is used to express multi-valued logic. The latter usage is to characterize different behaviors of a hybrid system. The notion of mode that I use is based on the ideas in [7] and has its origins in the theory of hybrid systems. Mode-Automata [10] use modes as a discrete version of Hybrid Automata. Modecharts [11] is an RTL-based language. These approaches do not differentiate between the notions of mode and a state as in [7]. Paynter [12] uses non-disjoint modes but does not use *mode-classes* [7]. Having several mode-classes that describe the same system allows a state to belong to several modes (each mode in a different mode-class) and achieving the same purpose of non-exclusion option adopted in [12] but in a separation of concerns manner.

On Synthesis from partial specifications. A common direction for addressing partiality is to enrich the machines, independently - generated from scenarios, with ‘may be’ behaviors, using special logic, and delay decisions to a later stage of synthesis (cf. [4]). Another range of techniques use predicate abstraction (cf. [2, 3]) or AI techniques (cf. [1]) to infer behaviors from the given specifications. A common denominator of these approaches is the use of bare scenarios without a structuring framework. This

distracts the synthesis process with issues related to partiality itself, such as the assumption that a scenario must start at the initial state and the likelihood of absence of a common refinement [4], which are avoided in my approach. Finally, I define the system contexts at the outset, instead of extracting them from the given unstructured specifications.

On the use of richer forms of scenarios. Other proposals accept more expressive forms of scenario as input. Uchitel et al. [13] synthesize behavioral models from Message Sequence Charts [14] to detect implied scenarios. Whittle and Jayaraman [5] use the recently introduced Interactions Overview Diagrams IOD in UML 2.0 [15] to generate Statechart-based designs. Though specified at a higher-level, these forms of specifications express control-flow information between scenarios in a flowcharting-like structure. The higher-level feature in such specifications does not address coverage of the behavior space nor modularize scenarios. My approach is distinct by structuring the specifications in the state space rather than relating them in a flowcharting manner. I assume simple form of scenarios, Sequence Diagrams [15], as input; however I augment it with mode-classes for structuring them.

4. CURRENT STATUS AND NEXT STEPS

Current status. In an earlier technical report on my ongoing work [16], I formulated behavioral modeling framework based on the concepts of modes. I proposed two theorems: (1) a theorem to support *partial refinement* of a mode machine that scopes a set of scenarios, and (2) a theorem supporting the merge of an arbitrary number of mode machines. Using these foundations of mode-based behavior modeling and refinement, I sketched a synthesis process to generate an integrated system model out of the given structured scenarios specifications. A simplified version of this process is illustrated in Fig. 1. Finally, I developed a set of algorithms that are necessary to implement the individual phases of the synthesis process in Fig. 1.

Evaluation Plan. (1) Using case studies to assess the effectiveness of specifications structuring for improving the elaboration of requirements, with comparisons to other approaches (cf. [2, 13]). The case studies shall assess: *how maximum* could the context-focused specifications cover the behavior space; *how intuitive* is specifying mode-classes. (2) Although the formal foundation of the proposed synthesis process provides an initial validation of its steps, I plan to apply the process to a real-world application, using a tooling support, to assess: *how refined* will be the generated models and *how productive* is to regenerate them.

Future work. The proposed synthesis process has a big potential to be automated. In the first phase of the process, the *consistency check* phase can be mechanized such that the disjointness and coverage properties can be automatically checked using theorem provers, as already demonstrated in [17]. Consistency between modes and the *pre-* and *post-*conditions, in the corresponding scenarios, can be checked using of SMT solvers. In the second phase, the integration of scenarios with mode machines can also be automated where scenarios are translation via existing algorithms (cf. [6]) that can be reused directly. Finally, in the third phase, SMT solvers can be used to detect overlaps between modes from different mode-classes. Identical states that are repeated in different modes can be detected and merged using standard automata-theoretic as applied in [6]. A proper tool support is also necessary during transitions elaboration steps in order to resolve nondeterminism arising as side effect of abstraction.

Besides implementing a tool to automate this process, I plan to investigate techniques for generating machine models from scenarios with *incomplete* state vectors. This is motivated by the fact that designers can not always determine a value for every variable in the *pre-* and *post-*conditions due to the immature vision about the system. In such case, scenarios themselves will be translated to mode machines resulting in a hierarchy of modes.

ACKNOWLEDGEMENT

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1. A special thank to Dave Parnas for taking the time to explain the concept of mode-classes.

REFERENCES

- [1] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Learning operational requirements from goal models. In *Proc. of ICSE*, 2009.
- [2] E. Mäkinen and T. Systä. MAS - an interactive synthesizer to support behavioral modelling in UML. In *Proc. of ICSE*, 2001.
- [3] J. Sun and J. S. Dong. Design Synthesis from Interaction and State-Based Specifications. *IEEE TSE*, 32(6), 2006.
- [4] S. Uchitel, G. Brunet, and M. Chechik. Synthesis of Partial Behavior Models from Properties and Scenarios. *IEEE TSE*, 35(3), 2009.
- [5] J. Whittle and P. K. Jayaraman. Synthesizing hierarchical state machines from expressive scenario descriptions. *ACM TOSEM*, 19(3), 2010.
- [6] J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *Proc. of ICSE*, 2000.
- [7] T. Alspaugh, S. Faulk, K. Britton, R. Parker, D. Parnas, and J. Shore. Software requirements for the A-7E aircraft. Naval Research Lab, NRL-9194,1992.
- [8] S. Kripke. Semantic considerations on modal logic. *Acta philosophica fennica*, 1963.
- [9] E. A. Lee. Finite State Machines and Modal Models in Ptolemy II," Report UCB/EECS-2009-151, EECS Dept., University of California, Berkeley, Nov. 2009.
- [10] F. Maraninchi and Y. Remond. Mode-automata: a new domain-specific construct for the development of safe critical systems. *Sci. Comput. Program.*, 46(3), 2003.
- [11] F. Jahanian and A. K. Mok. Modechart: A Specification Language for Real-Time Systems. *IEEE TSE*, 20(12), 1994.
- [12] S. Paynter, "Real-time mode-machines. In *Formal Techniques in RealTime and Fault-Tolerant Systems*, 1996.
- [13] S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM TOSEM*,13(1), 2004.
- [14] ITU. Message sequence chart. 2000.
- [15] OMG. UML Specification 2.0. 2003
- [16] H. Shokry and M. Hinchey. Modal Software-Behavior Modeling and Synthesis from Partial Specifications. Lero-TR-2010-01, University of Limerick, Limerick. April 2010.
- [17] J. M. Rushby and M. K. Srivas. Using PVS to Prove Some Theorems Of David Parnas. In *Proceedings of the 6th International Workshop on Higher Order Logic Theorem Proving and its Applications*: Springer-Verlag, 1994.