A Statechart-Based Approach of Eliciting Implied Scenarios

Hongyuan Wang , Jiachen Zhang and Tie Feng

Key Laboratory Symbolic Computation and Knowledge Engineering of Ministry of Education
College of Computer Science and Technology, Jilin University, Changchun, 130012, China

Summary
Scenarios present system behaviors by specifying collaboration and interaction between objects or components from users' perspectives. Statecharts are precise descriptions of system behaviors. Automatic transformation from scenarios to statechart is the process in which system behavior models are generated automatically from UML requirement models. In this paper we propose an approach to eliciting implied scenarios based on generated statechart from scenarios. This approach presents some rules of constructing state vectors that assist analysts to add semantic information to scenarios expressed by sequence diagrams, identify implied state transition paths by adding the scenarios information based on existing algorithm which supports the design process by generating statechart design automatically from scenarios, and synthesize implied scenarios by implied state transition paths with which analysts or users can further refine their requirements.

Key words:
UML, Scenarios, Sequence diagrams, Statechart, Implied state transition path, Implied Scenario

1. Introduction

Scenario models play central roles as tools for requirement elicitation and specification in current object oriented system modeling processes. A scenario may represent a concrete sequence of interactions steps or a set of possible interactions steps between system components, the environment and users. Scenarios may be expressed as UML[2] sequence diagrams.

When it comes to the dynamic aspects of a system, state machines (particularly statecharts, originally introduced by D. Harel [3]) represent a compact way of describing these aspects. Statecharts are finite state machines extended with hierarchy and orthogonal (parallelism), allowing the representation of a system in a compact and elegant manner.

While scenarios represent a single trace of behaviour of a complete set of objects, state machines (which we are going to refer to as statecharts from now on) represent the complete behavior of a single object.

Works have been done to explain how to automatically generate state machines from scenarios. These works lead automatically to maintain consistency between sequence diagrams and state machines. In this paper we present an approach to how to get implied scenarios from the implied state transition paths which are generated from scenarios. This backwards direction work can help users further refine their requirements.

Section 2 introduces the existing algorithm. Section 3 presents some derived rules of state vectors that assist analysts to add semantic information to scenarios expressed by sequence diagrams, and introduces the synthesis scenarios algorithm from implying state transition path. In Section 4 Related work is discussed. The conclusions and future directions of our work are given in Section 5.

2. The Existing Algorithm of Generated Statechart

Scenarios are instances of use cases. The lack of semantic information in the description of scenarios, such as sequence diagram and collaborate diagrams, results in different interpretations when analysts try to comprehend the system, so the analyst can't get the states of objects. J Whittle and J Schumann suggested using OCL to identify the object states in the sequence diagram in 2000[1].

J. Whittle's algorithm generates object statechart using scenarios labeled state vector based on OCL(Object Constraint Language). OCL specification provides the pre-conditions before the message is sent and the post-conditions after the message is received. The variables involved in the conditions are used as the state variables of the object states. The set of the state variables, called the state vector, determine state value. OCL can be represented as follows:

Message description:
preCon: varible1=value1, variable2=value2, …
postCon: varible1=value1', variable2=value2', …
This information is expressed as the state vectors like this : < variable1$^\wedge$, variable2$^\wedge$,…, variablen$^\wedge$ >. The value of variable is Boolean.
The whittle's algorithm can describe :

1. specify the preconditions and postcondition of each message of sequence diagrams with OCL.
2. capture state variables of the state vector which describe state from OCL specification.

Manuscript revised January 2006.

3. each message extend the state vextors by propagating variable values throughout the sequence diagrams The objects are assigned state vectors based on OCL information and the algorithm of [5] before or after transferring messages.
4. generate each object's statechart of single sequence diagram based on state vector. Repeat this step with deferent sequence diagram.
5. merge the similar state generated from 4 for each object.

This approach can generate statechart of each object in the sequence diagram. The algorithm is simple and clear. In this paper we propose some rules of constructing state vector. These rules will capture the reasonable and sufficient state variable to construct state vector. And we modify merge rule to keep some state transition paths which are not explicit in sequence diagrams. We named implied state transitions. In whittle's algorithm the implied state transition paths are neglected to a great extent..

3. Implied Scenarios

3.1 Choose State Variables to Construct State Vector

In this paper we analyze use case to construct the primary state vectors. The state vectors would be modified and complemented based on the primary ones.

Usually the use case descriptions consist of the simple descriptions, use case diagram, preconditions, basic flow, alternative flows, subflows and postconditions.

We analyze the useful information mainly from the preconditions, basic flows and alternative flows.

Rule 1: system internal objects can be prepared as state variables in the preconditions.

Rule 2: interactions of actor and system would be state variables. They will describe whether or not the information that system needs has input into the system, the system output has delivered to actor, and actor has responded. These state variables be usually expressed by the synthesized word, such as inputPasswd, takecard.

Rule 3: the objects in assumed sentence of the alternative flows usually can change the use case control flow. These ones can be state variables.

Considering the above rules, we can identify two classes of primary state variables. One is the Verb Phrase that describes the interactions between actor and system. The other is the noun that describes the object possibly changing the use case control flow. In order to verify whether the primary state variables can satisfy the need of describing the objects states, we design the following questions about the messages of the sequence diagrams that describe scenarios. We define some rules to query analysts as Patrick Heymans does[7]. There are five questions to be asked to decide the state variables of source object which sends this message, that is, it performs a certain action:

1. Which conditions, in the circumstance given in the scenarios, allow the action to take place?
2. What, in the circumstance given in the scenarios, forces the action to take place?
3. Which conditions, in the circumstance given in the scenarios, allow the action occurrence to have an effect on the object that performs it?
4. What is the effect of the action occurrence on the object which performs it?
5. Whether or not the objects outside the system interact with the system and what information exchange?

For the target object which receives the message, that is, receives the event sent by the other object, there are two questions which can decide its state variables:

1. Which conditions in the circumstance given in the scenarios, allow the event have the effect on the object received it?
2. What is the effect of the event received on the target object?

By answering these questions, the analysts would get complete state variables to confirm the state of source object.

3.2 Extract Implied State Transition Path

The state vector get by our proposed approach can satisfy the need of generating the statecharts. We think the state information is so sufficient that generated state transition paths are reasonable. The last step of whittle's algorithm is integrating object's statecharts generated from different sequence diagrams. Some state transition paths which are not explicit in sequence diagrams exist as the byproduct of merging similar state. We named implied state transitions. In Whittle's algorithm the implied state transition paths are neglected to a great extent. But implied state transition paths can help user refine the requirements. So we modify merge rule like following:

sc(n): scenario which state node n is generated from.

u(n): the state vector value of state node n.

t(n, l, m): the transition between state node n and m, *l* is the label of message which trigger transition.

1. $u(n1) = u(n2)$
2. $sc(n1) \neq sc(n2)$
3. $if \exists l$, there exists transitions $t1 = (n3, l, n1)$ and $t2 = (n4, l, n2)$, then $n1$ and $n2$ are similar,
4. After the nodes that satisfy 3 have merged, the nodes that only satisfy 1 and 2 will be merged.

For example, three object A's statecharts generated from different scenarios are going to merge.

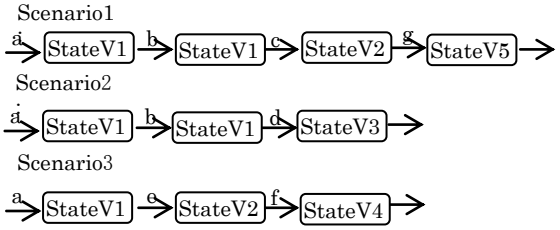

Fig. 1 The statecharts of object A.

Applying whittle's algorithm, the statechart is generated like Fig. 2. Applying our algorithm, the statechart is generated like Fig. 3. The implied state transition paths are generated like Fig. 4.

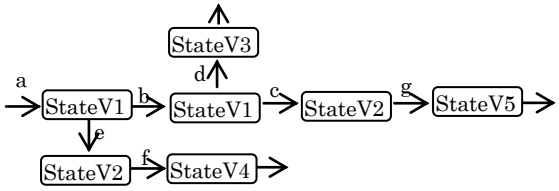

Fig. 2 Merged statechart with Whittle's algrithom

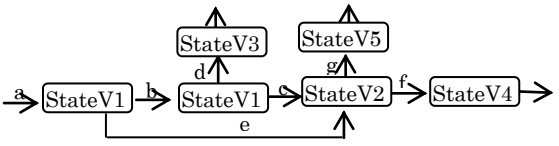

Fig. 3 Merged statechart with our algrithom

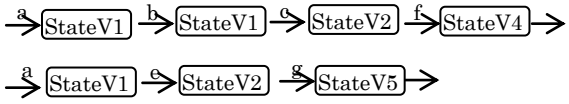

Fig. 4 The implied state transition paths

In order to implement our merging rule, the data structure of state transition consists of four elements:
Message lable *l*, state vector of source end *preStateV*, state vector of target end *postStateV*, the set of scenario *sc(t)* which transition *t* belongs to. Because the transition *t* possibly belongs to multi scenarios after merging, *sc(t)* is a set of scenarios. When transversing generalized statechart and getting state transition paths, we can judge implied state transition paths with an algorithm as follows.

1. compute the set *scs=sc(t0)* of the first transition *t0* in the path.
2. input the next transition *t* in the path by order, *scs=scs∩sc(t)*
   if *scs*=∅
   then this is implied state transition path
   else goto 2for example, like Fig. 5.

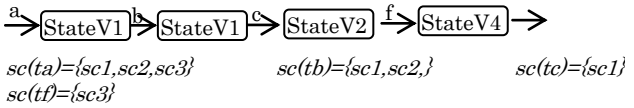

Fig. 5 An example of judging implied state transition paths

3.3 implied scenarios

The statechart generated from different scenarios of the same use case. The implied scenarios can be constructed from the implied state transitions. The implied scenarios can help to complete the requirements specification with unforeseen situations or indicate that the specification must be refined to prevent unwanted executions.

The sequence diagrams describe the interaction of multi objects. The set of objects *O* is given.

Definition 1. The sequence diagram of the scenario is the structure *S={O,V,M,L}*.

- *O* is a set of objects participating in the interaction of the sequence diagram, $o \in O$.
- *V* is the set of vertexes mapped the interactions on the objects lifelines of the sequence diagram.
- *M* is the set of messages, $M \subseteq \left[\cup_{o \in O}\left(V_o \times L \times V_o\right)\right] \cup \left[\cup_{o,p \in O, o \neq p}\left(V_o \times L \times V_p\right)\right]$
- *L: M→L*, the label of message.

Definition 2. The *trace* $Trace \subseteq (M \times M)$ is a total order of the set of messages in the sequence diagram.

Definition 3. The *snippet* is a partial trace which records the total order of the set of messages between two consecutive states of an object. The structure *Snippet={S_i, S_n,t}*

- S_i is a state of the studying object.
- S_n is the next state of S_i
- *t* is the trace between S_i and S_n.

We can link the snippets of one finite state transition path into a trace of the set of messages. We may neglect some objects whose states change simply, but only focus on the complicated ones. For the concerned object:

1. The snippet is attached to the corresponding transition when generating state.
2. When merging the similar state, the transition only keeps one snippet.

When finding out the implied state transition path, we synthesize the implied scenarios by linking the snippets.

We consider a use case of the elevator system. Use case CallProcess describes how the elevator responses the call of passenger and reach the floor desired. There are several objects anticipant this use case, including buttons, control, drive, dispatch, and safety devices. Thinking about two scenarios of CallProcess:

1. Passenger presses the call button in the hall. an immobile elevator responses this call and reached the floor of passenger staying.

2. Passenger entered an immobile elevator, and pressed the car button for the desired floor. The elevator moved to the floor desired, but can not stop. Safety device wakens the brake to make car stopping.

The Control is the most important object. it sends control information to other objects. According the method given above, we can develop the state vector of Control as fellow:

<move, validCall, callLocation, desiredFloorGiven, safety, doorState>

and set the values of state variables of Control in every phrase.

We can get the statechart of Control as Fig. 8. To describe the state path implied, we mark every transition with corresponding scenarios. There are two implied path in this statechart. One of the implied path is gray in Fig.8. That is, there are two new scenarios

1. Passenger pressed the call button in the hall, an immobile elevator responses this call, but can not stop at desired floor. Safety device wakens the brake to make car stop.
2. Passenger entered an immobile elevator, and pressed the car button for the desired floor. The elevator moved to the floor desired.

For example, the implied scenario 1 is described like Fig. 9. It indicates that whether the call happens in the hall or in the car, there is probability that the elevator can't stop.

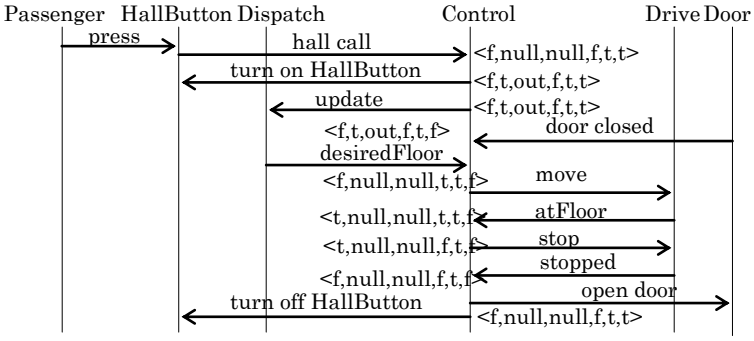

Fig. 6 Scenario 1

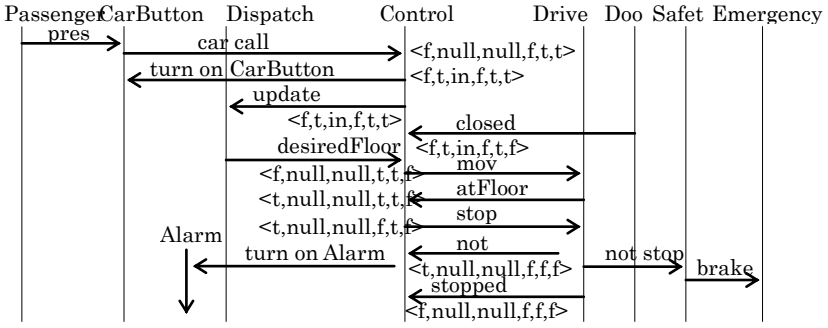

Fig. 7 Scenario 2

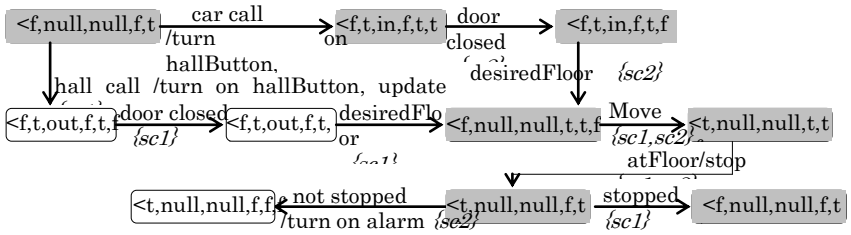

Fig. 8 Generated statechart

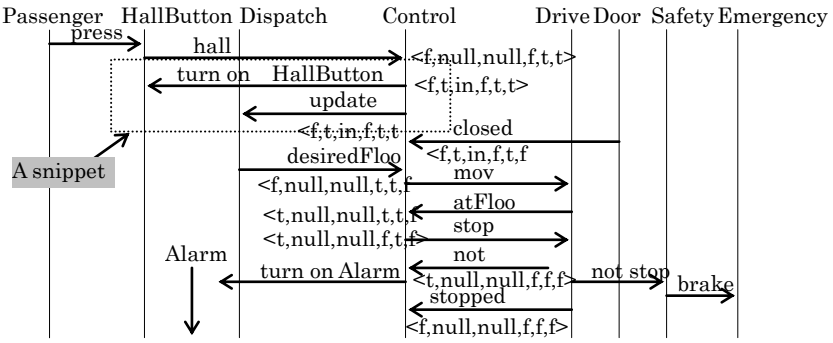

Fig. 9 implied Scenario 1

4. Related Work and Conclusion

Kai Koskimies and Erikki Makinen first proposed the method of generating objects' statecharts from scenarios by using syntax inference theory that is getting the behavior model of an object from the traces [4]. Koskimies accomplished SCED tool based on the SMS algorithm. Because it identifies the same state only on the basis of action, there will be overgeneralized problem, which can possibly produce some wrong or implied paths in the generated statechart. So Erikki Makinen and Tarja Systa propose MAS algorithm on the basis of SMS algorithm in 2000[5,6]. This technique is semi-automatic. Makinen and Systa regard that it is difficult to identify the overgeneralized paths automatically during creating the statechart.

Sebastian Uchitel, etc. Presented a framework for synthesizing implementation models for scenario-based specification[8,9]. The framework has been implemented and integrated in the LTSA tool, also provides a method for assessing if a scenario specification has implied behaviours. Implied scenarios are the result of specifying the global behaviours of a system that will be implemented component-wise.

Our work extends the algorithm presented by J. Whittle in [1]. We have used their notions of state vectors and algorithm generated statecharts. We present some rules of constructing state vectors that assist analysts to add semantic information to scenarios expressed by sequence diagrams. In [1] the state nodes which have same state vector and different transition label can not be merged. However, we accept it as possible transition path. We identify implied state transition paths by adding the scenarios information based on existing algorithm which supports the design process by generating statechart design automatically from scenarios, and synthesize implied scenarios by implied state transition paths with which analysts or users can further refine their requirements. In the future, we will consider the relation between multi-usecase and statecharts.

References

[1] Whittle, J. and Schumann, J. Generating statechart designs from scenarios. In Proceedings of International Conference on Software Engineering (ICSE2000), Limerick,Ireland (2000), 314~323.

[2] Object Management Group (OMG): Unified Modelling Language Specification version 1.5. OMG, Needham, MA, USA (Mar. 2003).

[3] D. Harel. Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8:231-274,1987.

[4] K. Koskimies, E. M□akinen: Automatic synthesis of state machines from trace diagrams. Softw. Pract. Exper. 24, 643~658 (1994).

[5] Mäkinen E., Systä T.: Minimally adequate teacher designs software, Dept. of Computer and Information Sciences, University of Tampere, Report A-2000-7, April 2000. Submitted. (ftp://ftp.cs.uta.fi/pub/reports/pdf/A-2000-7.pdf)

[6] Mäkinen E., Systä, T.: Implementing minimally adequate synthesizer, Dept. of Computer and Information Sciences,University of Tampere, Report A-2000-9, June 2000. (ftp://ftp.cs.uta.fi/pub/reports/pdf/A-2000-9.pdf).

[7] H. Behrens Requirements Analysis and Prototyping using Scenarios and Statecharts In Proceedings of ICSE 2002 Workshop: Scenarios and State Machines: Models, Algorithms, and Tools. 2002.

[8] S. Uchitel, J. Kramer and J. Magee. Detecting Implied Scenarios in Message Sequence Chart Specifications. In Proceedings of. European Software Engineering Conference (ESEC/FSE'01), pp.74-82, Vienna 2001.

[9] S. Uchitel. Synthesis of Behavioral Models from Scenarios. IEEE Transactions on Software Engineering, VOL. 29, NO. 2, pp. 99-115, February 2003.

Hongyuan Wang received the B.E. and M.E. degrees, from Jilin Univ. in 1998 and 2001, respectively. After working as a research assistant (from 2001), she has been an instructor (from 2004) in the College of Computer Science and Tech., Jilin Univ. Her research interest is software engineering.