

A Comparative Survey of Scenario-based to State-based Model Synthesis Approaches *

Hongzhi Liang
School of Computing
Queen's University
Kingston, Ontario, Canada
liang@cs.queensu.ca

Juergen Dingel
School of Computing
Queen's University
Kingston, Ontario, Canada
dingel@cs.queensu.ca

Zinovy Diskin
School of Computing
Queen's University
Kingston, Ontario, Canada
zdiskin@cs.queensu.ca

ABSTRACT

Model Driven Development and Use Case Driven Development methodologies have inspired the proposal of a variety of software engineering approaches that synthesize state-based models from scenario-based models. However, little work has been done to comprehensively compare these different synthesis approaches. In this paper, we define a set of comparison criteria, and survey 21 different synthesis approaches presented in the literature based on the criteria. The differences and similarities are highlighted in the comparison results. We then discuss the challenges that current approaches may face and provide suggestions for future work for state-based model syntheses.

Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements/Specifications — *Languages, Tools*; D.2.2 [Software Engineering]: Design Tools and Techniques — *Computer-aided software engineering (CASE), Petri nets, State diagrams*.

General Terms

Algorithms, Design, Theory.

Keywords

Model, Scenario, State machines, Synthesis, SDL, MSC, UML, Petri Nets.

1. INTRODUCTION

There are many ways of modeling the dynamic behavior of reactive systems. The most popular approaches are scenario-based and state-based modeling. A scenario-based

*This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), Communications and Information Technology Ontario (CITO), and IBM Software Lab in Ottawa, Canada

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, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SCESM'06, May 27, 2006, Shanghai, China.

Copyright 2006 ACM 1-59593-085-X/06/0005 ...\$5.00.

model represents the “inter-object” behaviors by describing interactions between multiple objects. In contrast, a state-based model is often used to represent the entire behavior of a reactive system, for instance either by a global state machine or by a set of communicating state machines in which each state machine describes the complete “intra-object” behavior of an object [12]. Dozens of notations can be used for the description of scenario-based or state-based models. The ITU standard of Message Sequence Charts (MSCs) [14], UML Sequence Diagrams (SDs), Communication Diagrams (CDs) and Interaction Overview Diagrams (IODs) [26], and Live Sequence Charts (LSCs) [5] are some of the most popular scenario notations. For state-based modeling, Statecharts [9], Petri Nets [27], and the behavior section of the ITU standard of Specification and Description Language (SDL) [13] are some of the most popular notations.

Scenario-based and state-based modeling provide two different views of reactive systems. Scenarios are well known to help requirements engineers elicit functional requirements, as well as comprehend and validate requirements. Therefore, requirements engineers may find that scenario-based models are more natural to use and easier to communicate to other stakeholders such as customers. On the other hand, code can be automatically generated from state-based models and therefore software designers may find state-based models are closer to design and implementation. The two views are not independent but strongly connected. In fact, scenarios (or use cases) can be used to drive the whole life cycle of software development processes [15]. In other words, scenarios are not only for the requirement phase, but also for the design and implementation phases.

The strong connection indicates that transformation bridging the gap between these two views should be possible. Such a transformation would significantly increase the effectiveness of the overall software development process. Figure 1 shows the different types of scenario-based to state-based model transformation. In the upper half of the diagram, scenario-based models are sub-categorized into either basic scenarios (BS) without using any composition mechanisms, or global scenarios (GS) obtained through the composition of BSs. In the lower half, we have two categories of state-based models: object state machines (OSM) and global state machines (GSM) which are composed of OSMs. The arrows in the diagram show four direct transformation paths from scenario-based models to state-based models via synthesis (i.e., BS→OSM, BS→GSM, GS→OSM, and GS→GSM), and four indirect transformation paths via synthesis and

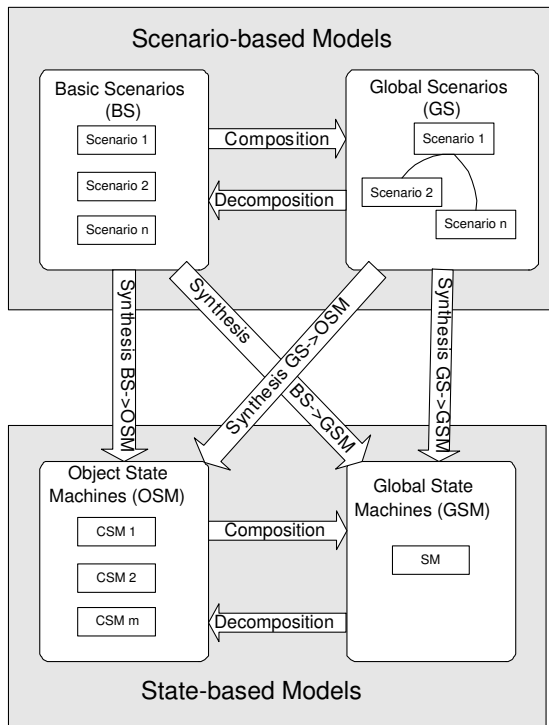


Figure 1: Different types of transformation between scenario-based models and state-based models.

(de)composition (i.e., $BS \rightarrow OSM \rightarrow GSM$, $BS \rightarrow GSM \rightarrow OSM$, $GS \rightarrow OSM \rightarrow GSM$, and $GS \rightarrow GSM \rightarrow OSM$). Synthesis of scenarios from state-based models is also an interesting topic, such as, for example, the generation of test scenarios from state-based models. However, we focus only on approaches that synthesize state-based models, because by far the most approaches proposed in the literature discuss this synthesis direction.

Although many different approaches have been proposed in the last decade or so, little work has been done to comprehensively compare any of the approaches. An overview and comparison of 26 scenario-based to state-based model synthesis approaches is given in [1]. However, the comparison does not use an equally comprehensive set of classification criteria. In order to systematically compare approaches, we define two sets of comparison criteria. One set of criteria is for assessing approaches from a user’s perspective. The other set of criteria compare the approaches from a more technical perspective. We then present a comparative survey of 21 approaches found in the literature based on these two sets of criteria. One of our goals is to identify the differences and similarities among approaches and highlight them in the comparison results. The other goal is to explore some of the challenges that current approaches may face and provide suggestions for future work. Finally, we hope to provide a comprehensive overview and an up-to-date bibliography in the field of scenario-based to state-based model syntheses for researchers or readers who are interested in this topic.

Our paper is organized as follows: we proceed by introducing our proposed comparison criteria in Section 2. Section 3 provides a survey of the state-of-the-art in scenario-based to state-based model synthesis approaches and shows the com-

parison results, highlighting the similarities and differences. The challenges current approaches may face and suggestions for future work are discussed in Section 4. Finally, in Section 5 we present our conclusion.

2. COMPARISON CRITERIA

Different approaches may address the synthesis problem from quite different viewpoints. Nevertheless, as one activity of software engineering processes, we can evaluate all of the synthesis approaches from different stakeholders’ perspectives. We have identified two groups of stakeholders with special interest in this topic: users (e.g., requirement engineers, software designers, and software developers) and synthesis approach designers. For users, who directly interact with synthesis approaches, the primary interest is whether or not the synthesis approaches provide the desired functionalities. Designers, on the other hand, care about technical details as well. Therefore, to categorize and compare different approaches, we collected two sets of comparison criteria, one set from the user’s perspective, and another set from a more technical perspective.

2.1 Criteria Relevant From a User’s Perspective

Intended use: Approaches which synthesize state-based models from scenario-based models seem to have two different uses: analysis and code generation. More precisely, the generated state machines can be analyzed, for instance, for correctness using model checking, or they can be used as the basis for the generation of executable code. In general, it appears that GSMs representing the global system behaviors are less useful for code generation, because the resulting code would be too monolithic. However, GSMs are extremely useful for checking that the global system behaviors meet certain desirable properties. Alternatively, OSMs are suitable for both code generation and object-level analysis. If an approach allows for the composition of the OSMs, system-wide analyses are also possible. Therefore, for each approach, we classify it as intended for analysis only, or for both analysis and code generation.

Source notation: While all scenario notations are similar in the sense that they describe interactions between objects, the syntax and semantics of different notations can differ in subtle ways. Such differences could, for instance, influence the users’ ability to describe scenarios with different levels of expressiveness. Moreover, the choice of source notation may significantly affect the synthesis algorithms.

Support of composition mechanism: By using composition mechanisms as provided, for instance in High Level MSCs (HMSCs), some scenario notations have the ability to express behavior of complex systems more comprehensively. Synthesis approaches may support composition mechanisms at various scales or not support them at all.

Support of parallelism: Some notations allow different scenarios or different portions of a scenario to be executed in parallel. By supporting parallelism, scenario notations can describe reactive systems more realistically. For more advanced scenario notations, parallelism is either implicitly supported by means of the underlying semantics or explicitly supported by means of parallel composition constructs. However, a synthesis approach may not support parallel scenarios because of the increase in computational complexity that parallelism typically entails.

	Approach	Source scenario-based			Target state-based		Synthesis path	Degree of automation	Tool support	
		Notation	Support for Composition	Support for parallelism	Notation	Model type				
Intended use	Analysis and code generation	Leue et al. '98 [20] ¹	MSC	Yes (HMSC) ²	No	ROOM	Object	GS→OSM	Full	Yes (MESA) ³
		Mansurov and Zhukov '99 [23]	MSC	Yes (HMSC)	No	SDL	Object	GS→OSM	Full	Yes (MOST)
		Krüger et al. '99 [19]	MSC	No	No	Statechart	Object	BS→OSM	Full	No
		Bordeleau et al. '00 [4]	MSC	Yes (UCM)	N/A ⁴	Automaton	Object	GS→OSM	Full	No
		Uchitel et al. '01 [31]	MSC	Yes (HMSC)	No	FSP + LTS	Global + Object	GS→OSM →GSM	Full	Yes (Prototype)
		Whittle and Schumann '00 [35] ⁵	SD	No	No	Statechart	Object	BS→OSM	Full	Yes (Prototype)
		Mäkinen and Systä '01 [22]	SD	No	No	Statechart	Object	BS→OSM	Semi	Yes (TED)
		Maier and Zündorf '03 [21]	SD	No	No	Statechart	Object	BS→OSM	Full	Yes (Fujaba)
		Ziadi et al. '04 [36]	SD	Yes (Operators)	No	Statechart	Object	GS→OSM	Full	Yes (Prototype)
		Nicolás and Martínez '05 [25]	SD+CSD	Yes (UCM)	Yes	SDL	Object	GS→OSM	Full	No
		Harel and Kugler '01 [10]	LSC	No	Yes	Automaton+ Statechart	Global + Object	BS→GSM →OSM	Full	Yes (Prototype)
		Harel et al. '05 [11]	LSC	No	Yes	Statechart	Object	BS→OSM	Semi	Yes (Prototype)
		Bontemps et al. '05 [3]	LSC	No	Yes	I/O Automaton	Object	BS→OSM	Full	Yes (Prototype)
		Khriss et al. '01 [16]	CD	Yes(Sequence numbers)	Yes	Statechart	Object	BS→OSM	Full	Yes (Prototype)
		Diethelm et al. '02 [7]	CD	Yes (AD)	No	Storychart	Object	BS→OSM	Semi	No
	Koskimies and Mäkinen '94 [18]	Trace diagrams	No	No	State machine	Object	BS→OSM	Full	Yes (SCED)	
	Dano et al. '97 [6]	Text	Yes (PN)	Yes	Automaton	Object	BS→OSM	Full	Yes (Prototype)	
	Analysis	Elkoutbi and Keller '00 [8]	SD	Yes (PN)	Yes	PN	Global	GS→GSM	Semi	Yes (Prototype)
		Kloul and Küster-Filipe '05 [17]	SD	Yes (IOD)	Yes	PEPA net	Global	GS→GSM	Full	No
		Sgroi et al. '04 [28]	MSC	Yes (MSN)	Yes	PN	Global	GS→GSM	Full	No
Somé et al. '95 [29]		Text	Yes (Operators)	No	Timed Automaton	Global	GS→GSM	Full	No	

Abbreviations:

AD: Activity Diagrams HMSC: High-Level MSCs MSC: Message Sequence Charts SDL: Specification and Description Language
CD: Collaboration Diagrams IOD: Interaction Overview Diagrams MSN: Message Sequence Nets UCM: Use Case Maps
CSD: Composite Structure Diagrams LSC: Live Sequence Charts PN: Petri Nets
FSP: Finite Sequential Processes LTS: Labeled Transition Systems SD: Sequence Diagrams

Notes:

1. Approach's name followed by proposed year and bibliography number.
2. Name of the composition mechanism.
3. Name of the tool. "Prototype" indicates no formal name.
4. Information is not available, as no detailed synthesis algorithm was given.
5. In [34], the algorithm of [35] has been modified to support composition, parallelism and GS -> OSM synthesis.

Table 1: Comparison of synthesis approaches from user's perspective.

		Inter-scenario relationships				
		Approach	Consistency check?	Completeness check?	State space reduction?	
Inter-scenario relationships	Inferred by	Semantics	Harel and Kugler [10]	Yes	No ¹	Yes (++)
			Harel et al. [11] ²	Yes	No	No
			Bontemps et al. [3] ²	Yes	Yes (Missing scenario)	Yes (++)
		Events	Koskimies and Mäkinen [18]	No	No ³	Yes (++)
			Diethelm et al. [7]	No	No	Yes (++)
			Maier and Zündorf [21]	No	No ³	Yes (++)
	Mäkinen and Systä [22]		No	Yes (Implied scenario)	Yes (+)	
	Conditions	Somé et al. [29]	No	No	Yes (+)	
		Dano et al. [6]	No	No	Yes (+)	
		Krüger et al. [19]	No	No	Yes (+)	
		Whittle and Schumann [35] ⁴	Yes	No	Yes (+)	
		Khriss et al. [16]	Yes	Yes (Implied scenario)	Yes (+)	
	Composition mechanisms	Leue et al. [20]	No	No ³	No	
		Mansurov and Zhukov [23]	No	No ³	No	
		Bordeleau et al. [4]	No	No	No	
		Sgroi et al. [28]	No	Yes (Implied scenario)	No	
		Ziadi et al. [36]	No	No ³	No	
		Kloul and Küster-Filipe [17]	No	No	No	
Nicolás and Martínez [25]		No	No	No		
Hybrid		Elkoutbi and Keller [8]	No	Yes (Implied scenario)	Yes (+)	
	Uchitel et al. [31]	No	Yes ⁵ (Implied scenario)	Yes (+)		

Notes:

1. Completeness check is not needed, as the authors prove that the generated models contain the exact same behaviors as the original scenario models.
2. The approaches are not complete, i.e., they may fail to produce correct models although such models exist.
3. Identify the implied scenarios problem without giving a concrete solution.
4. In [34], a hybrid approach based on the algorithm of [35] is given.
5. Discussed in [32].

Table 2: Comparison of synthesis approaches from a technical perspective.

Target notation: The choice of a target state-based notation is mainly influenced by the intended use. Also, the previous experience of the designer of the synthesis approach may contribute to the choice of notation.

Model type: This criterion is closely related to the intended use. An approach may concentrate on deriving a set of OSMs, or try to generate one single GSM for the whole system. We call the former *object* model and the latter *global* model.

Synthesis path: Although the synthesis of state-based models is the ultimate goal, an approach may pick specific paths, i.e., one of the direct or indirect paths shown in Figure 1.

Degree of automation: The generation of state-based models can either be *semi*-automatic or *fully*-automatic.

Tool support: We list whether a synthesis approach is supported by a tool.

2.2 Criteria Relevant From a Technical Perspective

Inter-scenario relationships: Without concrete knowledge of the inter-scenario relationships, for instance the temporal and compositional relationships, it is impossible to produce efficient state-based models, or impossible to produce models at all in extreme cases. Therefore, how to identify inter-scenario relationships could be the most crucial choice for designers of synthesis algorithms. We have identified five different ways to identify the relationships used by the approaches. The designers can implicitly infer the relationships from the scenarios by using *events* or from the *semantics* of scenario notations. Additionally, the designers can explicitly define the relationships among scenarios by *composition mechanisms*, *conditions*, or a combination (*hybrid*) of both composition mechanisms and conditions.

Consistency check: The requirements (i.e., the scenario-based models) can be semantically inconsistent. Approaches may allow checking the consistency of scenario-based models before or during the synthesis processes.

Completeness check: The behaviors inferred from the synthesized state-based models may not be equal to the behaviors specified by the scenario-based models, i.e., a synthesized model may contain more behaviors or fewer behaviors than the original model. Completeness checks on *implied scenarios* (extra behaviors) or *missing scenarios* (fewer behaviors) may be provided by the approaches.

State space reduction: Depending on how the inter-scenario relationships are identified, an approach may merge states and thus reduce the state space to various degrees. In case a reduction is offered, we distinguish two degrees of reduction. We use “++” to indicate that the authors have proved or shown that the state space is minimal, i.e., as small as possible. The case where reduction is performed, but the resulting state space is not provably minimal, is shown by “+”.

3. SURVEY AND COMPARISON RESULTS

We briefly describe a total of 21 approaches and highlight the key ideas. The approaches we surveyed are divided into five categories according to the “inter-scenario relationships” criterion: semantic-based approaches, event-based approaches, condition-based approaches, composition-based approaches, and hybrid approaches.

3.1 Survey

Semantic-based approaches: Even without any support for composition, different LSCs can be executed in parallel or sequentially. Overlap of scenarios, i.e., sharing of common states, is inferred from the semantics of LSCs. An LSC can be represented by an automaton, such that the cuts (a cut is a set of locations that consists of one location for each object) of the LSC are interpreted as states and transitions connect two cuts whenever the source cut can legally advance to the target cut via a required event. When integrating automata generated from a collection of LSCs, common states (i.e., identical cuts) are merged. Harel and Kugler’s approach [10], an improved approach based on the technique of “smart play-out” given by Harel et al. [11], and the Bontemps et al. “lightweight” approach [3] are all based on the semantics of LSCs.

Event-based approaches: Similar to semantic-based approaches, overlapping relationships can be inferred from a set of scenarios by identifying common states. However, the main difference to semantic-based approaches is the events of incoming messages in scenarios are interpreted as transitions with the events as triggers, and the events of outgoing messages are interpreted as states with the events as do-actions. Koskimies and Mäkinen [18] proposed an approach that synthesizes OSMs. When merging state machines, states and transitions are reused as long as possible. Similar approaches include the Maier and Zündorf approach [21], and the Diethelm et al. approach [7]. In Mäkinen and Systä’s Minimally Adequate Synthesizer (MAS) algorithm [22], OSMs are interactively synthesized by following the language inference algorithm of Biermann and Krishnaswamy [2].

Condition-based approaches: Rather than infer the relationships, conditions can be added to the scenarios to help identify the relationships. Conditions represent system states explicitly. Identical conditions in different scenarios are identified as common system states. The Somé et al. approach [29], the Dano et al. approach [6], the Krüger et al. approach [19], the Whittle and Schumann approach [34, 35], and the Khriiss et al. approach [16] all belong to this category.

Composition-based approaches: The inter-scenario relationships are explicitly defined by composition mechanisms. More precisely, related scenarios are connected by composition operators. Approaches in this category usually follow a two-step algorithm. First, composition operators and associated operands (basic scenarios) are translated to high-level states and transitions. Then, high-level states are replaced by sub-states and transitions are extracted from associated scenarios. The following approaches belong to this category: Leue et al. [20], Mansurov and Zhukov [23], Bordeleau et al. [4], Ziadi et al. [36], Sgroi et al. [28], Kloul and Küster-Filipe [17], and Nicolás and Martínez [25].

Hybrid approaches: Inter-scenario relationships are explicitly defined by both composition mechanisms and conditions. Like composition-based approaches, high-level states are replaced by sub-states and transitions are extracted from associated scenarios. Moreover, sub-states and transitions can be further merged based on the conditions. Two approaches are included: Elkoutbi and Keller [8], Uchitel et al. [31].

An overview of the selected approaches is given in Table 1 and Table 2 from a user’s perspective and from a more tech-

nical perspective, respectively. Due to space restrictions, we can not describe all approaches in detail. A survey containing comprehensive description of each approach will appear as a technical report at the Queen’s University at Kingston.

3.2 Comparison results

Although the tables are informative by themselves, we are more interested in the relationships between different approaches. By carefully analyzing the data collected in Table 1 and 2, we identify the following issues.

Lack of support for parallelism: Most of the approaches enjoy the expressiveness of advanced scenario notations, for instance MSCs, SDs, and LSCs. However, from the column “Support for parallelism” in Table 1, we observe that more than half of the approaches do not support parallelism. The reason behind this may be related to the computational complexity typically introduced by the support of parallelism. However, without parallelism, e.g., parallel composition, the ability of using scenarios to describe a system’s behaviour is limited. One possible solution is using Statechart’s orthogonal states to model parallel behaviors as demonstrated in [16].

Detection of implied scenarios: As described in [33], implied scenarios can be detected and labeled as either “positive” or “negative” scenarios. For positive scenarios, they indicate under-specification of the requirement models. In this case, requirement models (e.g., scenario-based models) should be refined to include these positive scenarios. In contrast, negative scenarios indicate inconsistency. Such negative scenarios should be either removed from the requirement models or explicitly added as negative scenarios to the requirement models. Several approaches identified the implied scenario problem when it exists. A few approaches even suggested solutions to either avoid implied scenarios [16, 28], or to detect implied scenarios [22, 32]. Since detecting implied scenarios allows for the discovery of incomplete or inconsistent requirements, we believe that rather than avoid implied scenarios, approaches should provide mechanisms to detect them, for instance the implied scenario detection algorithm proposed by Muccini in [24].

Optional consistency check: Most of the approaches synthesize the state-based models without checking consistency. Nevertheless, five of the approaches offer consistency checks. The checking is feasible only when the meaning of “system states” is precisely defined. This definition may be already contained in the semantics, such as for LSCs used in [3, 9, 11], or may be achieved through additional semantic constructs, such as the conditions used in [16, 35]. We consider consistency checks as one of the most important activities of requirement engineering and believe it should be carried out before the state-based models are synthesized. Therefore, we feel that additional consistency checking during synthesis should be optional rather than mandatory.

Real performance: Typically, the performance of an algorithm is described by providing its asymptotic complexity in the “Big-O” notation. Unfortunately, the asymptotic complexity is provided for only two of the approaches’ synthesis algorithms [9, 16]. Consequently, the discussion of performance is rather informal. The main factor that affects the performance is definitely how the inter-scenario relationships are identified by the approaches. The way how to identify relationships will immediately determine how the states in general and common states in particular are defined, and

will therefore determine the size of the state space. Even though algorithms such as [18], and the identifying states portion of algorithm in [16] have exponential runtime, we believe most of the synthesis algorithms have polynomial runtime with respect to the size of state space. Moreover, there seems to be an obvious tradeoff: the more reduced the state space is, the more complex the algorithm will be and vice versa.

Whether or not additional consistency or completeness checks are offered also influences the overall performance of an approach because additional checks will increase an approach's complexity. For example, although the synthesis process of [11] has polynomial runtime, the runtime needed to check consistency among scenarios is actually exponential.

Finally, we have to remember that performance measurement does not take human effort into consideration, such as the time needed to identify conditions, or the time needed to identify high-level relationships among scenarios by requirement engineers, or human input for an semi-automatic approach. From the approaches, it is not too difficult to infer a correlation between the amount of human effort required and the performance. In other words, the less (or more) human effort involved in an approach, the more (or less) complex the approach will be. For example, because the work needed to identify common states is handled by requirement engineers, composition-based approaches are usually more efficient than event-based approaches. In the last few years, approaches tend to prefer using composition mechanisms to explicitly define the inter-scenario relationships. Since it is natural to group related scenarios when eliciting requirements, such trend may indicate that using composition leads to an attractive balance between the human effort required and the performance of the approach.

4. FUTURE WORK

From the previous section, we identified a few issues and questions for the research community to address and possible topics for future work.

1. It would be interesting to determine to what extent any differences between the approaches are syntactical (i.e., due to difference in notations) or semantic. For example, different composition mechanisms have been used by the approaches. On the one hand, some of the mechanisms essentially have the same expressiveness, for instance HMSCs and IODs. On the other hand, some composition mechanisms may indeed have richer expressiveness, for instance UCMs. Such mechanisms allow for more flexible composition of scenarios as shown in [25]. However, it is not clear to what extent such increased expressiveness impacts the approaches' synthesis algorithms. Therefore, we would like to continue our work by comparing the expressiveness of different compositional mechanisms and analyzing their impact on synthesis algorithms.
2. Since the discussion on performance measurements is relatively informal, we would like to determine the asymptotic complexity of algorithms underlying the approaches by conducting empirical experiments using real case studies. Such complexity studies should further facilitate comparison of different approaches.

3. We would like to determine to what extent the results of different synthesis approaches are comparable. In other words, since different approaches may use very different synthesis paths and methodologies, it is unclear whether they all give comparable results. For instance, whether a GSM resulting from a direct synthesis contains the same behaviors as the GSM synthesized via an indirect path is unclear.
4. We would like to investigate the connection between model synthesis and model merging [30] involving different types of models, e.g., merging a scenario-based model with an existing state-based model with the help of model synthesis.

5. CONCLUSION

Although they model dynamic behavior of reactive systems from two different viewpoints, scenario-based and state-based models are not independent but strongly connected. This strong connection indicates that transformation between these two views is possible. In recent years, algorithms that generate state-based models from scenario-based models have been proposed in the literature. However, little work has been done to comprehensively compare different approaches. To address this problem, we provided a survey of total 21 different synthesis approaches. The differences and similarities of the approaches are identified using our two sets of comparison criteria and summed up in two tables. Finally, we analyzed the findings and summarized some of the challenges that current approaches may face and provide suggestions for future work.

6. REFERENCES

- [1] D. Amyot and A. Eberlein. An evaluation of scenario notations and construction approaches for telecommunication systems development. *Telecommunication Systems*, 24(1):61–94, 2003.
- [2] A. W. Biermann and R. Krishnaswamy. Constructing programs from example computations. *IEEE Transactions on Software Engineering*, 2(3):141–153, 1976.
- [3] Y. Bontemps and P. Heymans. As fast as sound (lightweight formal scenario synthesis and verification). In *3rd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools (SCESM '04)*, Edinburgh, UK, 2004.
- [4] F. Bordeleau, J.-P. Corriveau, and B. Selic. A scenario-based approach to hierarchical state machine design. In *3rd IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC '00)*, Newport Beach, California, USA, March 2000.
- [5] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. In *Formal Methods in System Design 19, 1*, pages 45–80, 2001.
- [6] B. Dano, H. Briand, and F. Barbier. An approach based on the concept of use case to produce dynamic object-oriented specifications. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE '97)*, pages 54–64, 1997.

- [7] I. Diethelm, L. Geiger, T. Maier, and A. Zündorf. Turning collaboration diagram strips into storycharts. In *Workshop on Scenarios and State Machines: Models, Algorithms, and Tools (SCESM '02)*, Orlando, Florida, USA, 2002.
- [8] M. Elkoutbi and R. K. Keller. User Interface Prototyping Based on UML Scenarios and High-Level Petri Nets. In *21st International Conference on Application and Theory of Petri Nets (ICATPN '00)*, pages 166–186, 2000.
- [9] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [10] D. Harel and H. Kugler. Synthesizing State-Based Object Systems from LSC Specifications. In *5th International Conference on Implementation and Application of Automata (CIAA '00)*, pages 1–33, London, UK, 2001. Springer-Verlag.
- [11] D. Harel, H. Kugler, and A. Pnueli. Synthesis revisited: Generating statechart models from scenario-based requirements. *Formal Methods in Software and System Modeling, Lecture Notes in Computer Science*, 3393:309–324, 2005.
- [12] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
- [13] ITU. ITU-T Recommendation Z.100: Specification and Description Language (SDL), 2000.
- [14] ITU-TS. Recommendation Z.120: Message Sequence Chart (MSC), 2000.
- [15] I. Jacobson and P. Ng. *Aspect-Oriented Software Development with Use Cases*. Addison Wesley Professional, 1st edition, Dec 2004.
- [16] I. Khriess, M. Elkoutbi, and R. K. Keller. Automatic synthesis of behavioral object specifications from scenarios. *Transactions of Society for Design and Process Science (SDPS)*, 5(3):53–77, September 2001.
- [17] L. Kloul and J. Küster-Filipe. From Interaction Overview Diagrams to PEPA nets. In *4th Workshop on Process Algebras and Timed Activities (PASTA '05)*, Edinburgh, 2005.
- [18] K. Koskimies and E. Mäkinen. Automatic synthesis of state machines from trace diagrams. *Softw. Pract. Exper.*, 24(7):643–658, 1994.
- [19] I. Krüger, R. Grosu, P. Scholz, and M. Broy. From MSCs to statecharts. In *IFIP WG10.3/WG10.5 International Workshop on Distributed and Parallel Embedded Systems (DIPES '98)*, pages 61–71, Norwell, MA, USA, 1999. Kluwer Academic Publishers.
- [20] S. Leue, L. Mehrmann, and M. Rezai. Synthesizing room models from message sequence chart specifications. Technical report, Dept. of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada, 1998.
- [21] T. Maier and A. Zündorf. The Fujaba Statechart Synthesis Approach. In *Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM '03)*, Portland, Oregon, USA, May 2003.
- [22] E. Mäkinen and T. Systä. MAS: an interactive synthesizer to support behavioral modelling in UML. In *23rd International Conference on Software Engineering (ICSE '01)*, pages 15–24, Washington, DC, USA, 2001. IEEE Computer Society.
- [23] N. Mansurov and D. Zhukov. Automatic synthesis of SDL models in use case methodology. In *Ninth SDL Forum (SDL'99)*, Montréal, Canada, 1999.
- [24] H. Muccini. Detecting implied scenarios analyzing non-local branching choices. In *Fundamental Approaches to Software Engineering*, pages 372–386, 2003.
- [25] H. Nicolás and C. Martínez. Synthesizing State-Machine Behaviour from UML Collaborations and Use Case Maps. In *SDL Forum*, pages 339–359, 2005.
- [26] OMG. Unified Modeling Language. Online, <http://www.omg.org>, 2005.
- [27] W. Reisig. *Petri Nets, An Introduction*. Springer Verlag, Berlin, 1985.
- [28] M. Sgroi, A. Kondratyev, Y. Watanabe, L. Lavagno, and A. Sangiovanni-Vincentelli. Synthesis of Petri Nets from Message Sequence Charts Specifications for Protocol Design. In *Design, Analysis and Simulation of Distributed Systems Symposium (DASD '04)*, Washington DC, USA, April 2004.
- [29] S. Somé, R. Dssouli, and J. Vaucher. From scenarios to timed automata: Building specifications from users requirements. In *Second Asia Pacific Software Engineering Conference (APSEC '95)*, page 48, Washington, DC, USA, 1995. IEEE Computer Society.
- [30] S. Uchitel and M. Chechik. Merging partial behavioural models. In *SIGSOFT FSE*, pages 43–52, 2004.
- [31] S. Uchitel and J. Kramer. A workbench for synthesising behaviour models from scenarios. In *23rd IEEE International Conference on Software Engineering (ICSE '01)*, May 2001.
- [32] S. Uchitel, J. Kramer, and J. Magee. Detecting implied scenarios in message sequence chart specifications. In *8th European Software Engineering Conference/9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-9)*, pages 74–82, New York, NY, USA, 2001. ACM Press.
- [33] S. Uchitel, J. Kramer, and J. Magee. Negative scenarios for implied scenario elicitation. *SIGSOFT Softw. Eng. Notes*, 27(6):109–118, 2002.
- [34] J. Whittle, R. Kwan, and J. Saboo. From scenarios to code: An air traffic control case study. *Software and System Modeling*, 4(1):71–93, 2005.
- [35] J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *22nd International Conference on Software Engineering (ICSE '00)*, pages 314–323, New York, NY, USA, 2000. ACM Press.
- [36] T. Ziadi, L. Hélouët, and J. Jézéquel. Revisiting statechart synthesis with an algebraic approach. In *26th International Conference on Software Engineering (ICSE '04)*, pages 242–251, Washington, DC, USA, 2004. IEEE Computer Society.