Statechart-based Use Case Requirement Validation of Event-Driven Systems

Saurabh Tiwari
Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, INDIA
saurabh.tiwari@iiitdmj.ac.in

Atul Gupta
Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, INDIA
atul@iiitdmj.ac.in

ABSTRACT

In this paper, we present an approach of validating functional requirements modeled as a set of use cases. In this approach, we perform requirement analysis both at use case level as well as system level. The approach consists of two steps. In step one, each use case is independently modeled by a UML Statechart, which is then analyzed for any state-specific anomalies. In step two, we perform system-level analysis by substituting each use case by its Statechart to validate all possible types of usages of the system. We demonstrate the applicability of the proposed approach using a case study.

Keywords

Software requirements, functional requirements, use case modeling, UML Statechart, validation.

1. INTRODUCTION

Software requirement engineering is the process of discovering the needs of the customers and documenting them in a form that is useful for analysis, communication, implementation, and maintenance of the software [3, 2]. The goals will always vary and conflict, depending on different perspectives of the users in which they work and the tasks they wish to accomplish [1]. One frequent problem a requirement analyst faced with is validating the requirements after gathering them [2, 4, 1]. Early the better, validating requirements in the requirement phase is a prerequisite for developing quality software, that also helps in reducing development time and various risks involved in the development.

In this paper, we make use of UML Statecharts to identify problems in requirements of a event-driven system that are specified as a set of use cases. Since each use case is documented in isolation, integrating use cases may result in revealing discrepancies and inconsistencies in the requirements. Information organization of a UML use case model typically does not facilitate the detection of conflicts to be explicit. However, we show that a transformation of this information in the form of a UML Statechart enables automatic detection of conflicts in the requirements.

In this approach, we first perform the use case level analysis by separately mapping each use case to a Statechart by identifying the events, transitions and associated actions. We compute additional semantic information in the form of state invariants obtained from the preconditions and post-conditions documented in the use cases. Using a Event-State matrix reflecting events and state invariants relationships, we develop a Statechart for each use case in the use case model. The generated Statechart is then analyzed for requirement related anomalies. Next, we perform the system level analysis by combining Statecharts so obtained as per the use case diagram. By enumerating all possible paths in the combined Statechart, this analysis helps to validate all possible types of usage of the system. We demonstrate the applicability of the proposed approach using a case study on a software game called RealState [6] where we report some conflicts in the problem description.

The organization of the paper is as follows. Next section briefly describes the proposed approach of generating the Statechart both at use case level and system level and subsequent requirement analysis using them. In Section 3, we present the case study and finally, we summarize our conclusions in Section 4.

2. THE APPROACH

The simplicity of use case models makes them suitable for expressing the requirements as they can be easily understood by the customers, requirement engineers, and software developers but the lack of a definite semantic structure of the information in the use case model makes them ambiguous and therefore difficult to interpret [3]. To overcome this problem we use a more formal use case model (see Figure 1) that allow us to perform a behavioral analysis, both at individual use case level as well as system level, by generating UML Statecharts from the use case requirements. For a generated Statechart, the state variables, in the form of a state vector, are used to characterize the specific states that are identified by the values assigned to these state variables.

Figure 1 shows the use case template to capture a piece of functionality of a event-driven system. In this model, we represent main flow events M1,M2, ..., Mj, sub flow events S1,S2, ..., Sk, and alternate flow events A1,A2, ..., An, along with optional actions associated with each of them. A use case may have associated precondition events P1,P2, ..., Pi, -events that must occur to perform the intended functionality, as well as postcondition events with actions T1,T2,...,

SAC'12 March 26-30, 2012, Riva del Garda, Italy.

Tm, -events that suggest successful completion of the intended functionality.

USE CASE <No>	< Use Case Name > Provides a short meaningful name
Goal	Represents the goal of the use case
Preconditions {P1,P2,...,Pi}	Preconditions indicate circumstances that must be true prior to the invocation of any event of the use case. Alternatives of preconditions are possible. They are ordered by numbering with letters (P1, P2,..., Pi)
Main Flow {M1,M2,...,Mj}	Main flow indicates the intended functionality of the given use case.
	Events / **Actions***
	[M1] / {MA1}{MS1}
	... / ...
	[Mj] / {MAj}{MSj}
Sub flow* {S1,S2,...,Sk}	Sub flows are the additional flows that are started along with main flow.
	Events / **Actions***
	[S1] / {SA1}
	... / ...
	[Sk] / {SAj}
Alternate flow* {A1,A2,...,An}	Alternate flows will always indicate the exceptional scenarios.
	Events / **Actions***
	[A1] / {AA1}
	... / ...
	[An] / {AAj}
Postconditions {T1,T2,...,Tm}	The postcondition validate that the main scenario and all alternative scenarios successfully achieved the stated goal of the use case
Comments	Notes for comments { if needed}

* *Optional*

Figure 1: Use case template

2.1 Use Case Level Analysis

We perform the use case level analysis by generating a UML Statechart for each use case. An state in the Statechart is represented by the specific values assigned to the state variable included in the state vector. A state vector is a set of boolean state variables formed by taking the union of precondition and postcondition of the use case model with the initial state vectors where a state variable is a condition which may be true or false for a specific state. Figure 2 shows the model for generating Statechart from the use case specifications.

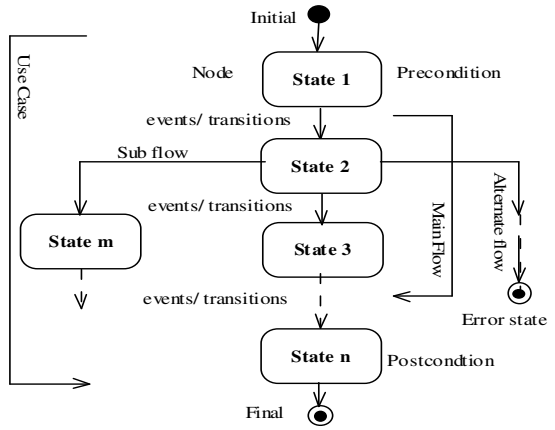

Figure 2: Model for single use case analysis

We generate the Statechart by identifying the initial state, which is represented by specific boolean value assignments corresponding to the preconditions specified in the use case documentation, followed by generating subsequent states that might result due to the change in state variable values caused by main flow events [5]. In this process, we use a Event-State matrix (denoted by E-S matrix) specifying the relationship between the events specified in the use case documentation and the resulting values of the state variables. An entry in the Ei-Sj matrix shows the value of state variable Sj after the occurrence of event Ei.

2.2 System Level Analysis

This analysis performs by integrating the individual Statecharts as obtained during use case level analysis, incrementally according to the use case diagram. This allows us to identify the interface related anomalies between the use cases. Moreover, once the integration is achieved, we can use the resulting combined Statechart to obtain all possible scenarios (nominal as well as exceptional). Traversing through these scenarios can lead to the identification of missing or incorrectly documented requirement, if any. Figure 3 shows the model to combine use case for system level analysis.

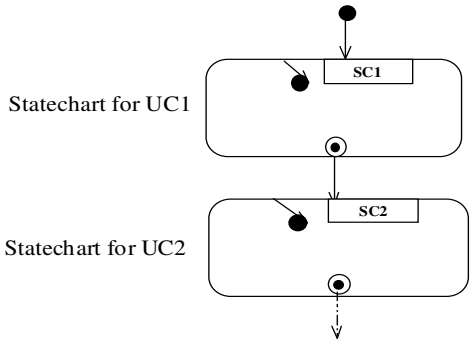

Figure 3: Combining use cases for system level analysis

3. CASE STUDY: THE GAME REALSTATE

This section presents a case study of validating use case requirements using the proposed approach. We apply the approach to a Monopoly-like computer game called RealState [6]. The use case documentation of the RealState consists of a total of fifteen use cases. Here we show the use case level analysis for UC7: Purchase Tradable Cell. The use case template for the use case UC7 is shown in Figure 4. Figure 5 shows a part of the Event-State matrix highlighting the effects of relevant events in use case UC7 on the state variables. Figure 6 shows the generated Statechart for the use case UC7. The generated Statechart is then analyzed for requirements related anomalies. We then performed the system level analysis by combining the Statecharts obtained for each use case and analyzed the resulting combined Statechart for more requirements related anomalies. The results of both the analyzes are shown in Table 1.

4. CONCLUSIONS

In this paper, we presented an approach of validating use case requirements by generating Statechart both at use case level as well as system level. By modeling the semantic information in the form of Statecharts, we are able to identify states and state variables to detect and report anomalies.

Use case Name	**UC7:** Purchase Tradable Cell		
Goal	Purchase the cells that is available if it has no owner		
Preconditions **Pre[1], ..., Pre[i]**	**Pre[1]**: Players turn **Pre[2]**: Player rolled the dice **Pre[3]:** Player lands on an available tradable cell		
Main Flow **{M1,M2,...,Mj}**	**Events**	**Actions**	
	[M1] Click on purchase button	Updated display	
	[M2] Click pay amount	Pay the price of Cell **[A1][S1]**	
	[M3] Hits end turn button	Updated display	
Sub flow **{S1,S2,...,Sk}**	**Events**	**Actions**	
	[S1] Click, if enough money	Pay the amount **[A2]**	
Alternate flow **{A1,A2,...,An}**	**Events**	**Actions**	
	[A1] Click, doesn't have enough money	Next player turn begins **Post[1]**	
	[A2] Hits end turn button	Updated display	
Postconditions **Post[1],..., Post[m]**	**Post[1]**: Player turn ends		

Figure 4: Use case template for UC7

Events/ State Variables	**S1**	**S2**	**S3**	**S4**	**S5**	**...**	**S13**
...	...	...	...	...	...	...	...
Purchase button	0	0	1	0	0	...	0
Pay amount	0	0	0	1	0	...	0
End turn button	1	0	0	0	1	...	0
Enough money	0	0	0	1	0	...	0
Not enough money	1	0	0	0	1	...	0
...	...	...	...	...	...	...	...
Event-State Matrix {Total number of events in UC specification X State vector(S1,S2,..,Sn)} where, {1= True, 0= False}							

Figure 5: Part of Event-State matrix (UC7)

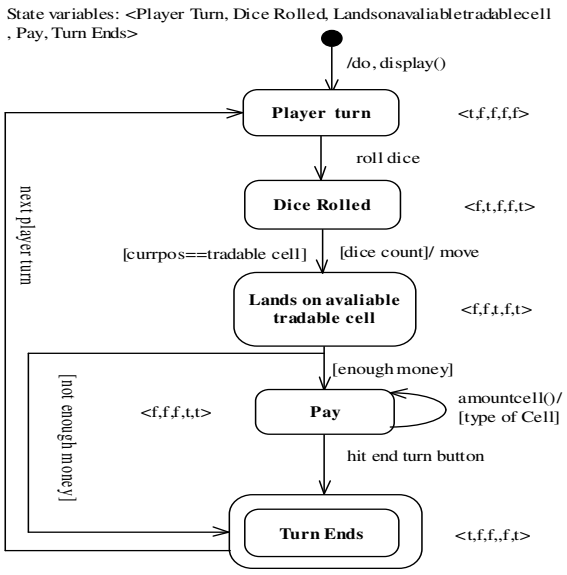

Figure 6: Statechart for use case UC7

Table 1: Results of analysis

Outcomes	**UC level**	**System level**
Conflicts detection	None	Three
Loops detection	Yes	Yes
Missing requirements	None	Three
Incorrect requirements	None	One

We have demonstrated the applicability of the proposed approaches using a case study where we map a use case to a Statechart, followed by the analysis of the generated Statechart. Next, we perform a system level analysis by combining the Statecharts so obtained as per the use case diagram. This analysis resulted conflicts detection, loops detection and identification of missing/incorrectly documented use case requirements.

We aim to extend this work to include the analysis of more complex behavior of the system and develop an automated tool for the approach presented in the paper.

5. REFERENCES

[1] H. Behrens and F. Hagen. Requirements analysis and prototyping using scenarios and statecharts. In *International Conference on Software Engineering*, 2002.

[2] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Transaction on Software Engineering Methodology*, pages 231–261, July 1996.

[3] R. Jeffords and C. Heitmeyer. Automatic generation of state invariants from requirements specifications. In *Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering*, SIGSOFT '98/FSE-6, pages 56–69, New York, NY, USA, 1998.

[4] G. Kösters, H.-W. Six, and M. Winter. Coupling use cases and class models as a means for validation and verification of requirements specifications. *Requirements Engineering*, pages 3–17, 2001.

[5] E. Sekerinski. Verifying statecharts with state invariants. In *Proceedings of the 13th IEEE International Conference on on Engineering of Complex Computer Systems*, IEEE Computer Society, pages 7–14, Washington, DC, USA, 2008.

[6] The RealState Game. *http://openseminar.org/se/modules/20/index/screen.do*, [Accessed May 2nd 2011].