

Scenario-Based Evaluation of Abstract State Machines

Daniel Klünder and Stefan Kowalewski

Embedded Software Laboratory
RWTH Aachen University
Ahornstr. 55, 52074 Aachen, Germany
{kluender, kowalewski}@cs.rwth-aachen.de

Abstract. In this paper we suggest a scenario-based approach to analyzing non-functional properties of ASM ground models. It is based on well-founded evaluation methods from the software architecture field and enables a designer to reason about the quality features of his model, like e. g. maintainability or testability. A set of scenarios that represent a system's non-functional requirements and the evaluation of these with an ASM model can enable an improvement of the system's qualities by using according structuring principles.

1 Introduction

The Abstract State Machines (ASM) method offers a practical and scientifically well-founded approach to high-level system design and analysis by guiding the developer seamlessly from requirements capture to implementation [1]. So called *ground models* are used for a sufficiently precise, unambiguous, consistent, complete and minimal system description resulting from requirements elicitation [2]. Therefore the modeling of algorithmic content is separated from non-functional properties like e. g. maintainability, testability or performance, though the latter are of course relatable to the former [1]. Nevertheless a ground model is first and foremost a means to formal modelling of functional requirements that can be refined down to the implementation level.

A seemingly contrary approach has evolved in the software architecture field: it has long been recognized that the software architecture has a major impact on a system's non-functional properties like e. g. maintainability or testability [3]. A system's architecture is therefore seen as an optimization of this system's non-functional requirements with the functional ones being only the optimization constraints. In this paper we therefore suggest to add non-functional requirements formulated as *scenarios* to evaluate non-functional properties of ASM ground models.

The *Timed Abstract State Machine Language and Toolset* [4] incorporates timing behavior and resource consumption in ASMs. Our paper considers a broader set of non-functional features including those that are not quantifiable and their interference with the inner structure of an ASM model. There are

several scenario-based evaluation methods [5] that differ in their procedure and considered non-functional properties. In this paper we don't want to give an advantage to any of these methods but merely describe their basic impact on evaluating a system using ASMs. We therefore describe the relevance of non-functional requirements in section 2 and their modelling using *scenarios* in section 3.

2 Software Architecture

A software system comprises several structures, an insight that was first observed by Parnas [6]. It has long been recognized that these structures have a major impact on the non-functional properties of a system which are heavily influenced by a product's business goals [3].

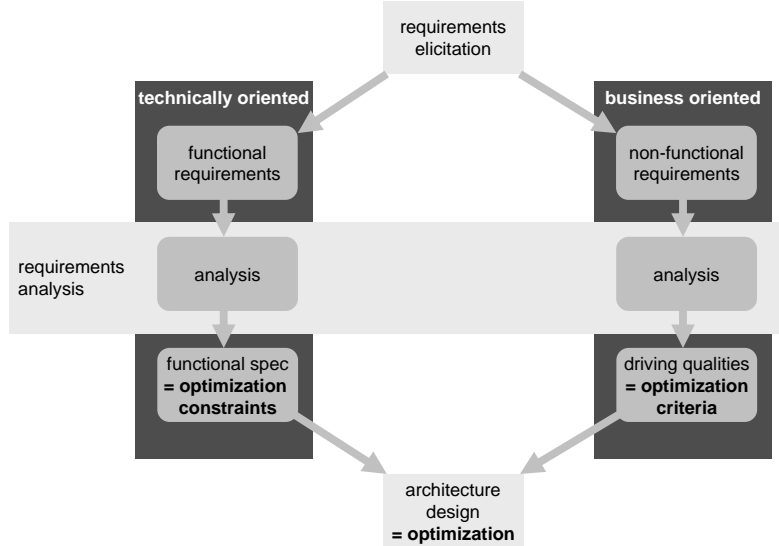


Fig. 1. Architecture design as an optimization process.

Figure 1 shows the process of requirements analysis according to [7]. The non-functional requirements represent the business goals associated with the system-to-be, the functional requirements are technically oriented. During requirements analysis the functional specification is written down and the driving qualities are identified. Driving qualities represent the hard to implement but yet most important stakeholder interests in a product. Because of their important impact on the architecture the driving qualities are also called architectural drivers. Architecture design can be seen as an optimization problem with the driving qualities being the optimization criteria and the functional specification being the optimization constraints.

Structuring principles which support certain qualities help the architect in finding the optimal architecture. These structuring principles can be architectural tactics or styles [7] like information hiding or architectural patterns [8] like a client-server architecture. Most structuring principles affect several qualities, either enabling or inhibiting them, e. g. information hiding supports the maintainability of a system but is impairing its performance. While the architect can choose from a set of well documented principles (see e. g. the work of Booch on a handbook of software architecture [9]) the merging and consolidation of different principles is by and large still an ad hoc and largely unsystematic process to date.

Conflicting quality requirements like performance and maintainability or conflicting structuring principles are compounding the design of a system's software architecture. The resolution of these conflicts relies heavily on the architect's experience and knowledge of the structuring principles. A software architecture represents the tradeoffs between the conflicting qualities that are acceptable for all stakeholders.

We feel that this insight is not regarded sufficiently enough in designing a system using ASMs. Therefore a designer should evaluate the non-functional properties of his ASM model before refining it as described in the next Section. Respectively the consideration of non-functional requirements represented by *scenarios* and their impact on a ground model's structure can be regarded as the first refinement step after constructing and evaluating the ground model.

3 Qualities and ASMs

The non-functional features or quality attributes of a system are generally arranged in a so called quality tree as shown in Figure 2. Whether a product fulfills its quality requirements depends heavily on the architecture of its software. A careful analysis of these requirements therefore establishes the basis for a sound architecture design. Scenario-based evaluation methods [5] analyze the business goals of a product using quality trees and quality attribute scenarios [7]. The quality tree helps to order and rate different qualities while scenarios illustrate single qualities more accurate. Table 1 shows an example scenario for the extendability of an electronic obstacle detection system in a car. The analysis of scenarios involves playing through a scenario with a system model which can be used for ASM models by their notion of a run.

Table 1 shows an example for a scenario refining the quality extendability.

One can argue that the ASM method already supports many of the qualities depicted in Figure 2 by itself. Especially dependability requirements like safety and reliability are supported by the verification and validation means of the ASM method. Nevertheless structuring principles which support certain qualities can improve the system further, a system's reliability can e. g. be improved by adding redundant software modules on diverse hardware platforms [10]. Such structuring principles can be architectural tactics or styles like information hiding or architectural patterns like a client-server architecture.

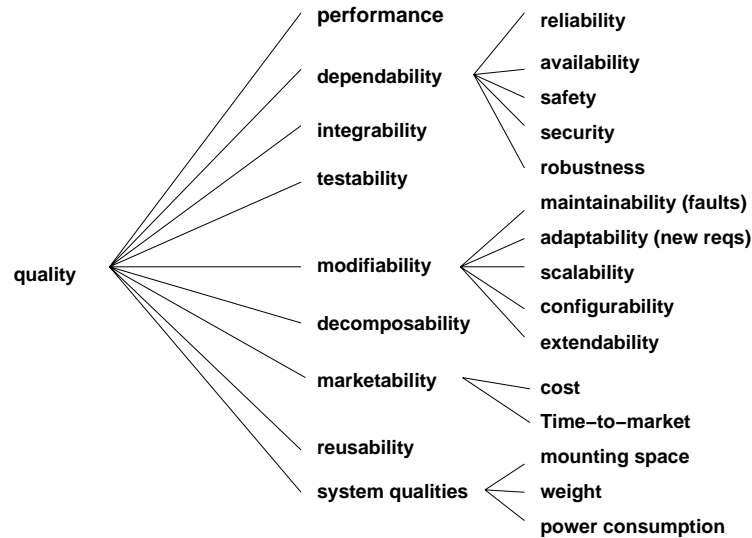


Fig. 2. Example for a quality tree.

On top of this, quantifiable non-functional requirements such as performance features can be formulated as constraints or assumptions on a ground model, thus restricting the legal refinements. One can for example specify the requirement on the overall power consumption on an ASM which is distributed on smaller units during refinement and leads to allocated power for the most refined units. Non-functional requirements that can not be quantified are frequently neglected because they are experientially harder to analyse but crucial for the success of software-intensive systems. Scenarios offer an approach to reason about non-functional or quality features of design alternatives.

4 Conclusion

This paper explains our observation of the need for scenario-based inspection of ASM ground models to include non-functional properties into the design process. This is driven by experiences from the software architecture community and could be enhanced by further cross-fertilization of the two areas including different *views* of a system as recognized by Perry and Wolf [11] and architectural patterns. The latter reveals one of the challenges for our future work, i. e. the modelling of software structures using ASMs that could build on Turbo ASMs [12], which allow the description of structure in terms of refinement, i. e. turning one ASM into several ASMs and thus form the basis for inner structure of ASM models.

Source of Stimulus	customer
Stimulus	an additional sensor shall be used to improve obstacle detection
Environment	modification is requested after finishing development
Response	modification is completed with low effort
Response measure	the modification does not take more than one man month

Table 1. Scenario for extendability of an obstacle detection sensor of a car.

References

1. Börger, E., Stärk, R.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer (2003)
2. Börger, E.: The ASM Ground Model Method as a Foundation of Requirements Engineering. In Dershowitz, N., ed.: Verification: Theory and Practice. Volume 2772 of LNCS. Springer-Verlag (2004) 146–161
3. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall (1996)
4. Ouimet, M., Lundqvist, K.: The timed abstract state machine language: An executable specification language for reactive real-time systems. In: Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS '07). (2007)
5. Dobrica, L., Niemel, E.: A survey on software architecture analysis methods. IEEE Transactions on Software Engineering **28**(7) (2002) 638–653
6. Parnas, D.L.: On a buzzword: Hierarchical structure. Proc. of the IFIP Congress (1974) 336–339
7. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. 2nd edn. Addison-Wesley Professional (2003)
8. Douglass, B.P.: Real-Time Design Patterns. Addison-Wesley Professional (2002)
9. Booch, G.: On architecture. IEEE Software **23**(2) (2006)
10. Salewski, F., Wilking, D., Kowalewski, S.: The effect of diverse hardware platforms on n-version programming in embedded systems - an empirical evaluation. Technical Report 105/2006 (2006)
11. Perry, D.E., Wolf, A.L.: Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes **17**(4) (1992) 40–52
12. E. Börger and J. Schmid: Composition and Submachine Concepts for Sequential ASMs. In Clote, P., Schwichtenberg, H., eds.: Computer Science Logic (Proceedings of CSL 2000). Volume 1862 of LNCS., Springer-Verlag (2000) 41–60

Structuring concepts for sequential composition (and iteration), parameterization, and encapsulation in ASMs are presented.