Methods for Modelling Software Systems: organisers' summary Dagstuhl seminar 06351

Ed Brinksma, David Harel, Angelika Mader, Perdita Stevens, Roel Wieringa

February 27, 2007

Abstract

We survey the key objectives and the structure of this Dagstuhl seminar, and discuss common themes that emerged.

1 Introduction

A proper engineering approach to the construction of complex software systems requires models for their design and verification. Such models must represent both the software system and its (intended) environment. Modelling the environment is needed to define both the events in the environment to which the system must respond, and the effects that the system must have on its environment. A typical environment may consist of other software, hardware, and physical and social systems. The environment of a manufacturing control system, for example, consists of manufacturing information systems, the physical plant, and work procedures to be followed by human operators, all of which affect and complicate the task of modelling at some point of the design and verification processes.

It is clear that the quality of a design or verification process is directly affected by the quality of the models that are being used. They should meet certain quality criteria, such as correctness, understandability and maintainability. An important question that we have to address is what the relevant quality criteria for design and verification modelling are. Are there guidelines on how to achieve them? And how can we validate them?

Models invariably introduce abstractions, and the modeller has, in principle, an obligation to show that he or she has introduced the "right" abstractions. This justification is essentially informal for a number of reasons:

- Complexity: many systems are too complex to be represented without a substantial recourse to abstraction. In practical cases, formal proofs of the adequacy of such abstractions are infeasible, because they somehow rely on the availability of a non-existing model of the "full" system.

Dagstuhl Seminar Proceedings 06351
Methods for Modelling Software Systems (MMOSS)
http://drops.dagstuhl.de/opus/volltexte/2007/957

- Physical reality: this is a non-formal domain by definition, and the quality of a model with respect to its physical environment must be validated by experimental or informal means.
- Social aspects: the user environment of a software system is also informal in nature, but must be taken Into account to ensure that it will respond properly.

It is clear that most abstractions introduced by design and verification models of real systems cannot be justified formally, and must also rely on informal argumentation. The quality of any analysis based on such models, therefore, depends crucially on the quality of informal arguments.

To build models of a certain quality, we need guidelines on how to construct them well. Too often, however, such guidelines are nonexistent or simply ignored in practice, e.g. in verification modelling often "model hacking precedes model checking". And the less we understand about the way a model is built, the harder it is to validate its quality. In design modelling, designers like to proceed as if they were starting from scratch, ignoring any models that have already been developed by others. How can we improve this practice, so that we know how to build models that can be validated?

Although the problems of modelling in design and verification have many similarities, there are differences too. On the one hand, design models represent (an aspect of) the intended structure of the software-to-be. They are prescriptions that must be used by implementors and must include details needed by the implementor. Usability of the model by the implementor is an important quality criterion of these models; completeness is another. For verification models, on the other hand, abstraction is a crucial technique, preserving just as much information about the system as needed to prove correctness, and providing models that can be efficiently verified. Each property to be verified may require a different model. Still, all models, design or verification, must have some isomorphy to the modelled system, so that a from the fact that the model has a property, we can conclude that the system has this property too. This requires a good understanding of the relationship between the model and the modelled system.

The questions that we invited participants in the seminar to address were:

Model quality:

- What are quality criteria for models? How can they be quantified and checked?
- What is the relationship between models and systems in design and in verification?
- What makes an abstraction reasonable?

Modelling method:

- What are the sources and principles for the construction of good models? What is the relation between design and verification models?

- How can the structure of a model be coupled to the structure of the system? What criteria should be used for the structuring of models?
- How to bridge between informal knowledge and formal representation?
- How can we use domain knowledge, and especially engineering documentation to build correct models?

Effectiveness:

- Can we build libraries of problem frames in the domain of embedded software, or in subdomains?

Maintainability:

- Can we build models in such a way that changes in the system (versions) can be easily mapped to versions of models?
- How can changes in the verification property imply changes in the verification model?

2 Structure of the workshop

Almost forty participants from all over the world accepted our invitations and over thirty of them gave talks, represented by papers or abstracts in this volume. Most of the talks were thirty minutes long, some an hour. Summarising what was said seems impractical: one key observation is that participants were working in a wide variety of domains which differ in almost every important respect. The importance of (de)composition, and, closely related, abstraction, was a recurring theme, but participants' ideas of how to address it varied widely. In some contexts, it is possible to decompose a problem according to the demands of verification, for example, so as to isolate and verify a crucial element of the design. In others, problem decomposition is driven by the engineering needs of the system development, and verification must work with what it can get. We heard about many successes applying sophisticated modelling and verification procedures, especially in the domain of embedded systems. Another recurring theme was, however, that we must not and cannot assume that engineers will adopt our formalisms and notations. Success and failure can depend on aspects of the languages that formalists would not always regard as important: for example, pragmatic features such as the ability to lay out diagrams with related elements together may be important to practical usability even if they make no difference to semantics. There are several possible reactions to this, and which is best will depend on the domain: projects can plan to include a verification specialist and to insulate most engineers from the need to understand verification and its associated notations in detail, or projects can carefully choose only those formalisms that can actually be used.

3 Acknowledgements

We thank all the participants for coming and sharing their work and participating in lively and thought-provoking discussion.

We would also like to thank the staff at Dagstuhl who, as always, helped to ensure that we spent an enjoyable and productive week. The (three) organisers who were parents of the (two) children present would especially like to thank Elvira Schnur who looked after them and ensured that they enjoyed their Dagstuhl as much as the rest of us did.

Finally, we would like to thank Wouter Kuijper for undertaking much of the work involved in preparing these proceedings.