Summary Report of the OOPSLA 2000 Workshop on Scenario-Based Round-Trip Engineering

Tarja Systä
Tampere University of Technology
Software Systems Laboratory
P.O. Box 553
33101 Tampere, Finland
tsysta@cs.tut.fi

Rudolf K. Keller
Université de Montréal
Département IRO
C.P. 6128, succ. Centre-ville
Montreal (Quebec) H3C 3J7, Canada
keller@iro.umontreal.ca

Kai Koskimies
Tampere University of Technology
Software Systems Laboratory
P.O. Box 553
33101 Tampere, Finland
kk@cs.tut.fi

CONTENT

This report summarizes the Workshop on Scenario-Based Round-Trip Engineering held in Minneapolis, Minnesota, USA, on October 16, 2000, in conjunction with the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2000). The workshop consisted of a keynote and seven presentations, which were organized into three sessions: From Interaction Diagrams to State Machines, Forward Engineering, and Reverse Engineering. Altogether nine position papers were accepted. The workshop web page, including the papers, the presentations, and the electronic version of this report, can be found at <http://www.cs.uta.fi/~cstasy/oopsla2000/workshop.html>.

BACKGROUND

Behavioral (or dynamic) modeling plays an important role in object-oriented (OO) software engineering. In modern OO modeling notations, behavioral modeling is supported by dedicated diagrams. The Unified Modeling Language (UML) [24, 25], for instance, provides scenario-based interaction diagrams, that is, sequence diagrams and collaboration diagrams. Use cases can conveniently be refined using interaction diagrams, which in turn aid in recognizing the operations and associations of classes and in specifying the dynamic behavior of objects as statecharts. Various kinds of tool support automating parts of this process have been suggested.

The usefulness and importance of dynamic modeling is not limited to OO software construction. Behavioral modeling is also essential for understanding existing OO systems. In dynamic reverse engineering, for instance, the object interactions are typically modeled using scenario-based approaches. Scenarios can also be used for analyzing interactions in concurrent and real-time systems and for synthesizing behavioral specifications. To make them applicable for modeling object interactions, basic Message Sequence Charts (MSCs) [9] have been extended in various ways to increase their expressive power. Some of the extensions have been adopted in variations of MSCs. For instance, UML sequence diagrams have been designed to accommodate algorithmic structures.

WORKSHOP GOALS AND FORMAT

The workshop participants were invited to cover practical and theoretical issues of the usage of scenarios in forward engineering, reverse engineering, and round-trip engineering. In the course of the workshop, various new scenario-based approaches were introduced. In addition, the limitations and shortcomings of current approaches were discussed, and possible directions for future research in the domain were outlined. Areas of interest of this one-day event were the following:

- extensions to the basic Message Sequence Chart (MSC) notation from both the forward or the reverse engineering point of view
- synthesis methods based on scenario specifications
- using UML behavioral models for reverse engineering OO software systems
- modeling sequential object interactions in software construction
- understanding the behavior of OO systems
- dynamic modeling in round-trip engineering and re-engineering
- modeling and analyzing interactions between concurrent scenarios
- tool support for the above activities

Reflecting these goals, the workshop started with a keynote address. The rest of the workshop was organized into three sessions based on the topics of the position papers: *From Interaction Diagrams to State Machines*, *Forward Engineering*, and *Reverse Engineering*. The sessions were highly interactive and included a thorough discussion of the respective session theme.

KEYNOTE ADDRESS

Øystein Haugen (Ericsson NorARC), the *rapporteur* for the Z.120 standard [9], gave a keynote that consisted of two parts. The first part focused on the role of scenarios in dialectic software development, and the second part introduced the MSC-2000 notation.

In the first part of his presentation, Haugen discussed the role that scenarios should play during a dialectic software development process. The preciseness and the degree of details of scenarios used for describing and documenting requirements should be systematically increased along the process. When making a scenario more precise, the scenario should be expressed using a more formal language. Furthermore, new aspects and properties should be gradually added. Several models are needed for capturing all the aspects of the behavior of the system to be designed. Even small systematic steps used for changing the models often result in conflicts (e.g., between a MSC and an SDL description) and therefore require constant verification. Haugen emphasized that such conflicts are an important and natural element of the process and should not be treated as errors. Harmonizing the models to remove conflicts should not be considered a bug fix, but rather a step forward in the dialectic development process.

In the second part of his keynote, Haugen introduced some concepts of the MSC-2000 standard and discussed the differences and extensions in respect to the MSC-96 standard. The main extensions in MSC-2000 include the following: means to add data with the data language of the user's choice, support for object-orientation, and time observations and time constraints. The support for object-orientation, for example, includes notational mechanisms for capturing inherited behavior and bears a close resemblance to the OO facets of SDL and to other OO languages such as Java. The idea of adding data to sequence diagrams raised both supporting and criticizing statements among the participants. It was argued that adding data to sequence diagrams provides a way to make the diagrams more precise. On the other hand, it was pointed out that a sequence diagram with a lot of data references and assignments becomes difficult to read and may distract from the original purpose of using MSCs.

SESSION I: FROM INTERACTION DIAGRAMS TO STATE MACHINES

In OO analysis and design (OOAD), dynamic modeling aims to describe the dynamic behavior of objects using variants of finite state machines. The UML variant of state machines is the so-called *statechart diagram*. A statechart diagram is a graph that represents a state machine. The semantics and notation used in the UML follow Harel's statecharts [7]. *Sequence diagrams* and *collaboration diagrams* are also used for behavioral modeling in UML based approaches. A sequence diagram shows the interaction over time but does not show relationships between objects other than the events belonging to the interaction. A collaboration diagram, in turn, does not show time as a separate graphical dimension. While sequence and collaboration diagrams are used for capturing object interactions, a statechart diagram can be used as a protocol specification, showing the legal order in which operations of an object may be invoked.

Automated support for constructing state machines from scenarios provides considerable help for the designer. Such support is provided by several algorithms and tools [17, 27, 29, 12]. However, the information represented in a standard sequence diagram is usually not complete enough for fully specifying the behavior of an object as a state machine. Typically, such synthesis algorithms generalize information given in sequence diagrams, that is, the resulting state machine will accept more paths through the modeled system than represented as sequence diagrams. Such generalization is usually desirable. However, in some cases a synthesized state machine might also accept unwanted or erroneous paths and thus be "overgeneralized". Applying the synthesis algorithm, for instance, to an incomplete set of sequence diagrams may result in an overgeneralized state machine that does not meet the user's intentions. Inaccuracies in the contents of the sequence diagrams can have similar effects. Since each sequence diagram is usually written in isolation, bringing many sequence diagrams together can easily cause inconsistencies. Sequence diagrams alone do not usually contain enough semantic information to enable the automatic detection of inaccuracies and inconsistencies. Thus, additional information is needed.

The problem of providing additional information (to that presented in sequence diagrams) is not properly solved in currently available techniques and tools. In this session, three position papers were presented addressing this problem.

Whittle [32] presented a synthesis algorithm, developed in joint work with Schumann [31], which supports the design process by generating statechart designs automatically from scenarios (expressed as UML sequence diagrams). The algorithm first takes a collection of sequence diagrams and generates automatically a flat state machine corresponding to the behavior in the sequence diagrams. The user can give additional information by expressing constraints on the diagrams, using the Object Constraint Language [21] (OCL). With pre and post conditions, expressed in OCL, the user can guide the algorithm to merge information read from several sequence diagrams in a desired way. The specifications should include the declaration of global *state variables*, where a state variable represents some important aspect of the system. Pre and post conditions should then include references to these variables. In the approach of

Whittle and Schumann, hierarchy and orthogonality (i.e., composite states) are included in the generated statechart diagram by taking into account information that is presented in the class diagram.

Mäkinen and Systä [18] proposed an interactive approach for synthesizing UML statechart diagrams from sequence diagrams. In this approach, the synthesis algorithm called MAS (Minimally Adequate Synthesizer) [19, 20] consults the user whenever needed to avoid undesired generalizations in the resulting statechart diagram. MAS models the synthesis process as a language inference problem and uses Angluin's [1] framework of minimally adequate teacher to infer the desired statechart diagram with the help of membership and equivalence queries. The algorithm can conclude the correct answer to most of the membership queries without consulting the teacher, i.e., the designer.

Being a minimally adequate teacher requires that the designer can answer two kinds of simple questions:

(1) she must decide whether a given behavior is possible in the system she is implementing (the membership queries),

(2) she must accept or reject the output statechart diagram, and moreover, if she rejects, a counterexample must be given (the equivalence queries).

The user can give both positive and negative counterexamples. Positive counterexamples define additional behavior that also should be included in the statechart diagram, while negative counterexamples define behavior that should be excluded.

A practical implementation of MAS is integrated to a real-world UML modeling tool, the Nokia TED [33]. TED is a multi-user software development environment that has been implemented at Nokia Research Center and is currently used at Nokia. It supports most of the UML diagram types.

The approaches presented by Whittle and Systä are clearly related. In the approach by Whittle and Schumann, additional information is provided by inserting OCL pre and post conditions to the sequence diagrams, while the approach by Mäkinen and Systä relies on consulting the user during the synthesis process. Whittle and Systä pointed out that a method that combines these approaches could be useful: in certain cases clear-cut conditions should be specified, whereas a more relaxed way to guide the algorithm during synthesis might be enough in other cases.

Bordeleau introduced a pattern-based method for constructing hierarchical state machines from scenarios. He claimed that whereas a flat state machine (possibly synthesized automatically) is suitable for exhaustive path analysis, hierarchical state machines allow for the clustering of different behavioral facets of a component (often referred to as *roles)* into distinct hierarchical states of this component.

The approach by Bordeleau and Corriveau separates different concerns in the behavior [3]. The approach goes from use cases [10] defined at the requirements level to a Use Case Map (UCM) [4] model, which is used as a high-level design model of the system. The UCM model mainly describes system scenarios with respect to their interactions and to high-level system structure. The UCMs are then refined into interaction diagrams such as MSCs or UML sequence diagrams. Finally, the detailed interaction diagrams are transformed to the definition of hierarchical state machines.

The generation of hierarchical state machines from interaction diagrams works in two steps. In the first step, the detailed scenario information contained in the interaction diagrams is used to produce *role state machines* on a per scenario basis. A role state machine shows one aspect of the behavior. In the second step, inter-scenario relationship information (captured in the UCM model) is used to integrate the role state machines of a component into the hierarchical state machine of that component.

Bordeleau and Corriveau suggest that the integration of role state machines be a pattern-driven task. They have presented a catalogue of patterns [3] that can be used to accomplish that task.

All three approaches presented in this session aim to solve the same two problems:

(1) how to construct state machines from scenarios, and

(2) how to support the user in providing additional information needed for constructing state machines that meet her intensions.

All three approaches offer different, yet complementary solutions to these problems. The workshop participants thus discussed the benefits of future collaborations.

SESSION II : FORWARD ENGINEERING

Three papers were categorized under the broad title of this session. Two of them, which were only loosely related, were presented.

Krüger discussed notational and methodological issues in forward engineering with MSCs [15]. This presentation thus dealt with similar issues as those presented in the keynote. Krüger further discussed the problem of synthesizing state machines from MSCs and once again pointed out that the MSC-96 notation [3] as such is not sufficient to accomplish this task properly.

Krüger claimed that only if the MSCs can express the relevant structural and behavioral properties of the system under development will they be accepted and considered

useful as a means for system specification beyond informal requirements documentation. He pointed out that the MSC-96 notation has limited capabilities to combine MSCs. For example, overlapping interactions, parallelism, concatenation, hierarchy, and control and data state indicators are not properly supported.

To be better suited for systematic use in the software development process, Krüger suggests the following behavioral interpretations for MSCs: *existential*, *universal*, and *exact* [13, 14]. Existential behavior can, but need not occur. It also may be interleaved with arbitrary additional interactions. Universal behavior, in turn, must always occur. Finally, exact interpretation fixes the system's behavior to match precisely what the MSC specifies.

Selonen and Systä introduced a scenario-based approach for synthesizing annotated class diagrams from sequence diagram in the UML [28]. Their approach provides additional support for the designer in an early phase of the software design process, helping her to quickly check and view the current state of the design.

After modeling examples of interactions, the designer should add the information implied by the sequence diagrams to the static model (class diagrams), or check that the static model conforms to the sequence diagrams. To achieve complete models, several iterations are typically needed: the models are gradually refined from rough sketches to specifications. Class diagrams are often annotated with pseudocode descriptions of operation implementations, attached as UML comments to class symbols. Such descriptions are frequently used, for example, in design pattern documentations [6].

The position paper by Selonen and Systä details the transformation operation that synthesizes a class diagram from a set of sequence diagrams. In addition, operation descriptions are synthesized from the same set of sequence diagrams and visualized as state machines. The state machines, in turn, are used as an input for a pseudocode generator, which outputs a sketch of the operation body in a textual format. Finally, the generated class diagram can be annotated with the pseudocode presentations: a UML note that contains the pseudocode is attached to the corresponding operation. The mechanisms presented in the paper are primarily to be used during relatively early phases of software construction (analysis and design phases). Selonen and Systä are not aiming at producing complete models or executable code. Instead, they believe that a relaxed, pragmatic approach for modeling can benefit designers at early and intermediate phases of software development.

The third position paper about validating conceptual models by animation in a scenario-based approach [26] was not presented in the workshop. Sánchez *et al.* have developed a formal approach to specify conceptual models following an OO approach. This approach is called OASIS (Open and Active Specification of Information Systems) [16]. In their position paper, Sánchez *et al.* introduce OCA (OASIS Concurrency and Animation), a quality software animation module of OASIS specifications. For validating OASIS specifications, two techniques are used:

(1) requirements elicitation based on captured real world examples (represented using the UML)

(2) requirements validation by animating formal specifications

The diverse topics discussed in the position papers indicate that scenarios are applicable in various ways during the forward engineering process. This session contained only three position papers and could thus cover only a small part of the possible applications. Unlike in session I, the position papers in this session did not have much in common.

SESSION III : REVERSE ENGINEERING

Scenarios are typically used in dynamic reverse engineering for capturing run-time object interactions in the subject software system [11, 8, 30]. In two position papers presented in this session, scenarios are used for a different purpose: a certain procedure in the reverse engineering process (static or dynamic) is captured in a scenario.

Demeyer introduced a reverse engineering pattern language [5] which explains how to reverse engineer an OO software system. The pattern language for the reverse engineering process has been developed and applied during the FAMOOS project, which had the explicit goal to produce a set of re-engineering techniques and tools to support the development of OO frameworks. Many of the patterns were applied on software systems provided by industrial partners of the project (i.e., Nokia and Daimler-Chrysler). These systems ranged from 50.000 lines of C++ up until 2,5 million lines of Ada.

A reverse engineering pattern has a name, a definition of problems, a solution, a rationale, a list of known uses, and references to related patterns. Demeyer introduced an example reverse engineering pattern "Interview During Demo", which aims at obtaining an initial feeling for the functionality of a subject software by seeing a demo and interviewing the person giving the demo.

Demeyer elaborated on reverse engineering to program understanding. He pointed out that the reverse engineering process can and should be supported by appropriate tools, but that the actual understanding can only be obtained by humans. Demeyer sees reverse engineering as an iterative speculation process, supported by tools that are simple enough.

Peltonen and Selonen introduce visual tool support for scenario-based software engineering [22]. Many OO CASE tools support textual scripting mechanisms for extending and customizing the tool. Peltonen and Selonen point out that because the tools themselves are graphical, there is a paradigm shift from a graphical environment to a textual language. Even for a simple script, the user has to learn a new language.

Peltonen and Selonen discuss a method for customizing and extending a CASE tool supporting the UML using a visual scripting mechanism. They introduce a visual scripting language VISIOME [23], which is based on the UML activity diagram notation itself. VISIOME is currently being implemented in the Nokia TED tool, which is a UML-centered CASE tool that can be extended through a COM-based API.

In their position paper, Peltonen and Selonen discuss various ways (scenarios) to use VISIOME scripts to achieve certain software engineering tasks. As an example, they define a script that is able to synthesize annotated class diagrams from sequence diagrams in the UML. The rationale behind the synthesis process is discussed by Selonen and Systä in [28].

Many scenarios aiming at achieving specific software engineering tasks are typically repeated several times during the development of a software system (or when designing another software system). Thus, recording and storing such scenarios is useful. VISIOME scripts provide a way to support this.

The third position paper by Bojić and Velašević discusses a use-case and test driven approach to round-trip engineering a complete UML model, as prescribed by required element of the Unified Process Template (RUP) [2]. This position paper was not presented in the workshop. In their paper, Bojić and Velašević introduce a method called UCRA (Use-case Driven Model Recovery by means of Formal Concept Analysis), which utilizes dynamic analysis of an application for a selected set of test cases.

DISCUSSION AND FUTURE PLANS

The approaches discussed in the workshop were mostly based on academic research. Haugen argued that to evaluate the real value of the approaches, they need to be tested and applied on real world software engineering tasks. The other workshop participants agreed on this argument. It was further remarked that the co-operation between academia and industry is important, if not essential, for further development of the approaches.

The workshop achieved its goal to provide a forum for interactive discussions on the usage of scenarios in forward engineering and reverse engineering. Especially in session I, the topics of the position papers were closely related. On several occasions during the one-day workshop, items for a common research agenda in scenario-based round-trip engineering were debated, and plans for future collaborations were discussed. Moreover, it was decided that a follow-up workshop on the topic should be organized within the next two years.

REFERENCES

1. Angluin D.: Learning regular sets from queries and counterexamples, Inf. Comput. 75, 1987, 87-106.
2. Bojić D. and Velašević D.: UCRA Approach to Scenario-based Round-trip Engineering, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 51-56.
3. Bordeleau F. and Corriveau J.-P.: From Scenarios to Hierarchical State Machines: A Pattern-Based Approach, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 13-18.
4. Buhr R. and Casselman R.: *Use Case Maps for Object-Oriented Systems*, Prentice Hall, 1996.
5. Demeyer S., Ducasse S., and Nierstrasz O.: Interview during Demo: a Sample Reverse Engineering Pattern, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 45-50.
6. Gamma E., Helm R., Johnson R., and Vlissides J.: *Design Patterns – Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.
7. Harel D.: Statecharts: A visual formalism for complex systems, *Science of Computer Programming*, 8, 1987, 231-274.
8. Jerding D. and Rugaber S., Using Visualization for Architectural Localization and Extraction, in *Proc of WCRE'97*, Amsterdam, The Netherlands, 1997, 56-65.
9. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), ITU-T, Geneva, 1996.
10. Jacobson I., Booch G., and Rumbaugh J.: *The Unified Software Development Process*, Addison-Wesley, 1999.
11. De Pauw W., Helm R., Kimelman J., and Vlissides J.: Visualizing the behavior of object-oriented systems, in *Proc of OOPSLA'93*, ACM Press, October 1993, 326-337.

12. Koskimies K., Männistö T., Systä T., Tuomi J.: Automated support for modeling OO software, IEEE Softw, 15, 1, 1998, 87-94.

13. Krüger I: Distributed System Design with Message sequence Charts, PhD thesis, Technische Universität München, 2000. (http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2000/krueger.html)

14. Krüger I, Grosu R., Scholz P., and Broy M.: From MSCs to Statechart, in *DIPES'98*, Kluwer, 1999, 61-71.

15. Krüger I.: Notational and Methodical Issues in Forward Engineering with MSCs, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 19-25.

16. Letelier P., Ramos I., Sánchez P., and Pastor O.: Object- oriented Conceptual Modeling using a Formal Approach. Servicio de Publicaciones Universidad Politécnica de Valencia, SPUPV-98.4011, 1998. (in Spanish).

17. Leue S., Mehrmann L., Rezai M.: Synthesizing software architecture descriptions from message sequence chart specification, In *Proc. of the 13th IEEE International Conference on Automated Software Engineering (ASE98)*, Honolulu, USA, 1998, 192-195.

18. Mäkinen E. and Systä T.: An Interactive approach for synthesizing UML statechart diagrams from Sequence Diagrams, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 7-12.

19. 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)

20. 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)

21. Object Management Group, Unified modeling language specification version 1.3, Available from Rational Software Corporation, Cupertino, CA; June 1999.

22. Peltonen J. and Selonen P.: Visual tool support for scenario-based software engineering, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 38-44.

23. Peltonen J.: Visual Scripting Language for UML-based CASE tools, in *Proc. of ICSSEA 2000*, Volume 3, Paris, France, 2000.

24. Rational Software Corporation: *The Unified Modeling Language Notation Guide v.1.*, http://www.rational.com, 2000.

25. Rumbaugh J., Jacobson I., and Booch G.: *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

26. Sánchez P., Letelier P., and Ramos I.: Validation of Conceptual Models by Animation in a Scenario-based Approach, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 32-37.

27. Schönberger S., Keller R., Khriss I.: Algorithmic Support for Model Transformation in Object-Oriented Software Development, In *Concurrency Practice & Experience (Theory and Practice of Object Systems, TAPOS)*, 13, John Wiley & Sons, 2001. To appear.

28. Selonen P. and Systä T.: Scenario-based Synthesis of Annotated Class Diagrams in UML, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 26-31.

29. Somé S., Dssouli R., Vaucher J.: From scenarios to automata: building specifications from users requirements, In *Proc. of APSEC'95*, Brisbane, Australia, 1995, 48-57.

30. Systä, T.: Static and Dynamic Reverse Engineering Techniques for Java Software Systems. Ph. D. Dissertation, Dept. of Computer and Information Sciences, University of Tampere, May 2000.

31. Whittle J. and Schumann J.: Generating Statechart Designs From Scenarios, in *Proc. of ICSE 2000*, Limerick, Ireland, June 2000, 314-323.

32. Whittle J.: On the Relationship Between UML Sequence Diagrams and State Diagrams, in *Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering*, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 1-6.

33. Wikman J.: Evolution of a Distributed Repository-Based Architecture. (http://www.ide.hk-.se/~bosch/NOSA98/JohanWikman.pdf), 1998.