2010 34th Annual IEEE Computer Software and Applications Conference

Transformation Rules for Synthesis of UML Activity Diagram from Scenario-based Specification

Sungwon Kang, Hyunho Kim*, Jongmoon Baik, Hojin Choi, Changsup Keum
KAIST, Daejon, Korea
{sungwon.kang, jbaik, hjchoi, cskeum}@kaist.ac.kr
* Software Center, LG Electronics, Seoul, Korea
1h2kim@lge.com

***Abstract*—Although synthesis was considered an important and challenging approach to construction of a program or a program model in software development, most of research on synthesis has been devoted to the construction of state machine models or variations of them. Recently, as process modeling through languages like UML Activity Diagram and BPMN appears as a new paradigm of software development, the ability to synthesize models in such languages from requirements would tremendously increase the scope of automatic software development. This paper presents transformation rules for synthesis of UML Activity Diagrams from scenario-based specifications modeled as UML Sequence Diagrams. To that end, we first identify various control flow patterns of Sequence Diagrams and define rules for mapping them to corresponding parts of Activity Diagram. In order to make precise such mapping labeling rules are introduced for the patterns. Also we provide a synthesis algorithm for construction of a UML Activity Diagram from scenarios.**

Keywords-component; Transformation, Synthesis, UML, Activity Diagram, Scenario-based Specification, Sequence Diagram, BPMN

I. INTRODUCTION

The term "synthesis" has been traditionally used to represent automatic construction of a program or a behavioral model from formal requirements. This can be seen in the remark by Manna et al. [12] that it "takes *a relational description and tries to produce a program that is guaranteed to satisfy the relationship, and therefore does not require debugging or verification*" and the remark by Harel et al. [5] that it is "*the problem of automatically constructing a behaviorally equivalent state-based specification from the scenarios.*"

In the past, although synthesis was considered important and challenging, research on synthesis was carried out mostly to derive state-based models from scenario-based specification [2,6,9,11,15,16,18]. Recently, as the software systems and business processes of enterprises become more and more complex, process modeling through languages like UML Activity Diagram and BPMN appears as a new paradigm of software development. A notable example can be found in the Service-Oriented Architecture approach to software development [4], in which services are basic elements for constructing new complex services and the complex services are defined in terms of business process models that are to be automatically executed by a process execution engine. Therefore the ability to synthesize models in such languages from requirements would tremendously increase the scope of automatic software development.

This paper presents a method for synthesizing process models from scenario-based specifications [3,11], which can be formally specified with the UML Sequence Diagram [14] or the ITU-T Message Sequence Chart [1,7]. For process models languages, most well known are UML Activity Diagram [14] and BPMN [13]. Both of them are graphical notations for specifying workflows and business processes.

In contrast with synthesis of state machine models, the synthesis of UML Activity Diagram has different characteristics that pose unique challenges: First, UML Activity Diagram is typically used for system-level design. Therefore, the synthesis should not end with individually synthesized state machines bust should end with a holistic Activity Diagram that include all participants of the system. Second, the synthesis of state machine models would depend on state identification in Sequence Diagram. To figure out common states throughout multiple scenarios, previous researches either used explicit or implicit state labeling. In contrast, for synthesis of UML Activity Diagram, identify states would not work as Activity Diagram does not show states.

This paper presents a synthesis method that overcomes these challenges. To that end, first, we identify how control structures in Sequence Diagrams can be transformed to equivalent control structures in Activity Diagram and present it as mapping rules. Then we introduce labeling rules for adding new labels to Sequence Diagrams for explicitly representing the ordering of events in each process of Sequence Diagram, and, finally, provide a synthesis procedure which constructs an Activity Diagram from multiple scenarios.

The remainder of the paper is organized as follows: In Section 2, mapping rules and labeling rules for synthesis of

0730-3157/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSAC.2010.51

Sequence Diagrams to Activity Diagram are presented. Section 3 presents our synthesis procedure. Finally, Section 4 is the conclusion and discusses the contributions of the paper and the future research directions.

II. TRANSFORMATION RULES

There are seven basic control flow patterns that we identified in Sequence Diagram. The most elementary rule maps an action of an object in Sequence Diagram to an action in Activity Diagram. For the rest of the six control flow patterns, we associate two kinds of rules. The first is mapping rules for transforming parts of a Sequence Diagram into corresponding parts of an Activity Diagram. The second is labeling rules that are necessary for tracking while disassembling a Sequence Diagram into parts and reassembling the transformation results for them.

A. Rules for Mapping Control Flow Patterns

Mapping Rule 0 (The Action Pattern). A self call action of an object in a sequence diagram translates to an action of the corresponding participant of an Activity Diagram.

Figure 1. Mapping of the Reference Pattern

Notice that in drawing a sequence diagram in Figure **1**, we used a few simplifications not to clutter diagrams. So an object name is not boxed and activation is not explicitly shown.

As more complicated aspect of transformation, the control flow of the objects in Sequence Diagram can be categorized into the six kinds to be called *patterns*: 1) *simple interaction,* 2) *reference*, 3) *sequence*, 4) *branch and merge*, 5) *loop* and 6) *parallel split and join*. They are translated to Activity Diagram by applying the following six mapping rules.

Mapping Rule 1 (The Simple Interaction Pattern). The left hand side of Figure 2 shows a simple interaction which has just one message passing between two objects. Message passing description in Sequence Diagram can be translated using the notations such as send signal, receive signal, object nodes, and transition between the signal node and the object node. To represent such parallel behavior that represents message send event and message receive event between two concurrent objects, join and fork nodes are used.

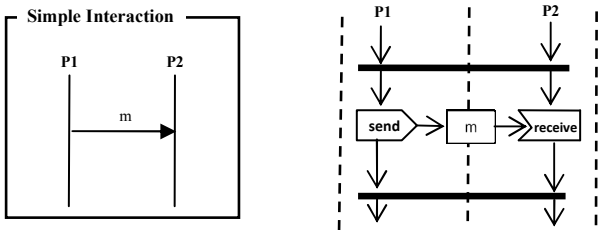


Figure 2. Mapping of the Simple Interaction Pattern

Mapping Rule 2 (The Reference Pattern). This pattern is used when a reference is used in a Sequence Diagram. The left hand side of Figure 3 shows how reference to a subdiagram ("K" in this example) is translated. Note that when reference is used in Sequence Diagram, we omitted explicit reference notation and simply used reference name. Then it should be expanded later after the expansion of the reference of Sequence Diagram is recursively translated into the Activity Diagram and connected to the rest of the Activity Diagram.

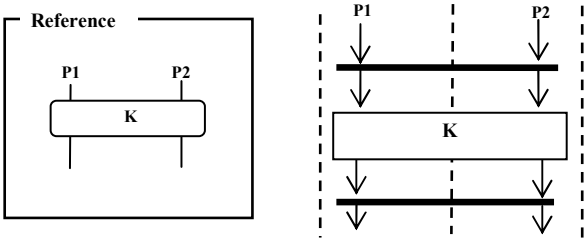


Figure 3. Mapping of the Reference Pattern

Mapping Rule 3 (The Sequence Pattern). This pattern represents a simple sequence of message passing between two objects. The order of message events is determined by the vertical positions along the lifeline of each object. The Sequence Pattern is illustrated in Figure **4**. When references are used, the reference type of Sequence Diagram specification is translated into the interaction occurrence type of Activity Diagram.

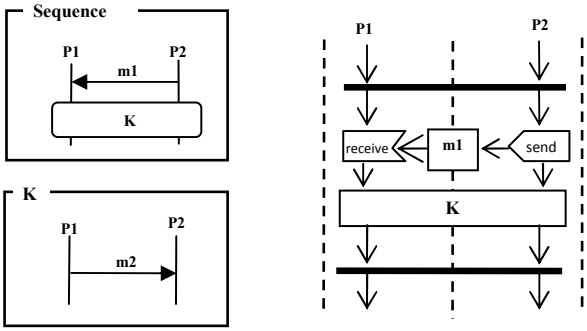


Figure 4. Mapping of the Sequence Pattern with a Simple Interaction and an Sequence Diagram reference

Mapping Rule 4 (The Branch and Merge Pattern). Application of the `alt` operator of Sequence Diagram can be transformed to Activity Diagram using the synchronization bars, decision nodes and the merge nodes. The operands in the `alt` operator contain multiple paths

with guard conditions and alternatives. All objects participating in the `alt` operator select the same path. In the corresponding Activity Diagram, all participants that correspond to the objects that selects are synchronized and forked before going into the alternative sections. Then the participants that finish the alternative sections are merged in the merge node as shown in Figure 4. This example and the following examples show the case for two objects but it can be generalized straightforwardly to the n objects case.

Mapping Rule 5 (The Loop Pattern). In order to transform a `loop` operator of Sequence Diagram into an Activity Diagram, setup, test, and body sections should be identified first. After finishing the body section, each process that participates in the body section goes back to the join node to check the condition again.

Mapping Rule 6 (The Parallel Split and Join Pattern). The `par` operator of Sequence Diagram represents parallel composition of multiple scenarios. By using the fork and join nodes of Activity Diagram, a participant can be split into multiple threads and joined together as in Figure 7.

B. Labeling Rules for Sequence Diagram

When transforming a specification in Sequence Diagrams to an Activity Diagram, fragments of Sequence Diagrams are labeled with Entry and Exit (EE) labels. EE labels provide information about relative positions of fragments of Sequence Diagrams so that the transformed fragments can be combined within an Activity Diagram [14] based on them. There are six labeling rules as Mapping Rule 0 does not have its corresponding labeling rule.

Labeling Rule 1 (The Simple Interaction Pattern). Insert labels before and after each message event of each object.

In each object of a Sequence Diagram, an entry label is inserted before and an exit label is inserted after a message event (Figure 8). Each label must be unique.

Labeling Rule 2 (The Reference Pattern). Insert labels before and after each reference in Sequence Diagram.

In each object of a Sequence Diagram, an entry label is inserted before and the exit label is inserted after a Sequence Diagram reference (Figure 8).

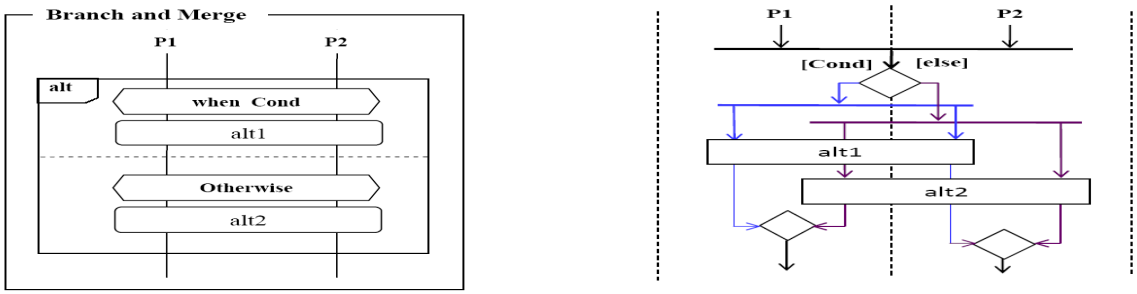


Figure 5. Mapping of the Branch and Merge Pattern

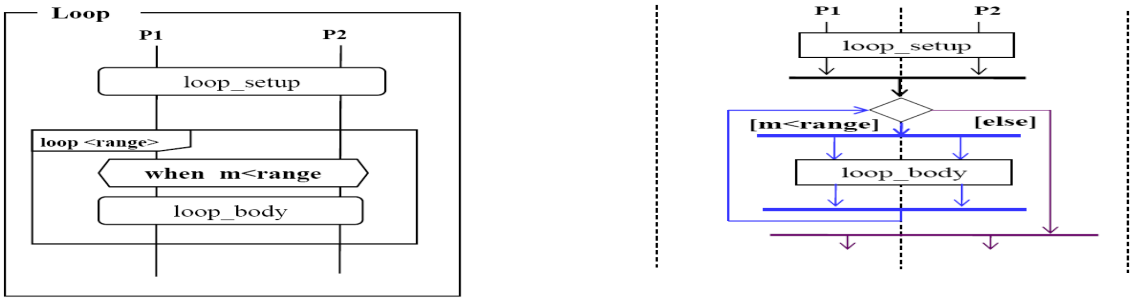


Figure 6. Mapping of the Loop Pattern

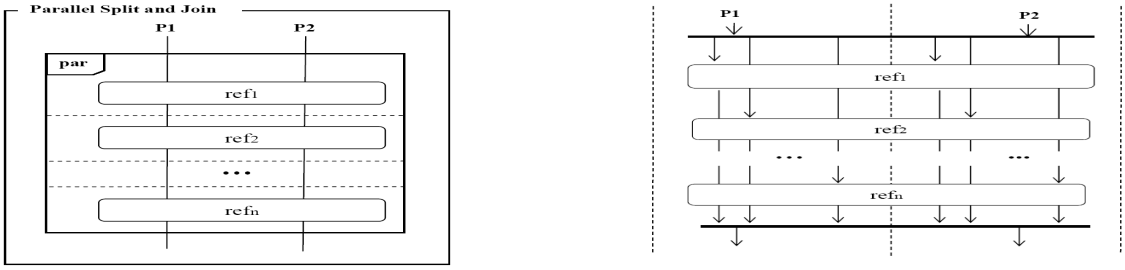


Figure 7. Mapping of the Parallel Split and Join Pattern

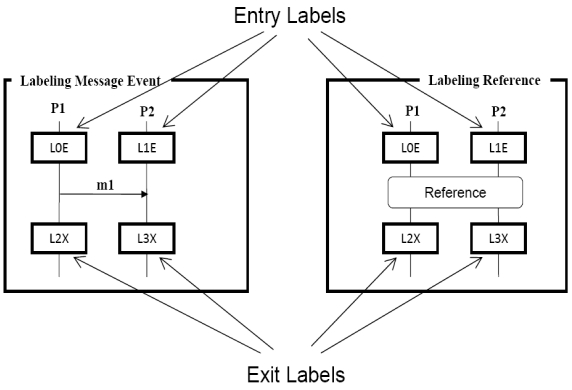
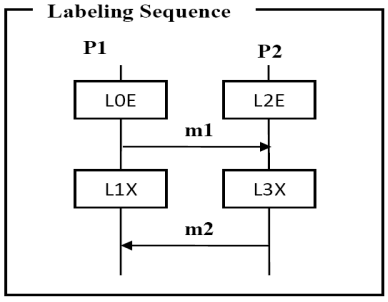


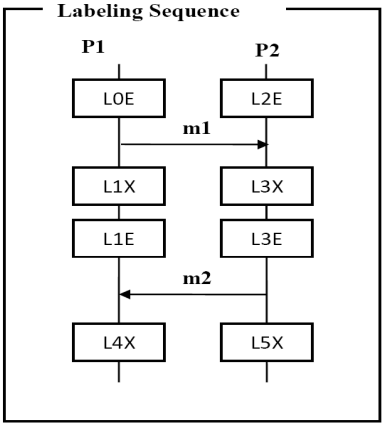
Figure 8. Application of Labeling Rules 1 and 2

Labeling Rule 3 (The Sequence Pattern). Insert the same entry label before a message event if a message event is immediately followed by another message.

With this rule, the exit label of the current message event and the entry label of the following message have the same label. In this way, the order of message events can be explicitly represented (Figure 9). This rule also allows fragments of Sequence Diagram to be easily connected to other transformed fragments in the Activity Diagram (Figure 10).



(a) Diagram before applying Labeling Rule 3



(b) Diagram after applying Labeling Rule 3

Figure 9. Application of Labeling Rule 3

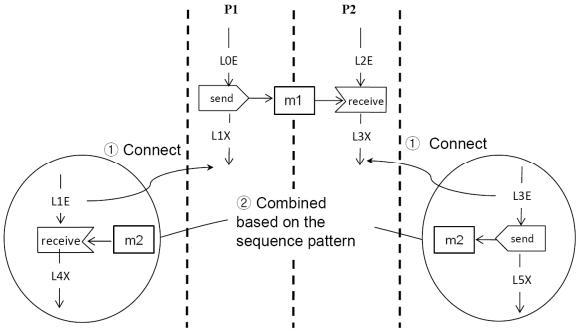
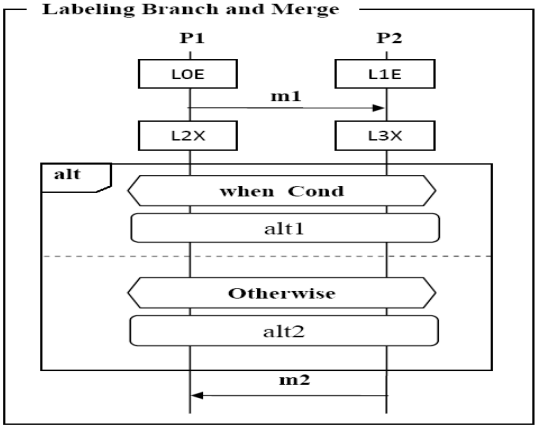
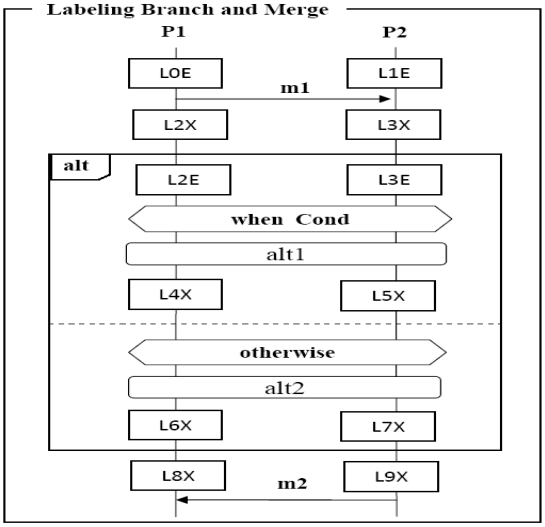


Figure 10. Connecting Sequence Diagram Fragments using Labels



(a) Diagram before applying Labeling Rule 4



(b) Diagram after applying Labeling Rule 4

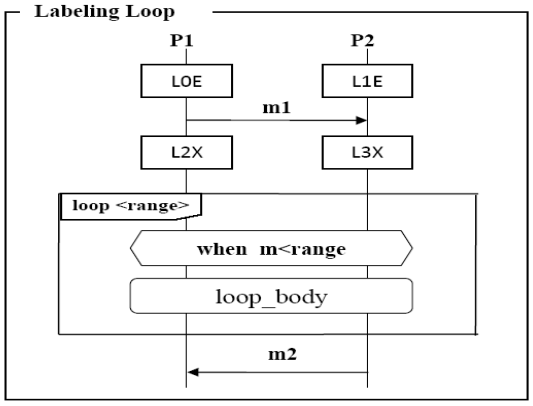
Figure 11. Application of Labeling Rule 4

Labeling Rule 4 (The Branch and Merge Pattern). If a Branch and Merge Pattern occurs, mark one entry label at the start of the branching flow, exit labels at the end of each alternative path per a participant and the end of `alt` operator.

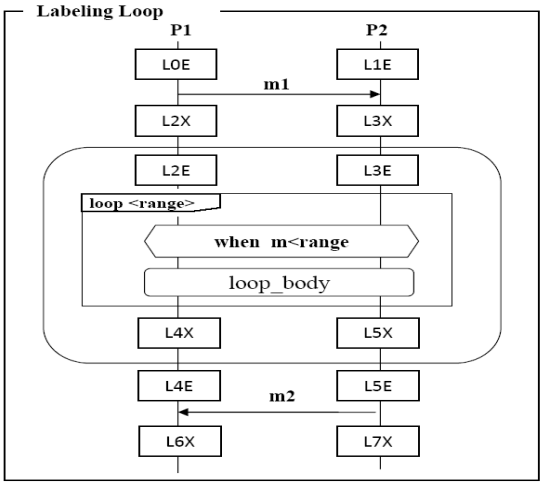
One entry and one exit exist at the start and at the end of the `alt` operator. Two or more exits may exist in a Branch and Merge Pattern as there can be at least two alternative paths. Thus when we label the branching structure with EE labels, we need to mark the labels outside the `alt` operator for one entry and one exit and multiple labels for each alternative path inside the `alt` operator. By Labeling Rule 4, we can explicitly represent which message events have occurred before and after the Branch and Merge Pattern (Figure 11).

Labeling Rule 5 (The Loop Pattern). If a Loop Pattern occurs, mark one entry label at the start of the loop flow structure and one exit label at the end of the flow structure.

The Loop Pattern has one entry and one exit per each object. Thus the `loop` operator is labeled with EE labels, at the start and at the end, per each object. With Labeling Rule 5, we can explicitly represent which message events have occurred before and after a Loop Pattern (Figure 12).



(a) Diagram before applying Labeling Rule 5



(b) Diagram after applying Labeling Rule 5

Figure 12. Application of Labeling Rule 5

Labeling Rule 6 (The Parallel Split and Join Pattern). If a Parallel Split and Join Pattern occurs, then mark one entry label at the start and one exit label at the end, respectively, of each control flow.

Application of this rule is not shown here but the overall structure is as shown in the right hand side of Figure 6 where each reference is labeled as with Labeling Rule 1.

III. SYNTHESIS PROCEDURE

In our synthesis procedure, we start with layering the Sequence Diagrams that possibly contain references to other Sequence Diagrams. The synthesis procedure is shown in Figure 13. The procedure takes a set of scenarios in Sequence Diagrams as input. It consists of three kinds of steps: Step 0, Step i (for $1 \le i \le n$) and Step n+1. Step 0 is the initial step and layering of the Sequence Diagrams is performed in this step. Steps 1~n are represented by Step i, which should be repeated n times where n is the number of layers determined in Step 0. Step n+1 is the last step of the procedure.

In the procedure, each step consists of a number of substeps and each substep is either performed mechanically without human intervention or performed manually by humans. The substeps involving human guide are lines 1 and 8. The rest of the substeps are performed mechanically. In line 1, the user determines the layering of the Sequence Diagrams. In line 8, the user decides how to partition them into fragments based on human intelligence.

Step 0:	1. *Make layering of the Sequence Diagrams*
	2. n = number of layers
	3. Draw swim lanes
	4. i = 1 // i indicates the current layer
Step i:	5. L = the set of MSCs in layer i
($1 \le i \le n$)	6. **while** $L \neq \emptyset$ **do**
	7. Select a Sequence Diagram $M \in L$
	8. *Determine applicable mapping patterns and partition M into fragments* (The same pattern may apply multiple times)
	9. Label fragments with EE labels using the labeling rules
	11. Apply the mapping rules
	12. Connect fragments with the same labels
	13. Remove M from L
	14. **endwhile**
	15. **if** i < n **then**
	16. i = i + 1
	17. **goto** 5
	18. **end if**
Step n+1:	19. Insert the initial node and the final node
	20. Remove the EE labels

Figure 13. The Synthesis Procedure

In this procedure, partitioning of the Sequence Diagrams and the resulting labeling with the EE labels should be done based on insight on what locations in of scenarios are equivalent and what locations are not. After identifying the control flow patterns to be used, the labels are inserted at the positions as defined by labeling rules. Then mapping of the patterns to Activity Diagram and connecting the partitioned fragments based on the EE labels are carried out.

Sequence Diagram references are expanded to basic Sequence Diagrams until there are no more Sequence Diagram references or the current layer is the lowest one. As we expand Sequence Diagram references, we apply labeling rules and mapping rules to each Sequence Diagram. Then they are transformed to the elements of an Activity Diagram and combined.

We applied our synthesis procedure to the adapted version of the online bookstore example in Weiss et al. [17] to demonstrate that the transformation rules and the synthesis procedure correct and work well with a larger size of problem. The full details of the case study can be found in [8].

IV. CONCLUSION

In this paper, we presented the transformation rules for synthesis of Sequence Diagram specifications to UML Activity Diagrams. These rules consisted of mapping rules and labeling rules and were incorporated into our synthesis procedure for constructing a UML Activity Diagram from multiple scenarios.

Through those transformation rules and the synthesis procedure, we showed the possibility of automating business process modeling from requirements through. An important application of our synthesis method would be development of the SOA style applications [4] where services are basic elements for constructing application and complex new services can be designed with the languages such as UML Activity Diagram or BPMN. In that approach, , our work can be used for automated conversion of Sequence Diagrams as a choreography description — which specifies dynamic interactions and message flows among existing applications — into UML Activity Diagram or BPMN. The newly defined or modified use cases can get into the business process model through our synthesis procedure. Ultimately, we expect that our approach will reduce time and cost for developing a system by bridging the gap between requirements and business process model, thus contributing to business process automation.

For future work, we plan to improve our method by further automating the manual steps of the synthesis procedure and also to develop a tool that implements our transformation rules and synthesis procedure.

REFERENCES

[1] R. Alur, K. Etessami, M. Yannakakis, "Inference of Message Sequence Charts," IEEE Transactions on Software Engineering, v.29 n.7, p.623-633, July 2003.

[2] F. Bordeleau, J.-P. Corriveau, and B. Selic, "A scenario-based approach to hierarchical state machine design," *The 3rd IEEE Int'l Symposium on Object-Oriented Real-time Distributed Computing (ISORC '0*0), March 2000.

[3] A. Cockburn, "Structuring Use Cases with Goals," *Journal of Object-Oriented Programming*, Sep-Oct and Nov-Dec, 1997.

[4] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design,* Pearson Education, Inc., 2005.

[5] D. Harel, H. Kugler, "Synthesizing state-based object systems from LSC specifications," *Int'l Journal of Foundations of Computer Science*, 13(1):5–51, Febuary 2002.

[6] D. Harel, H. Kugler, and A. Pnueli, "Synthesis revisited: Generating statechart models from scenario-based requirements," *Formal Methods in Software and System Modeling*, LNCS 3393, pp. 309-324, 2005.

[7] ITU-T, *Message Sequence Charts,* ITU-T Rec. Z.120, April 2004.

[8] H. H. Kim, *Synthesis of UML Activity Diagram from Scenario-based Specification*, Master of Science Thesis, KAIST, Korea, 2009.

[9] I. Krűger, R. Grosu, P. Scholz, and M. Broy, "From MSCs to statecharts," *Int'l Workshop on Distributed and Parallel Embedded Systems (DIPES '98)*, 1999.

[10] H. Liang, J. Dingel, Z. Diskin, "A comparative survey of scenario-based to state-based model synthesis approaches," *2006 Int'l Workshop on Scenarios and State Machines: Models, Algorithms and Tools*, pp. 5-12, 2006.

[11] S. Leue, L. Mehrmann, M. Rezai, "Synthesizing ROOM models from Message Sequence Chart specifications," Technical Report, Dept. of Electrical and Computer Engineering, Univ. of Waterloo, 1998.

[12] Z. Manna, R. J. Waldinger, "Toward automatic program synthesis," CACM, Vol.14, Issue 3, pp. 151-165, 1971.

[13] Object Management Group (OMG), *Business Process Modeling Notation (BPMN) Version 1.1*, January 2008.

[14] Object Management Group (OMG), *UML 2.1.2: Superstructure Specification*, OMG document ptc/2007-11-01 November, 2007.

[15] M. Sgroi, A. Kondratyev, Y. Watanabe, L. Lavagno, and A. Sangiovanni-Vincentelli, "Synthesis of Petri Nets from Message Sequence Charts Specifications for Protocol Design," *The Design, Analysis and Simulation of Distributed Systems Symposium (DASD '04)*, April 2004.

[16] S. Uchitel and J. Kramer, "A workbench for synthesizing behaviour models from scenarios," *The 23rd IEEE International Conference on Software Engineering (ICSE '01)*, May 2001.

[17] M. Weiss and D. Amyot, "Business Process Modeling with URN," *Int'l Journal of E-Business Research, 1(3)* 63-90, July-September 2005.

[18] J. Whittle and J. Schumann, "Generating statechart designs from scenarios," *The 22nd International Conference on Software Engineering (ICSE'00)*, pp. 314–323, 2000.