

Requirements Analysis Using Statecharts and Generated Scenarios

Henrik Behrens
FernUniversität Hagen
58084 Hagen
Germany
+49 2331 987 2966

Henrik.Behrens@fernuni-hagen.de

ABSTRACT

This paper shows a way to model the requirements of a business information system using scenarios and statecharts. Redundancy problems are avoided by generating the scenarios from an integrated structured statechart model. The approach uses precise action semantics, supports changing requirements and provides animated scenario visualization for requirements validation by end users.

1. INTRODUCTION

1.1 Using Scenarios For Requirements Analysis

When an analyst asks a domain expert to describe the functionality of business information system which is to be built, the domain expert will usually describe the functionality in terms of scenarios. For example, if an information system for a company offering seminars has to be developed, he might describe the functionality as follows:

“Consider a customer calling by telephone, let's say his name is Mr. Smith. He wants to attend one of our seminars, let's say, the UML seminar. We first have to note his name and address. If he is not yet present in our customer database, we enter his name and address etc. into the system. Then we check if there are still empty places for the UML seminar he wants to attend. If so, we enroll him for this seminar and tell him about the seminar details (price, date etc.). The computer system must maintain an account for that customer so that we can always check his payment balance. If the UML seminar costs \$250, the computer system must generate an accounting entry, which causes his account be charged \$250.”

We encounter the following properties of such a scenario description:

- The scenario is written in natural language and is easy to understand for domain experts, analysts and system developers.
- The scenario describes a sequential series of user and/or system actions.
- The scenario represents a typical usage of the system to be implemented.
- The scenario description may contain concrete data (“Mr. Smith”, “UML seminar”, “\$250”).

We define a scenario to be a sequential series of user and/or system actions that might reasonably take place. This definition is very close to the definition in [4].

We define a scenario to be *concrete*, if the scenario description uses concrete data (*design by example*, [7]). Otherwise we call them *abstract*.

1.2 Problems With Scenario Usage

Scenarios have become key artifacts in systems engineering, but their management is poorly understood [3]. Scenarios are a good communication base with customers and other non-technical people and support early validation of requirements at a low abstraction level [5]. However, the full potential of scenarios is not always fully exploited because of problems including the following [5][8]:

- modeling the behavior of a whole system with scenarios requires a great multitude of scenarios
- scenarios are highly redundant, because some parts of scenarios are common to many scenarios
- the redundancy in the scenarios makes changes tedious and can cause inconsistencies between them
- missing traceability between scenarios and other software artifacts can cause inconsistencies between different models
- UML interaction diagrams (which are often used to describe scenarios) lack adequate expressiveness and semantic foundation
- misunderstandings can happen because of unclear semantics

1.3 Solutions

Our approach

- avoids redundancy by combining scenarios to statecharts and by allowing statecharts to be used in more than one context
- provides dynamic visualization of scenarios and the underlying object structure, allowing dynamic execution of scenarios right from the state model representation
- supports changing requirements by automatically updating affected scenarios when the statechart model is modified
- supports a mixture of formal and informal statements for the description of scenario semantics, using the UML conform Action Specification Language (ASL) [13].

2. Eliminating Redundancy Using User Interface Statecharts

2.1 Concept

Redundancy in scenarios can be eliminated by generating a statechart model from the scenarios [9][10][12]. The scenarios are incorporated and merged into the statechart model and can then be generated back from the statechart using a statechart driver [10]. We use a user interface oriented view to define statecharts representing the functional requirements of a business information system. To ensure local understanding, we define exactly one statechart diagram each use case. This diagram defines the *main flow* and all *exceptional flows* of that use case [2].

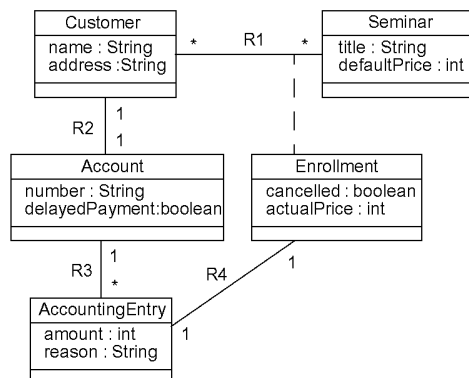


Figure 1. Class diagram of a seminar information system (SeminarIS)

2.2 Example:

Consider the class diagram in Figure 1 and the statechart diagram in Figure 2. The statechart diagram represents all scenarios (flows) of the use case “Enroll a customer for a seminar”. One can consider the scenario described in the introduction as the *main flow* of this use case. Other flows include the possibility that the seminar may have to be created (before it can be selected) and different error conditions (e.g. that the customer is already enrolled for the requested seminar).

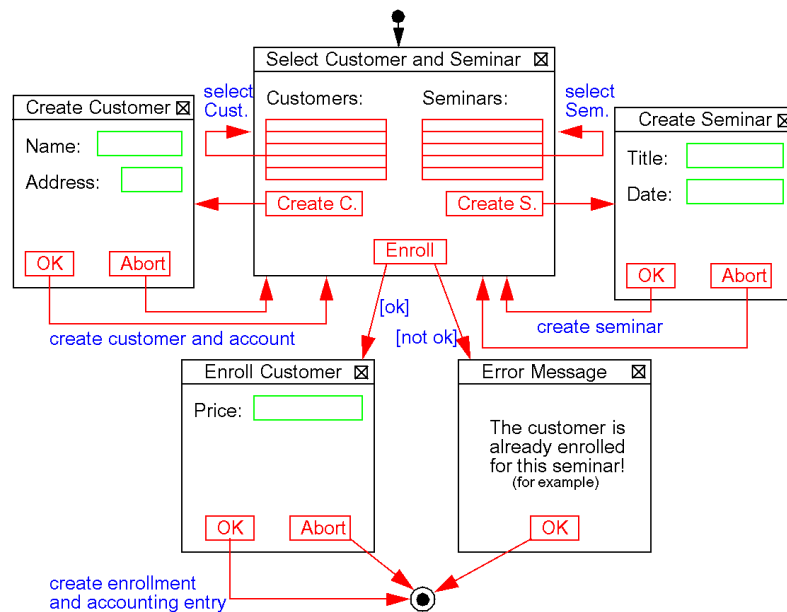


Figure 2. A user interface statechart for the use case “Enroll a customer for a seminar”.

An equivalent description of the statechart diagram in Figure 2 with single scenarios would require a great number of scenarios, making up a very redundant model. Instead, we model scenarios by modifying a statechart, so that scenarios and statecharts are developed in concert rather than sequentially [11].

3. How to Define Precise Action Semantics For Scenarios

For describing the semantics of scenarios, we use both natural language and a formal notation. Natural language is good for recording requirements at an early stage, when great expressiveness and ease of use are more important than formal correctness and executability. On the other hand, sometimes "analysis" gets confused with "vagueness" leading to models that cannot be understood without significant interpretation by the system designers [6]. Therefore we use precise action specifications (in addition to natural language action descriptions) to capture requirements in an unambiguous and exact way and to enable animated scenario visualizations, which can be very valuable for requirements validation by the end user and as basis for later implementation. We use the Action Specification Language [13] (which is conform to the emerging UML action language specification [14]) to specify action semantics and guard conditions formally and thereby make scenarios executable. During scenario execution, the domain object structure is dynamically visualized, so that one can watch the changes to the object structure after each scenario step.

4. Generation of concrete scenarios from statecharts

As we prefer to work with concrete scenarios rather than abstract ones, we have to provide concrete data values for each scenario being incorporated into a state machine. The concrete values come from user input during scenario execution.

4.1 Example

Consider the following scenario of the use case “Enroll a customer for a seminar”:

- The customer “Smith” is not in the customer list, so the user clicks on “Create customer”.
- The user enters the customer data and confirms with “OK”.
- The system creates customer and account instances.
- The user creates a seminar with the title “UML”.
- The system creates a Seminar instance.
- The user selects a customer.
- The user selects a seminar.
- The user clicks “Enroll”, the system checks, if the customer can be enrolled for the seminar (OK=true)
- The user enters the enrollment data.
- The system creates Enrollment and AccountingEntry instances.

When the analyst executes this scenario for the first time, he will have to enter concrete data values. The system stores the scenario path through the state diagram, together with the user input information. If the states are represented by letters and the transitions by numbers, this scenario might look like shown in Figure 3.

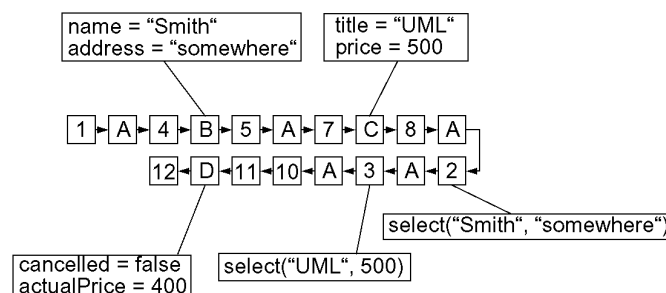


Figure 3. A concrete scenario for the use case “Enroll a customer for a seminar”.

When the scenario is executed a second time, the system can enter the data automatically. We have developed a technique so that a tool can map the concrete user input values scenarios even if the state diagram (and the resulting path through the state diagram) is changed due to requirement changes.

5. Introducing structure

Use cases can be connected via include and extend dependencies. These dependencies map to statechart dependencies as illustrated in Figure 4.

6. Summary

Our approach to requirements specification with scenarios and statecharts allows an analyst to build an integrated requirements model with informal and formal action specifications and hierarchical decomposition. By generating scenarios from statecharts, it avoids inconsistency and redundancy

problems, which are typical for scenario-based approaches. It explicitly supports changing requirements and allows interactive scenario animation.

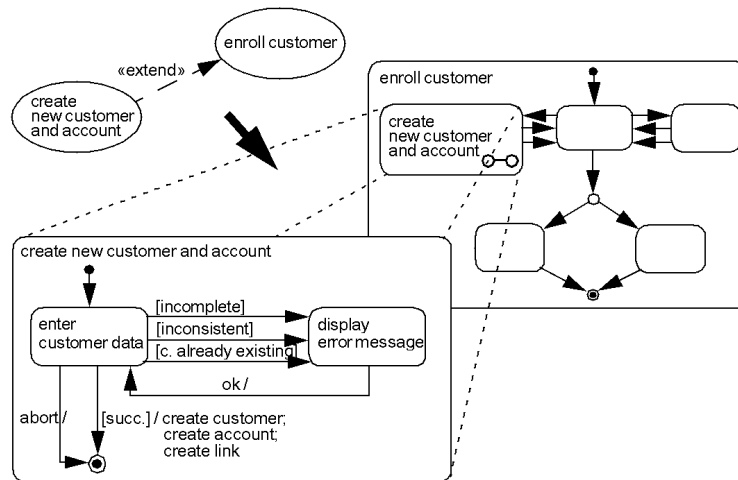


Figure 4. Mapping use case dependencies to statechart dependencies

7. References

- [1] F. Bordeleau, J.-P. Corriveau, and B. Selic. A Scenario-Based Approach to Hierarchical State Machine Design. Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 1998.
- [2] I. Jacobsen, M. Christerson, P. Jonsson, G. Övergaard: Object-Oriented Software Engineering. Addison Wesley/acm Press, Reading, Mass., 1992
- [3] M. Jarke. Scenarios for Modelling. Communications of the ACM, Volume 42, 1999.
- [4] M. Jarke. CREWS: Towards Systematic Usage of Scenarios, Use Cases and Scenes, Proc. 4. Internationale Tagung Wirtschaftsinformatik (WI99), Saarbrücken, 469-486 (1999)
- [5] M. Jarke, K. Pohl et al. Scenario Use in European Software Organizations - Results from Site Visits and Questionnaires. CREWS-Report 97-10, Namur, Aachen 1997.
- [6] Kennedy Carter Ltd. Supporting Model Driven Architecture with eExecutable UML, <http://www.kc.com/MDA/xuml.html>, 2001.
- [7] K. Koskimies, T. Systä, J. Tuomi, and T. Männistö. Automated Support for Modeling OO Software. IEEE Software, 15(1): 87-94, 1998.
- [8] I. Krüger. Notational and Methodical Issues in Forward Engineering with MSCs. Proceedings of OOPSLA 2000 Workshop: Scenario based round-trip engineering, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000.
- [9] S. Leue, L. Mehrmann, and M. Rezaei. Synthesizing ROOM Models From Message Sequence Charts Specifications. Proc. 13th IEEE Conf. on Automated Software Engineering, 1998.
- [10] E. Mäkinen and T. Systä. An Interactive Approach for Synthesizing UML Statechart Diagrams from Sequence Diagrams. Proceedings of OOPSLA 2000 Workshop: Scenario based round-trip engineering, October 2000.
- [11] T. Systä. Incremental Construction of Dynamic Models for Object-Oriented Software Systems. Journal of Object-Oriented Programming Vol. 13 No. 5 pp. 18-27, September 2000.
- [12] J. Whittle and J. Schumann. Generating Statechart Designs From Scenarios. Proceedings of OOPSLA 2000 Workshop: Scenario based round-trip engineering, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000.
- [13] I. Wilkie et al. UML Action Specification Language (ASL) Reference Guide, <http://www.kc.com/download/index.html>, 2001.
- [14] <http://www.umlactionsemantics.org>