2012 IEEE/ACIS 11th International Conference on Computer and Information Science

Functional and User Interface Model for Generating Test Cases

Zhu Bin, Wang Anbao
School of Computer and Information
Shanghai Second Polytechnic University
Shanghai, China
tozhubin@163.com, abwang@it.sspu.cn

***Abstract*—Use cases models are used to specifying functional requirements whereas task models are employed to modeling UI requirements. Test cases generated from use cases models are focused on the core functionality of the system, while test cases generated from task models are used to user interface testing which concerned with details of user interactions. Therefore, test cases derived from use case models or task models only capture partial system behavior .These test cases are inadequate for testing full system behavior.**

In this paper we employ use cases models for modeling system functionality and user action notation (UAN) for describing user interfaces. We assumed that the task model and use case model are consistent and they refined each other in some part of model. Use case models and task models are transformed to FSM respectively. We propose a method for formally integrating the model for use case models and task models. The resulting integrated model is then used to generate test cases which capture more complete and detailed user interactions and secondary system interactions.

Keywords- use case model; task model; UAN; FSM; test case

I. INTRODUCTION

Interactive application has been developing rapidly around the world, especially in mobile devices and web applications. The importance of the user interface component is being more and more recognized and a lot of effort is being spent in the academic and industrial community in order to make user interfaces able to support a wide variety of activities, in a wide variety of contexts and available to a larger and larger number of users[1]. Modeling is one of the most recognized techniques for design of interactive systems. These models are simplifications of reality. The main characteristic of model is to consider and to represent some aspects of a problem while leaving out other ones deemed less relevant for a specific objective. Software today is more complex than ever. This approach is particularly useful to manage the real applications. Model-based approaches are exactly to identify relevant abstractions and to analyses interactive software applications from a more logical point of view.

In traditional software engineering, the Unified Modeling Language (UML) has become the standard notation for software models. UML is a family of diagrammatic languages to modeling software. The user interface should also be modeled using UML. In fact, UML is a natural candidate notation for UI modeling. However, it is by no means always clear about how to model user interfaces using UML because the developers of UML did not pay much attention to how to support the design of the interactive components of a software system. It is not easy to identify how user interface elements are supported in UML application models. There are few reports on projects specially applying UML for modeling the UI [3]. Thus, a number of specific approaches have been developed to address the model-based design of interactive systems. Task models have been developed to address the model-based design of interactive systems. Task models can be useful to provide an integrated description of system functionality and user interaction.

In practice, use case models are used to modeling functional requirements, whereas task models are employed to describe user interface requirements and designs. Any one of the two models cannot capture the full system behavior. In this paper, we propose an integrated model from use case model and task model. The task model is described by User Action Notation (UAN). The UAN is a user and task oriented notation that describes behavior of the user and interface as they perform a task together. A common formal semantics for use case and task models are based on nondeterministic finite state machine (FSM). The two models are merged to a composite FSM. Existing techniques are used to generate integrated test cases.

The remainder of this paper is structured as follows: In Section 2, we introduce background information on use case and task models. Section 3 presents method for Transforming task models and use case models to FSMs. Based on the formal semantics we define the integration of both models and generate test cases from composite model in Section 4. Section 5 reviews relevant related work and we conclude in Section 6.

II. USE CASE AND TASK MODELS

Use cases have become a widespread practice for capturing functional requirements. A use case captures the interaction between actors and the system under development. It is organized as a collection of related

978-0-7695-4694-0/12 $26.00 © 2012 IEEE
DOI 10.1109/ICIS.2012.54

success and failure scenarios that are all bound to the same goal of the actor [4]. Use cases are primary elements in software development. They drive all development work from initial gathering of requirements to design, implementation, deployment and testing.

A use case is a set of scenarios tied together by a common user goal，without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behavior required of the system, bounding the scope of the system. Usually, every use case starts with a header section consisting of various properties (e.g., primary actor, goal, etc.). The core of a use case is its main success scenario, which lays out the most common way for the primary actor to reach their goal when using the system. Use case extensions define alternative scenarios which may or may not lead to fulfillment of the use case goal [5].

In the development of e-commerce website, you would have a Buy Products use case with the successful purchase and the authorization failure as two of the use case's scenarios. A simple format for capturing the use case involves describing its primary scenario as a sequence of numbered steps and the alternatives as variations on that sequence, as shown in Figure 1.

Use Case: Buy Products
Properties
Primary Actor: Customer
Secondary Actor: Payment System
Goal: Primary Actor buys a product
Main Success Scenario
1. Customer browses through catalog and select items to buy
2. Customer goes to check out
3. Customer fills in shipping information
4. System verifies product's availability in requested amount
5. System presents summary information
6. Customer fills credits card information
7. System authorizes purchase
8. System confirms sale
9. System send confirming email to customer

Extensions
Alternative: authorization failure
At step 7, system fails to authorize credit purchase
Customer to re-enter credit card information and retry
Alternative: product is not available
At step 4, desired product is not available
System informs Primary Actor of unavailability.

Figure 1. "Buy Products" Use Case

Buy Products of specification of a task in terms of its subtasks and their temporal relationships is described below.

Task Buy Products:

Choose Products(t_1)
Fills in Shipping Information(t_2)
Check Availability(t_3)
Purchase(t_4)
Task Choose Products:
Browses Products(t_{11})
Select Products(t_{12})
Select Quantity(t_{13})
Submit(t_{14})
Check Out(t_{15})
Task Fills in Shopping Information:
Fill Delivery Day(t_{21})
Fill Address(t_{22})
Task Check Availability:
Display Out of Stock(t_{31}) | Presents Summary Information(t_{32})
Task Purchase:
Fills Credits Card Information(t_{43})
Display Authorization Failure(t_{41})| Display Confirmation(t_{42})

This expression describes the task Buy Products and then one or more sequence of the tasks composing the expression.

Basic tasks can be further described with UAN. In UAN the expression ~[object] means that the user moves the cursor to point on the object indicated. The $\vee$ $\wedge$ symbol represent press and release a button. The task Submit can be described as

~[Button_sub] $\vee$ $\wedge$

Button_sub is the name of the submit button. In the paper, Expression of basic task will not be considered. More UAN symbols for describing basic task can be found in [7, 8].

Use cases and task models both belong to scenario-based notations, and can capture sets of usage scenarios of the system. In theory, use cases and task models can be used to describe the same information. Some differences between use cases model and task models are given in [5, 9]. Use cases capture requirements at a higher level of abstraction whereas task models are relation to lower level UI details. Task models don't capture internal system interactions which may be specified in use cases. A task model may only describe one of the scenarios specified in the use case.

III. FORMAL SEMANTICS FOR USE CASES AND TASK MODELS

In this paper, the formal semantic for use case and task models is nondeterministic finite state machine (FSM).

Definition 1: A deterministic finite state machine is defined as quintuple (S,Σ,s_0,δ,F), where:

- S is a finite, non-empty set of states.
- Σ is the input alphabet (a finite, non-empty set of symbols).
- s_0 is an initial state, an element of S.
- $\delta:S\times\Sigma\rightarrow S$ is the transition function which returns for a given state and a given input symbol the set of states that can be reached.
- F is the set of final states, a (possibly empty) subset of S.

A. *Transforming Use Cases to FSM*

In this section we define a mapping from the textual description of use cases to the FSM. Usually a use case consists of a main success scenario and a set of extensions. Both the main success scenario and extension consist of a sequence of steps. These steps can be of three different kinds: Basic steps are performed either by an actor and contain no sub steps; Choice steps provide the actor with a choice between several interactions; Goto steps means jumps to other steps within the same use case.

We propose representing each use case step as a transition of finite state machine. For each use case step, the steps of constructing a FSM are presented as follows, using q_0 as a starting state:Basic step: Create a new state q_{post} and a new transition (q_{pre}, {L}, q_{post}). q_{pre} is the last state that has been create and L is labeled as a identifier of the currently visited use case step. If there exists an extension for the currently step then, using q_{post} as a starting state, repeat the same procedure for each step defined in the of the extension

- Choice step: For each of the choice repeat the same procedure as basic step.
- Goto step: Continue with the next step referenced in the step. If the target step has been already visited then add a transition definition that included q_{post}, accordingly.

Figure 2 illustrates the FSM generated from the use case of Figure 1.

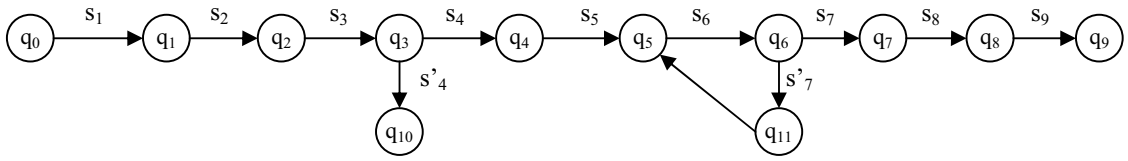

Figure 2. FSM of Buy Products Use Case

B. *Transforming UAN Task Models to FSM*

In this section, we will demonstrate the mapping from UAN task models to FSM. In UAN, action tasks are composed to complex tasks using a variety of temporal operators. We propose representing task as a transition of finite state machine. UAN temporal operators are mapped into relationship of transitions. In what follows we demonstrate how tasks and UAN temporal operators are semantically mapped into FSM.

In the same abstract level, we supposed that FSM state is q_i before executing task A, so the sequential execution of task A and task B are mapped into FSM as Figure 3.

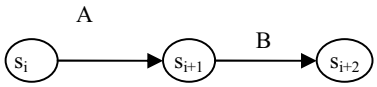

Figure 3. FSM of two Sequence Tasks

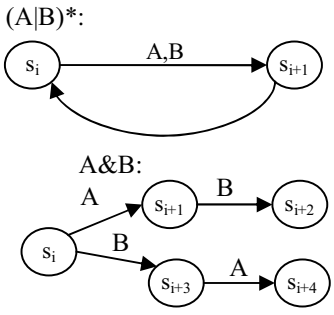

Figure 4. FSM of Temporal Relationships of Tasks

In Figure 4, the order independency (A&B) operator is rewritten as the choice of either executing A followed by B or executing B followed by A. Another example is the concurrency (A÷B) operator, which can be rewritten as the choice between all possible interleaving of action tasks entailed in A and B. Similar rewritings can be established for the interruptibility and mutual interleavability operators. In UAN task model, tasks can be hierarchically decomposed into sub-tasks until an atomic level has been reached, so FSM can be constructed from each task. Figure 5 describes the FSMs generated from task model Buy Products at every abstract level.

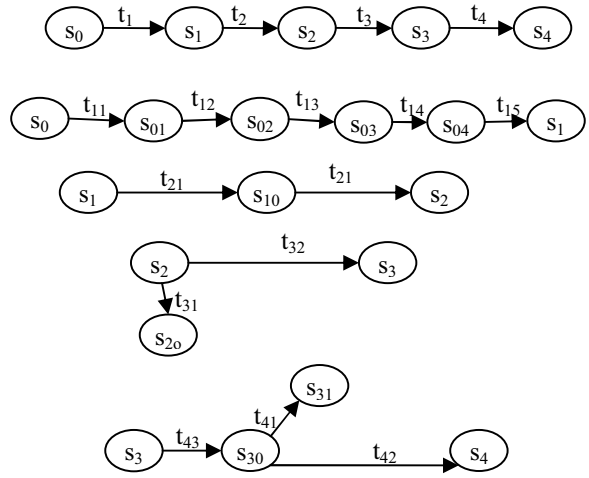

Figure 5. FSMs for each task

After generating FSMs for each task, we can use a FSM replace upper level transition of the FSM from bottom to top, according to the state. For example use t_{41} and t_{42} replace t_4. We can get a flat FSM as Figure 6.

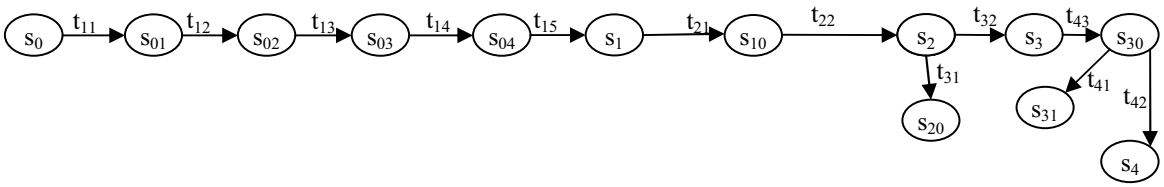

Figure 6. FSM for Buy Products Tasks

IV. COMPOSITE MODEL FOR GENERATING TEST CASE

Both the FSM representing the "Buy Products" use case and the FSM representing the "Buy Products" task model can be used separately to generate test cases. Test cases generated from the FSM of task model do not cover interactions with system actors, whereas test cases generated from the FSM of use case do not cover the detailed primary interactions. Each model generated test cases will only cover parts of the involved interactions. Since task models are more detailed than use case models, we assume some state both in FSM of task model and in FSM of use case model. Let Q is state sets of FSM of use case model and S is state sets of FSM of task model, then $Q \cap S \neq \emptyset$.

Definition 2 Let $M_1=(Q_1, \Sigma, q_{01}, \delta_1, F_1)$ be the FSM representing the use case U, $M_2=(Q_2, \Sigma, q_{02}, \delta_2, F_2)$ be the FSM representing the task model T and $M_{ut}=(Q_3, \Sigma, q_{03}, \delta_3, F_3)$ representing the Composite model, where:

$Q_3 = Q_1 \cup Q_2$

$q_{03}=q_{01}$

$$\delta_3(q_{i3},a)\begin{cases}\delta_1(q_{i3},a) & q_{i3}\in M_1 \wedge q_{i3}\notin M_2\\ \delta_2(q_{i3},a) & q_{i3}\in M_2 \wedge q_{i3}\notin M_1\\ \delta_1(q_{i3},a) & q_{i3}\in M_1 \wedge q_{i3}\in M_2 \wedge \delta_1(q_{i3},a)\in M_1 \wedge \delta_2(q_{i3},a) \text{ not defined in } M_2\\ \delta_2(q_{i3},a) & q_{i3}\in M_2 \wedge q_{i3}\in M_1 \wedge \delta_2(q_{i3},a)\in M_2 \wedge \delta_1(q_{i3},a) \text{ not defined in } M_1\\ \delta_2(q_{i3},a) & \text{other}\end{cases}$$

$F_3=F_1 \cup F_2-F'$ $\quad F'=\{q|(q\in F_1 \wedge q\notin F_2 \wedge q\in Q_1 \cup Q_2) \cup (q\in F_2 \wedge q\notin F_1 \wedge q\in Q_1 \cup Q_2)\}$

According to the definition 2 we can get the Composite FSM which can be reduce further. Let p and q are two states in FSM M**,** there exist two accepting sequence σ_1 and σ_2 from p to q. If σ_1 is more detail than σ_2 then remove $\sigma 2$ in FSM M, otherwise remove σ_1. Figure 7 describes steps and composite model.

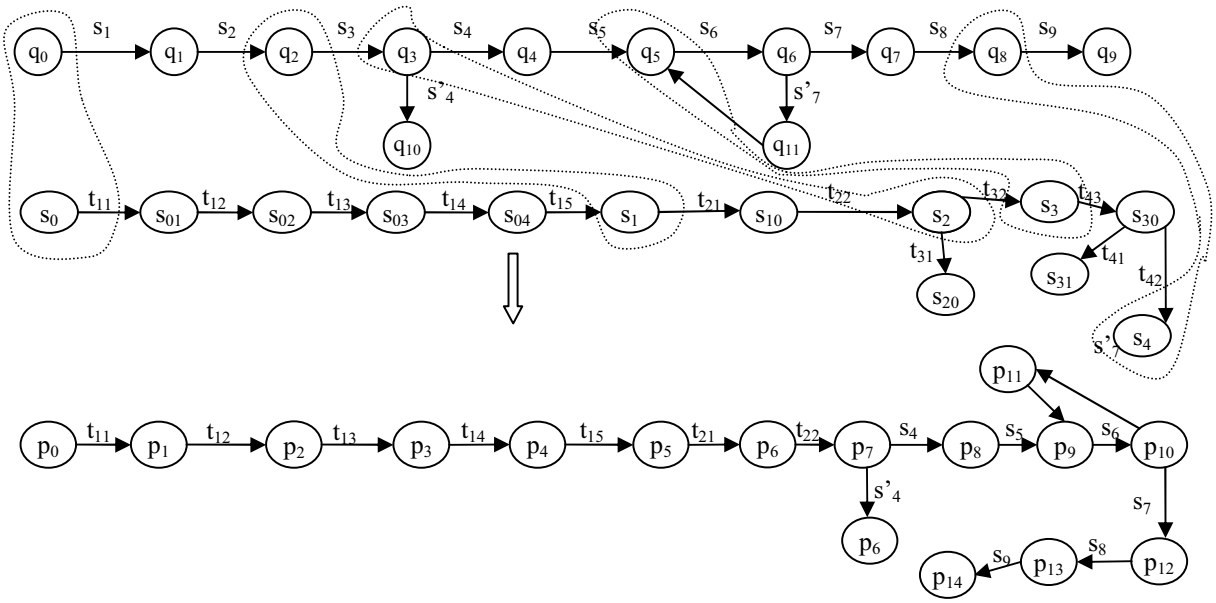

Figure 7. Composite FSM

Base on the context, state q_0 in use case FSM and s_0 in task FSM are identical, the other identical states are marked in Figure 7. The sequence $t_{11}t_{12}t_{13}t_{14}t_{15}$ are more detailed than s_1s_2, so s_1s_2 is reduced in composite FSM. Once the composite FSM has been obtained, it can also be used to further drive development of the system. We are interested in

using this composite model for generating test cases which integrated functionality and UI. Test cases generated from this model will cover the detailed user interactions, as well as secondary interactions. Test case generation from FSM has been extensively studied in some literature and any one of several test case generation techniques and strategies available can be used [10-13].

V. Related Work

Usually, test cases are typically derived from use case models or from task models respectively. Test case generation from use cases models has been proposed by various researchers. Nebut et al.[14] propose a approach for automating the generation of system test scenarios in the context of object-oriented software, taking into account traceability problems between high level views and concrete test case execution. Both functional and robustness test cases from parameterized use cases, where preconditions and post conditions for use case execution are formalized in OCL. Gutierrez et al. [15] studied how to generate test cases using use cases for Web applications. They used activity diagrams to represent the behavioral model of the system's requirements. Froehlich and Link [16] propose a approach to generating system-level test cases from use cases. Each use case is transformed into a state machine which is then used for test case generation based on certain test criteria. Kassel [17] presents a methodology to generate test cases from structured use cases. Requirements are documented in XML as structured use cases and test cases are automatically generated from the XML use cases.

Giulio Mori et al.[18]present a tool that provides thorough support for developing and analyzing task models of cooperative applications, which can then be used to improve the design and evaluation of interactive software applications. Silva, J et al. [19] shown that task models can be used to generate oracles economically in an interactive systems model based testing context. Tool support enabling the use of task models as oracles for model-based testing of user interfaces is being developed. The approach presented by Benz [20] enables the generation of test cases for integration testing that cover the typical error prone interactions between different system components. A set of task model specific coverage criteria such as task coverage, temporal relationship coverage, and random selection is also defined. Daniel Sinnig et al.[5]propose a formal integration of use case and task models for the purpose of deriving test cases. The common semantic domain for use case and task models is a typed labeled transition system (tLTS). The use case and task model tLTSs is used to generate separate functional or user interface test cases.

VI. Conclusion

In this paper, we proposed a method to create an integrated model for the generation of functional and user interface test cases. Use case and task models are merged to obtain integrated model. Use case models are employed to document functional requirements, while task models are used to capture UI requirements and design information. FSM as a formal semantic for use case and task models is the basis of the behavioral merge. The use case and task model FSM can be used to generate test cases respectively. These test cases capture only partial system behavior; moreover, the integration of the detailed user interactions and the interactions with secondary actor is ignored by such test cases.

We have shown that under the assumption that the task model and use case model are consistent and they refined each other in some part of model, the artifacts can be combined to get a more accurate model of the system. This model integrates the internal interactions and the detailed primary interactions. Technically, the merge is performed by combining the state sets of FSM representing both artifacts in such a way that the resulting FSM is alternatively either in a task FSM transition or in a use case FSM transition. The resulting FSM can then be used to generate more complete and detailed test cases covering user interactions and secondary actor's interactions. As future work, we plan to implement more comprehensive case studies and assess the fault detection ability of test cases.

References

[1] Carmen Santoro, A Task Model-Based Approach for the Design and Evaluation of Innovative User Interfaces. Presses universitaires de Louvain, 2005

[2] Paternò, F., From Model-based to Natural Development. In Proceedings HCI International 2003, 592-596.

[3] Pinheiro da Silva, P., Paton, N., User Interface Modelling with UML. In Proc. of the 10th European-Japanese Conference on Information Modelling and Knowledge Representation, 2000.

[4] Larman, C., Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process, Prentice Hall PTR, Upper Saddle River, NJ, 2002.

[5] Sinnig, D., Khendek, F, Chalin, P., A Formal Model for Generating Integrated Functional and User Interface Test Cases. In ICST(2010) 255-264

[6] Giulio Mori , Fabio Paternò , Carmen Santoro, CTTE: support for developing and analyzing task models for interactive system design, IEEE Transactions on Software Engineering, v.28 n.8, p.797-813, August 2002

[7] H. R. Hartson, Temporal Aspects of Tasks in the User Action Notation. Human-Computer Interaction, 7:1–45, 1992.

[8] Hartson, H. R., Siochi, A. C., & Hix, D. (1990), The UAN: A user-oriented representation for direct manipulation interface designs. ACM Transactions on Information Systems, 8, 181-203.

[9] Sinnig, D., Chalin, P. and Khendek, F., Consistency between Task Models and Use Cases, in Proc. of DSV-IS 2007, Salamanca, 2007.

[10] [10]Chow, Tsun. Testing Software Design Modeled by Finite-State Machines, IEEE Trans. on Software Engineering, Vol SE-4, #3, pp 178-187, May 1978

[11] Huaikou Miao, Zhongsheng Qian, Tao He, Modeling Web Browser Interactions Using FSM. The 2nd IEEE Asia-Pacific Service Computing Conference (APSCC' 2007), IEEE Computer Society, Tsukuba Science City, Japan, Dec. 11-14, 2007, pp. 211-217.

[12] Zeng Hongwei and Miao Huaikou, Model checking-based testing of Web applications. Wuhan University Journal of Natural Sciences, Volume 12, Number 5, 2007,922-926

[13] D. Lee, and M. Yannakakis, Principles and methods of testing finite state machines, Proceedings of IEEE, 84(8):1090-1123, 1996.

[14] Nebut, C., Fleurey, F., Le Traon, Y. and Jezequel, J.,Automatic Test Generation:A Use Case Driven Approach, in IEEE Trans. Softw. Eng., 32 (3), pp. 140-155, 2006.

[15] Gutierez J., Escalona M. J. and Torres M. M., An Approach to Generate Test Cases from Use Cases. Proceedings of the 6th International Conference on Web Engineering. pp. 113-114 (2006)

[16] Fröhlich, P. and Link, J., Automated Test Case Generation from Dynamic Models, in Proc. of ECOOP'00, Sophia Antipolis and Cannes, France pp. 472-492, 2000.

[17] Kassel, N., An Approach to Automate Test Case Generation from Structured Use Cases, Thesis in Computer Science, lemson University, Clemson, South Carolina, 2006.

[18] Giulio Mori, Fabio PaternoÁ , and Carmen Santoro , CTTE: Support for Developing and Analyzing Task Models for interactive System Design, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 28, NO. 8, AUGUST 2002

[19] Silva, J., Campos, J. C. and Paiva, A., Model-based user interface testing with Spec Explorer and ConcorTaskTrees, in Proc. of Formal Methods for Interactive Systems Macau,China, 2007.

[20] Benz, S., Combining Test Case Generation for Component and Integration Testing, in Proc. of AMOST'07, London, UK, pp. 23-33, 2007.