Innovate
Telelogic
trends in systems and software development
Issue #02

Rapid Prototyping with Constructive Scenarios

Nikolai N. Mansurov, KLOCwork Solutions

<Send this article>

The value of good requirements

"The hardest single part of building a software system is deciding precisely what to build… Therefore, the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements".

F. Brooks, "No silver bullet: Essence and Accidents of Software Engineering", IEEE Computer, (20), 4 April 1987, pp. 10-19

Defining and validating requirements early in the software development life cycle is one of the keys to a successful time-to-market strategy in the software industry. Proper requirements management helps development teams build software systems right from the start, avoiding costly redesign and rework late in the development cycle. In order to validate the requirements one can look for the feedback from the customers and other project stakeholders, or use some formal techniques.

This article outlines a tool-supported methodology in which requirements are captured in an intuitive and manageable format, which can automatically be transformed into effective visual prototypes as well as formal validation model. The suggested notation is also scaleable enough to allow refinement from requirements to design and to implementation in order to deliver productivity gains at later stages in the software development cycle.

Early prototyping can help you achieve that much-needed feedback on requirements, especially when the prototype is visual. But developing a prototype that collects customer feedback effectively can take time. A prototype is not the most effective form of managing and changing requirements, but it is necessary to capture requirements in some sort of a formal notation in order to apply formal validation techniques. This type of notation requires advanced skills and significant effort. Moreover, the three goals - ease of management and change, visual prototyping and formal validation - seem to be rather incompatible. Or are they?

Using scenarios to create prototypes

Scenarios are well recognized as useful ways to define requirements for real-time embedded software. Scenarios describe sequences of events in time and can be used to illustrate certain behavior in a way that is easy to understand and that can be discussed with non-technical stakeholders. Scenarios are also making their way into several other aspects of software development. For example, scenarios are great for visualizing traces, especially for systems with concurrency. Scenarios are used to represent validation traces. There are also similarities between scenarios and test cases. However, not all scenario techniques are equal:

Illustrative scenarios tell stories about how users interact with a system, or how components of the system interact with each other. While these scenarios are easy to understand, the do not formally relate requirements to the system design and implementation. There is little guarantee that the system that gets built will really map to the one envisioned in these scenarios.

Analytical scenarios, on the other hand, use a formal scenario notation, such as Message Sequence Charts (MSC), which are standardized by ITU-T (International Telecommunications Union). The meaning of a scenario-based specification is precise, however its relation to the implementation is still vague. MSCs can describe multiple situations, for example complete or partial intended behavior, or even undesired behavior. Analytical scenarios are great when it comes to reasoning precisely and unambiguously about behavior, but they do not map scenario events to an implementation.

Constructive scenarios, like illustrative scenarios, tell stories that easily elicit feedback. And, like analytic scenarios, they use a formal MSC notation. Constructive scenarios have certain characteristics that enhance their value. By definition, they describe sequences of events exactly, and provide a one-to-one mapping between scenario events and implementation events. Therefore constructive scenarios define not only the meaning of the specification itself, but also the behavior of the prototype implementation.

- Constructive scenarios let you model complete behavior, not just a set of illustrations. Basic MSCs allow you to describe simple sequences of events while HMSCs (Hierarchical MSCs, also an ITU-T standard) allow you to describe compositions of simple scenarios including alternatives and loops. Constructive scenarios also include means for describing flows of data through scenarios, data-dependent behavior as well as user interface details.
- Constructive scenarios can be transformed into an executable prototype or a formal validation model.
- Constructive scenarios constitute a single notation that represents requirements, traces, validations and tests.

Using constructive scenarios for simulation...

Constructive scenarios can be automatically transformed into executable code for a particular run-time environment. The KLOCwork Synthesizer from KLOCwork contains a synthesis algorithm that performs this transformation. The KLOCwork synthesis algorithm produces a collection of communicating event automata (state machines that support the events for a single scenario instance). A code generator produces the implementation of the automata in the selected programming language.

As users interact with the executable prototype, they simulate the original specification. Results of the simulation can be presented in three ways: in a text log as events are executed, highlighted step-by-step in the original scenarios, or as a new scenario. In the later case the simulation trace is presented using the same notation as the requirements.

... for animation

Additionally, events can be related to the way they appear in the user interface thus co-simulating behavior and the user interface. Co-simulating scenarios and the user interface achieves animation of both, which is very important for getting feedback from the stakeholders (see Figure 1).

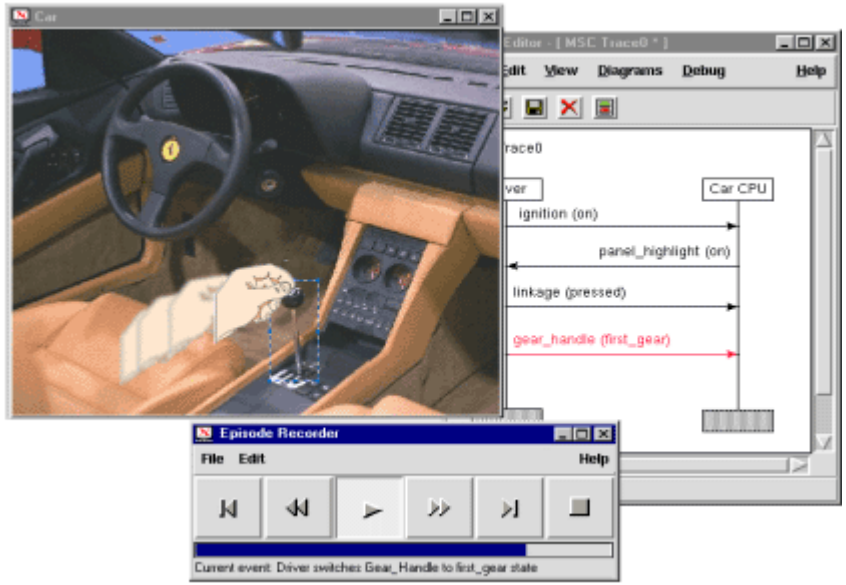

Figure 1. Co-simulation of the user interface and a scenario

...for validation

A special code generator can transform a constructive scenario into a formal validation model. For example, the KLOCwork Synthesizer can transform event automata into a formal model expressed in SDL (the ITU-T Specification and Description Language). The SDL model can then be validated using a tool that supports SDL, such as Telelogic Tau (see Figure 2). The Telelogic Tau "Validator" performs efficient state-space exploration of SDL models and automatically detects faults such as deadlocks, infinite loops, and communication buffer overflows. Faults in automatically synthesized validation models correlate directly with faults in the initial requirements specification. In our experience, the validation approach is capable of automatically detecting very complex faults, including feature interactions involving data. Faults are presented as traces in the same notation as the input constructive scenarios. Once again, you benefit from using just one notation.

... and for testing

Constructive scenarios can be used for testing in two ways. Firstly, requirements captured using constructive scenarios can be tested using test cases also expressed as constructive scenarios. The same approach described for validating constructive scenarios works for testing them. The Telelogic Tau Validator verifies a scenario by proving that a certain SDL model satisfies a certain scenario. In the case of constructive scenarios, the SDL model is automatically synthesized from the input requirements model using the synthesis algorithm. The requirements model itself serves as a test case for the implementation.

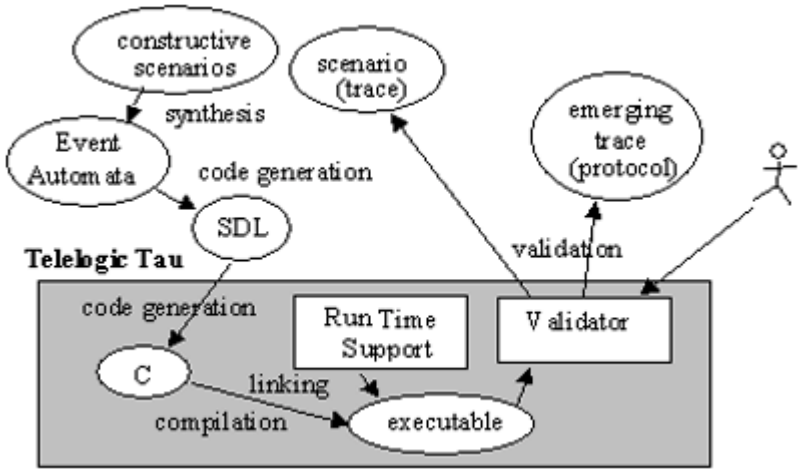

Figure 2. Using Telelogic Tau for validating constructive scenarios

The tools for the job

KLOCwork and Telelogic have formed a partnership to make the above-mentioned methodology available to a wider market. The resulting KLOCwork MSC to SDL Synthesizer is now offered as an integrated module in Telelogic Tau. The steps described below demonstrate how, using Telelogic and KLOCwork tools, you can start to take advantage of the benefits of using constructive scenarios for rapid prototyping:

1. Use Telelogic Tau to capture functional requirements as MSC and HMSC scenarios that focus on the interaction between your system and its environment.

2. Use the KLOCwork MSC to SDL Synthesizer to synthesize SDL models from the scenarios.

3. Run the synthesized models in the Telelogic Tau "Simulator" to simulate system behavior and get feedback from stakeholders.

4. If you discover problem areas, revise the MSCs or add more scenarios to solve the problems.

5. Rerun the KLOCwork MSC to SDL Synthesizer to create a SDL specification.

6. Repeat this process until you are satisfied with your prototype.

Conclusions

Constructive scenarios use a flexible and abstract notation for defining and managing requirements. A synthesis algorithm can transform constructive scenarios into a number of useful artifacts such as an executable prototype and a validation model. Further productivity improvement results from learning just one flexible notation that is used to represent requirements, traces, validation results and tests. The integrated rapid prototyping solution based on constructive scenarios is now available through the partnership between Telelogic and KLOCwork.

About the author

Nikolai N. Mansurov has a Ph.D. in Computer Science from Moscow State University and is currently Chief Scientist and Chief Architect of KLOCwork Solutions, Ottawa, Canada. Since 1994, he has been head of the Department for CASE Tools at the Institute for System Programming, Russian Academy of Sciences. Since 1987, he has been an Associate Professor at the Department for Computational Mathematics and Cybernetics at the Moscow State University. Nikolai participates in international standardization work for telecommunications languages and methodologies at ITU-T

SG17.