

Simulation-Driven Creation, Validation and Evolution of Behavioral Requirements Models

Martin Glinz Christian Seybold¹ Silvio Meier

Institut für Informatik
Universität Zürich
Binzmühlestrasse 14
CH-8050 Zurich, Switzerland
{glinz | smeier}@ifi.unizh.ch
cseybold@gmx.ch

Abstract: Requirements models for large systems cannot be developed in a single step; they evolve in a sequence of iterations. We have developed a simulation-driven process that supports iterative, evolutionary modeling of behavioral requirements. We start with modeling type scenarios (i.e. use cases) and simulate these interactively. The simulation runs yield exemplary system behavior, which is documented in message sequence charts (MSCs). The modeler can then generalize this recorded partial behavior into statecharts. The resulting model is simulated again for validating that the modeled behavior matches the previously recorded behavior. The validated model is then used in the next incremental step for eliciting new, yet unspecified behavior by simulating new scenarios.

1 Introduction

Requirements models under development are typically incomplete and not completely formalized. In an iterative development process, modeling proceeds by progressively making models more complete and more formal. This process of model evolution stops when the model has reached the desired degree of formality and completeness (depending on risk, time and budget). In order to support such a process, three requirements must be met: we need (i) a modeling language that provides features for modeling intentional partial incompleteness and a variable degree of formality, (ii) a technique for early and frequent model validation, and (iii) guidance for systematically eliciting and evolving models towards more formality and more completeness.

The ADORA language [GBJ02] that has been developed in our research group satisfies the first requirement.

For early model validation, simulation would be a quite appropriate technique both for modelers and stakeholders: they can play with the model's dynamics by entering stimuli

¹ Christian Seybold is now with Zühlke Engineering (Switzerland)

and receiving system reactions. However, simulation, as well as other automated techniques (e.g. model checking) cannot be applied as long as we do not have complete and formal models. Therefore, we have developed an interactive simulation technique that allows to simulate incomplete and semi-formal models by inquiring missing information interactively from the expert who runs the simulation. The information provided by the expert is recorded so that regression simulation becomes possible [SMG05]. This technique satisfies the second requirement.

Our simulation technique also enables an iterative, outside-in development process for behavioral requirements models that starts with some external behavior specified by type-level user-system interaction scenarios (aka use cases) and progressively elicits and defines system behavior with simulations that are driven by playing through the scenarios [SMG04], [SMG06], which addresses the third requirement.

In this paper we summarize our approach, which is documented in [Se06], [SMG04], [SMG05] and [SMG06], and give an example.

2 Our approach

2.1 Simulation² of semi-formal models

We use ADORA [GBJ02] as a modeling language. Models in ADORA are composed of hierarchically structured abstract objects. Each object represents a state and may be further decomposed by other objects and by embedded statecharts. All objects and states together form one joint, hierarchical statechart. However, our approach is not restricted to ADORA. Alternatively one could use a sufficiently formalized UML profile. The advantage of ADORA is that it already has the features we need, whereas with UML, one first had to define and formalize a suitable subset.

The simulation engine of the ADORA tool simulates models specified in ADORA regardless of their degree of formality and completeness. In all situations where the simulator can't interpret the model because information is missing or the given information is not formal enough, the simulation engine lets the modeler interactively specify the desired behavior. That means, the simulation engine executes the specified system behavior, i.e. on occurring events it performs transitions between states and executes specified actions. As soon as an event appears that cannot be handled, the simulation is interrupted to allow the user to handle this event interactively: whether it shall be received at all, by which object and which actions shall be performed on this event. Afterwards, the simulation continues as usual. Fig. 1 shows a screenshot of the ADORA tool during an interactive simulation session. The simulation engine is currently executing the Input Value scenario in a partial model of a calculator. As the model does not specify whether the Add or the Subtract scenario shall be executed next, the simulation engine lets the mod-

² When we talk about simulation, we mean an event-driven, discrete simulation. We do not consider real-time or continuous simulations.

eler interactively select an alternative. More details on our interactive simulation engine can be found in [SMG05] and [Se06].

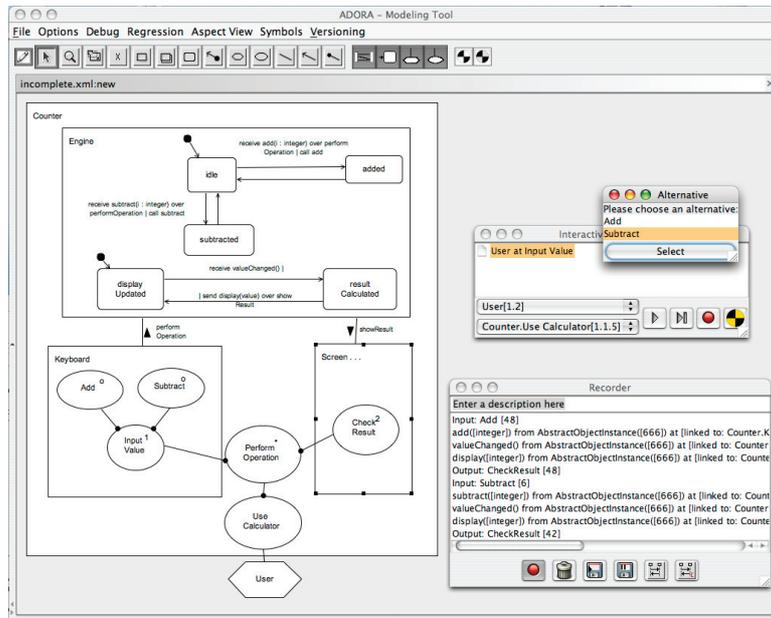


Figure 1: A screenshot of the ADORA tool during an interactive simulation session

The events driving the simulation are generated by playing through the scenarios of the model. Scenarios in ADORA describe the communication protocol between the modeled system and the actors in the context of the system, i.e. the behavior as it is seen by the external actors. Note that scenarios in ADORA are *type scenarios*, i.e. use cases in UML terminology. In ADORA, scenarios are modeled with so-called scenariocharts, a notation that is derived from Jackson's JSP diagrams [Ja75] and is capable of modeling scenario decomposition in terms of sequence, alternative, iteration and parallelism of sub-scenarios [GI95], [GBJ02]; see Fig. 2.

For each modeled actor, the user simulating the model may create an instance, thereby launching the traversal of the connected scenariochart. The simulation stops at leaf nodes allowing the user simulating the model to enter stimuli or receive system reactions. The graphical interface to enter stimuli is automatically built from the transform expressions specified at the leaf nodes of the scenariocharts.

As every small modification of a requirements model can unintentionally destroy required properties that held in the model prior to the modification, a model should be revalidated after each incremental step. In our approach, revalidation is done by *regression simulation*, i.e. automatically re-executing previously recorded simulation runs. While the principal idea is fairly obvious, some major problems have to be solved to

make regression simulation work in our context. In particular, regression simulations have to run automatically regardless of the amount of interactivity that was required to generate the original simulation runs. We achieve this by letting the model drive the regression simulation, but automatically resort to the interactively recorded behavior in those cases where the simulation engine does not get enough information from the model during a regression simulation run.

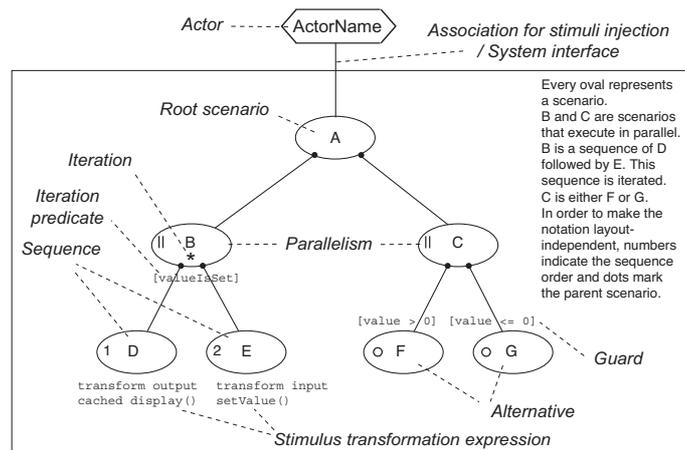


Figure 2: A scenariochart and its interpretation

For every completed simulation run, the ADORA tool can automatically generate a Message Sequence Chart [MSC96]. These MSCs can be used for analyzing simulations (in particular, when a regression simulation fails [SMG05]) and for driving the incremental modeling process (see next sub-section).

2.2 Iterative development of behavior models

Models of system behavior can be created in two ways: (a) the modeler begins with chunks of exemplary behavior which are then synthesized and generalized, (b) she or he creates a generalized model of behavior (i.e. a type level model that covers all potential cases) from the beginning and validates it by examining how the model behaves when playing through specific, exemplary cases.

Exemplary behavior is cognitively easier to develop and also better suited for discussing models with stakeholders for validation purposes. So it is a good starting point for system modeling. However, as exemplary behavior just gives an outline of certain situations, we eventually need type-level behavior models. On the other hand, creating type-level models from scratch is difficult and requires considerable effort and expertise.

An obvious solution would be generating type-level models automatically from a collection of exemplary, instance-level models (e.g. synthesizing statecharts from MSCs

[WS00], [KGSB99]) However, these approaches have drawbacks. Firstly, the generated statecharts are typically hard to read for humans and, hence, difficult to validate. Secondly, when we interpret MSCs existentially³, there is typically more than one statechart that describes the behavior specified by a given set of MSCs. So there is no unambiguous mapping from MSCs to statecharts (unless one introduces a special syntax and semantics for MSCs, as done in [KGSB99]).

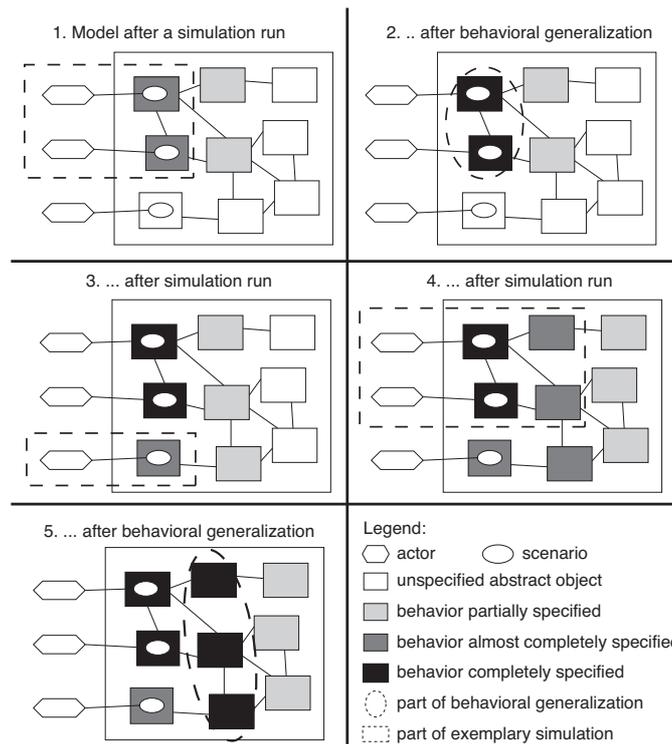


Figure 3: Incrementally building a behavior model

Therefore we decided to pursue a manual, tool-guided approach which combines exploration of exemplary behavior with systematic (but manual) construction of statecharts. Starting from scenarios describing external behavior as seen by actors, we derive exemplary behavior by running simulations on a void or incomplete behavior model and define the expected behavior interactively. For every such simulation run, the ADORA tool generates a message sequence chart. These MSCs are used to guide and inform a manual process of constructing statecharts that exhibit the expected behavior when executed with the given scenarios. Regression simulation is used to validate the created statecharts systematically.

³ Existentially means that the specified behavior must be possible, but other behavior may also occur. This is the usual MSC semantics.

Having specified parts of the desired behavior with statecharts, subsequent simulation runs become easier: the modeled behavior is integrated into the simulation, thus making it more precise and requiring less interactivity during simulation runs.

When used this way, the two modeling modes, viz. recording exemplary system behavior by playing through scenarios and modeling system behavior with statecharts, i.e. on a type level, stimulate each other and drive the model development towards more complete and more formal specifications. Fig. 3 illustrates how an alternating application of simulation runs and behavioral generalization/synthesis contributes to the evolution of a system model.

3 Example

We illustrate our approach with an example (taken from [SMG06] and [Se06]). As an example application, we refer to the specification of a control system for car doors given in [HP02]. A first version of the model is given in Fig. 4. Some scenarios have been modeled already whereas the system is yet unspecified except that it has been partitioned into the components Seat Adjustment, Door Locking and User Management.

In a first step, we decide to focus on the seat positioning via the user management switches. There are four switches inside the car to recall four different seat positions (for different drivers). A seat may be adjusted in five different ways: seat height in the front and back, angle of the back, seat distance to the wheel, and tightness of the casing. While being seated, seat adjustments must be done in a comfortable way, i.e. the relaxing adjustments must happen before constrictions take place (in order not to trap the driver), and not more than two movements may be done at the same time. However, when unlocking the car with a radio transmitter (there are two different ones for two drivers), the seat position shall be adjusted as fast as possible to be ready before the driver enters the car. The switches inside the car are connected to the door control system via interface S1, the radio transmitter uses the CAN-bus for communication.

In our example, we start with a simulation of a typical sequence of interactions how the seat adjustment could take place, executing the CAN Bus Control scenario first (i.e. a user unlocks the car) and then the User Management Scenario (i.e. a user presses seat adjustment switches). At that time, there is no behavior specified in the system. All objects taking part in the simulation (CAN, S1, User Management and Seat Adjustment) are being played by the modeler. This means that for each incoming event, the modeler specifies the corresponding actions that should take place. The result of this simulation run is shown in Fig. 5. As we are currently concentrating on the user management, we are not interested in the actual seat positioning. That is why we do not continue to handle the messages in the Seat Adjustment object (marked with a cross in Fig. 5). This may be the focus of further simulations. Therewith, we have outlined some exemplary behavior for the involved objects. We could either proceed by recording more simulation runs to enrich the system with exemplary behavior or we continue with the generalization of the existing behavior.

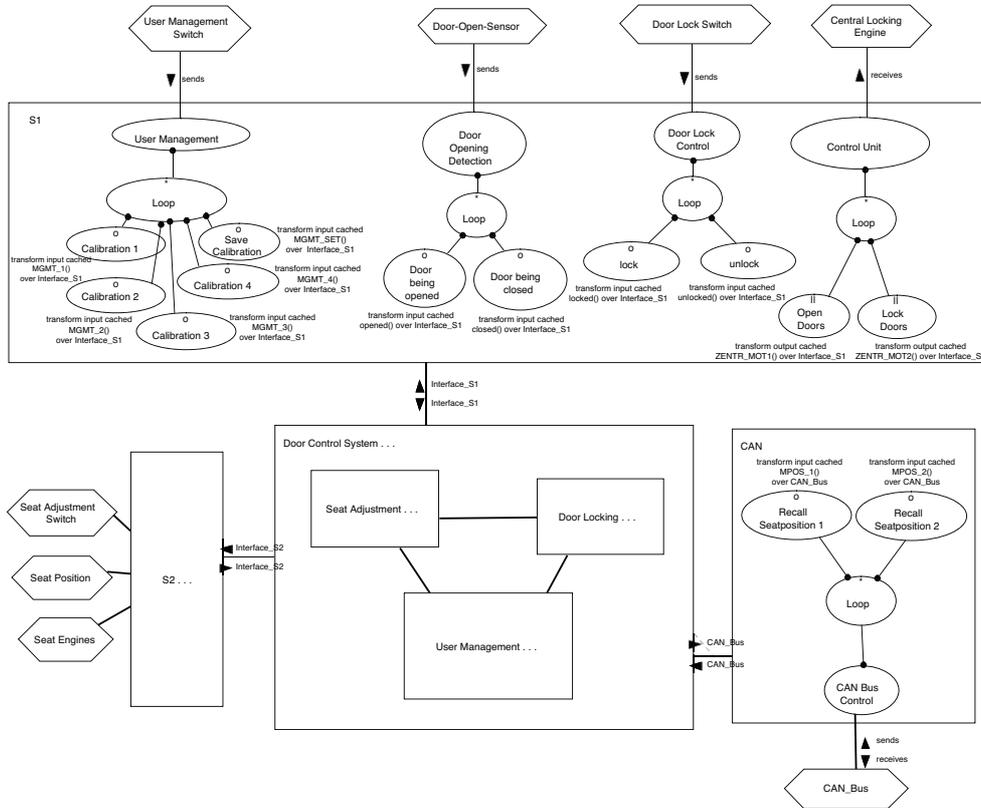


Figure 4: Initial model of a car door locking system

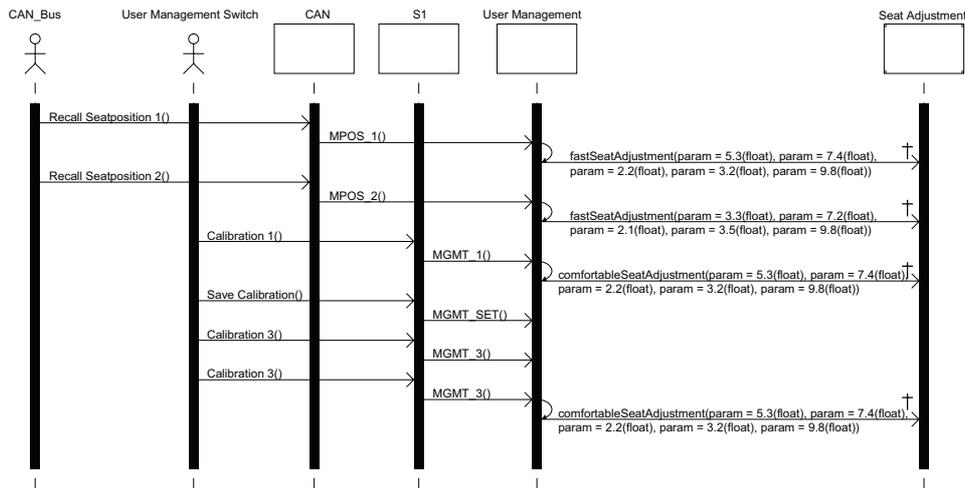


Figure 5: Trace of an exemplary play-through of the user management seat adjustment scenarios (automatically generated by the ADORA tool)

We have done the latter for the User Management object in Fig. 6. We selected the User Management lifeline from the MSC in Fig. 5 and created states and state transitions such that all incoming and outgoing messages are handled. This example also illustrates why we prefer a manual process: we cannot infer automatically whether the order of messages in the MSC matters or not. For modeling an adequate statechart, we hence need additional knowledge (domain knowledge or information elicited from stakeholders). In our case we found that the order of messages does not matter, leading to the statechart given in Fig. 6. Up to now, this statechart can exactly handle the recorded sequence chart, nothing else. However, for all following simulations, the specified behavior does not need to be played by the user any more. It will be taken from the statechart instead. Only new behavior must be played by the user.

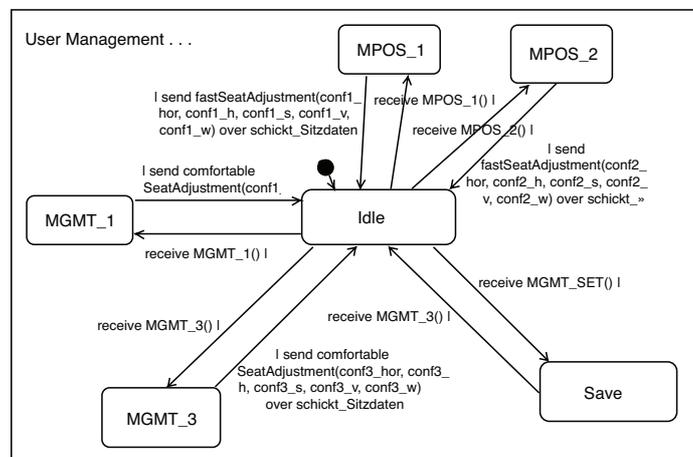


Figure 6: Generalized behavior of sequence chart in Fig. 5 in object User Management

This allows the modeler to focus on the specification of new behavior, for example enrich the behavior of the User Management Object. Simulations and behavioral generalizations can alternately take place, driving the model evolution until the model eventually exhibits the desired behavior.

4 Related Work

To the best of our knowledge, there is only one approach which is closely related to ours: Harel et al. developed the Play-Engine [HM03] that allows to play and test behavior of an incomplete component interactively via a prototypically built user interface. Harel focuses on the interface being developed whereas we are focusing the model being developed. Our main focus lies on adequate support of the requirements engineer in the modeling process. The model can be executed in any state of completeness to drive the further development of the model.

The idea of building models from exemplary behavior follows a rather old idea which was introduced by Shapiro for synthesizing logic programs from examples [Sh83].

Simulation as a means of validating a model is not a replacement for model checking approaches (e.g. [Ho97]). Simulation is used earlier in the process to validate the model and drive the further development of the model when it is not yet formal and complete enough to allow model checking.

5 Conclusions

In this paper, we have summarized and demonstrated our approach to simulation-driven modeling, validation and evolution of requirements models, using the ADORA language and tool.

The modeling and simulation capabilities described in this paper have been implemented in the ADORA prototype tool. This is a standalone system implemented in Java. Currently, the ADORA tool is being re-implemented as an Eclipse plug-in, which allows us to re-use a lot of existing model editor features. The re-implementation of the basic tool functionality has been completed. However, the re-implementation of the simulation engine has yet to be done. Our future work will concentrate on the incorporation of aspect modeling into ADORA and on further enhancing the modeling capabilities of both the language and the tool.

References

- [GI95] M. Glinz. An Integrated Formal Model of Scenarios Based on Statecharts. In W. Schäfer and P. Botella, editors, *Proceedings of the Fifth European Software Engineering Conference*. Lecture Notes in Computer Science Vol. 989, Springer, 1995. 254-271.
- [GBJ02] M. Glinz, S. Berner, and S. Joos. Object-Oriented Modeling with ADORA. *Information Systems* 27(6):425-444, 2002.
- [HM03] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer, New York, 2003.
- [Ho97] G. J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering* 23(5):279-295, 1997.
- [HP02] F. Houdek and B. Paech. *Das Türsteuergerät – eine Beispielspezifikation* [The Car Door Control System – An Example Specification (in German)]. Technical report, Fraunhofer Institut für Experimentelles Software Engineering, Kaiserslautern, 2002.
- [Ja75] M. Jackson. *Principles of Program Design*. Academic Press, New York, 1975.
- [KGSB99] I. Krüger, R. Grosu, P. Scholz, M. Broy. From MSCs to Statecharts. Proceedings of the IFIP WG10.3/WG10.5 International Workshop on Distributed and Parallel Embedded Systems, Schloss Eringerfeld, Germany, 1999. 61 - 71.
- [MSC96] Message sequence charts (MSC). *ITU-TS Recommendation Z.120*, 1996.
- [Se06] C. Seybold. *Simulation teilformaler Anforderungsmodelle* [Simulation of semi-formal requirements models (in German)] PhD Thesis, University of Zurich, 2006.
- [Sh83] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.

- [SMG04] C. Seybold, S. Meier, and M. Glinz. Evolution of Requirements Models by Simulation. *Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE '04)*, Kyoto, Japan, 2004. 43-48.
- [SMG05] C. Seybold, S. Meier, and M. Glinz. Simulation-Based Validation and Defect Localization for Evolving, Semi-Formal Requirements Models. *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC 2005)*, Taipei, Taiwan, 2005. 408-417.
- [SMG06] C. Seybold, S. Meier, M. Glinz. Scenario-Driven Modeling and Validation of Requirements Models. *Proceedings of the 5th ICSE International Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM 2006)*. Shanghai, China, May 2006. 83-89.
- [WS00] J. Whittle and J. Schumann. Generating Statechart Designs From Scenarios. *Proceedings of the 24th International Conference on Software Engineering (ICSE 2000)*, Limerick, Ireland, 2000. 314-323.