06351 Abstracts Collection
Methods for Modelling Software Systems (MMOSS)
— Dagstuhl Seminar —

Ed Brinksma[1], David Harel[2], Angelika Mader[3], Perdita Stevens[4] and Roel Wieringa[5]

[1] University of Twente, NL
brinksma@cs.utwente.nl
[2] Weizmann Inst. - Rehovot, IL
[3] University of Twente, NL
mader@cs.utwente.nl
[4] University of Edinburgh, GB
perdita@inf.ed.ac.uk
[5] University of Twente, NL

Abstract. From 27.08.06 to 01.09.06, the Dagstuhl Seminar 06351 “Methods for Modelling Software Systems (MMOSS)” was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Modelling Methods, Design Models, Verification Models, Problem-solution co-refinement

06351 Summary – Methods for Modelling Software Systems (MMOSS)

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

Keywords: Modelling Methods, Design Models, Verification Models, Problem-solution co-refinement

Joint work of: Brinksma, Ed; Harel, David; Mader, Angelika; Stevens, Perdita; Wieringa, Roel

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2007/957

Comprehensibility of Model Representations: Designing an Empirical Evaluation Framework

Jorge Aranda (University of Toronto, CA)

One of the main purposes of conceptual models is to serve as communication artifacts in software development projects. However, their communication qualities are rarely evaluated -and when they are, the empirical validity and the theoretical support of the evaluations are often questionable.

I will present the progress and some of the obstacles we have found in the design of an empirical framework to evaluate the comprehensibility of model representations, as well as the issues we have identified with other approaches to this problem. Finally, I will briefly cover the roadmap of our model comprehensibility project.

Keywords: Conceptual models, modelling, comprehensibility, empirical study, evaluation

Configurable Modelling Notations and Tools

Joanne Atlee (University of Waterloo, CA)

Semantically configurable modelling notations and tools would enable specifiers to create their own custom modelling notations, and yet have access to tools for editing, manipulating, and analyzing models in those notations.

This talk builds on Jianwei Niu's talk on template semantics, which is a parameterized (i.e., template) semantics definition for a family of notations. Each notation in the family is defined in terms of parameter values that instantiate the template. The result is a semantics definition that isolates as parameters the semantic variation points in modelling notations.

I will talk about how we take advantage of this parameterization to create tools that are configured by semantic parameter values. I will also talk about how we use this formalism to compare notation variants.

Modelling with a grain of SALT

Andreas Bauer (TU München, D)

Creating suitable and correct models for systems verification can quickly become a cumbersome and error-prone activity. In this talk I will present some recent results on the creation and implementation of the Structured Assertion Language for Temporal logic (SALT), which was developed to facilitate the creation of concise, human-readable, and unambiguous temporal specifications that can be used in various verification tasks, such as model checking or runtime verification.

SALT incorporates ideas of existing approaches, such as specification patterns, but also provides nested scopes, exceptions, support for regular expressions and real-time. The latter is needed in particular for verification tasks to do with reactive systems imposing strict execution times and deadlines. However, unlike other formalisms used for temporal specification of properties, SALT does not target a specific domain, such as verification of integrated circuits and CPUs.

Keywords: Temporal logic, specification languages, verification

Full Paper:
http://home.in.tum.de/~baueran/publications/tum-i0604.pdf

The Abstract State Machines Method for Modeling and Analysis of Software-Based Systems

Egon Börger (Università di Pisa, I)

We survey the ASM system design and analysis method that within a single precise yet simple conceptual framework supports and integrates - the major software life cycle activities and - the principal modeling and analysis techniques by linking ASM "ground models" seemlessly to executable code and its maintenance via ASM-refinement-driven design and analysis.

Ground models, in the International Technology Roadmap for Semiconductors also called golden models, are accurate concise high-level system bueprints (system contracts), which are formulated in domain-specific terms and represent the requirements in an application-oriented language that can be understood by all stakeholders. Using various ways to make ASM specifications executable yields simulations to validate ground models against the requirements ("user scenarios").

ASM refinements are an extension of Wirth's and Dijkstra's refinement concept. They support architectural and component design, reflecting modular techniques for accurately crossing system design levels in a way which makes the transformation of ground models by piecemeal, systematically documented detailing of abstract models to code controllable and well documentable for inspection, reuse and maintenance.

The verification and validation reports of ASM refinements contain explicit descriptions of the software structure and of the major design decisions, thus providing an economical way to achieve extendability and modifiability for the design of evolving systems.

The ASM method has been successfully used in a number of industrial projects (in particular at IBM, Siemens, Microsoft and SAP) and of standardization efforts (e.g. at ISO, IEEE, ITU, ECMA) as well as in a variety of research projects on design and analysis (covering both validation and verification) of architectures, languages, compilers, control systems, protocols, web services, etc.

We illustrate the method by a ground model construction for some fundamental web service interaction patterns.

References:
[1] E. Boerger and R. Staerk: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer 2003. http://www.di.unipi.it/AsmBook/
[2] R. Staerk, J. Schmid, E. Boerger: Java and the Java Virtual Machine: Definition, Verification, Validation. Springer 2001, re-published 2003 in Springer's Textbook CD-ROM. http://www.inf.ethz.ch/personal/staerk/jbook
[3] E. Boerger: Construction and Analysis of Ground Models and their Refinements as a Foundation for the Verification of Computer Based Systems. In preparation
[4] M. Barros and E. Boerger: A Compositional Framework for Service Interaction Patterns and Interaction Flows. Springer LNCS 3785 (2005), pp. 5-35.

Keywords: Modelling, Verification, Validation, Ground Model, Refinement, Abstract State Machines

System model for UML – The interactions case

María Victoria Cengarle (TU München, D)

A system model for an OO specification language is any timed state transition system whose states are composed of a data store, a control store, and a message pool. To define a semantics for any OO specification language (as e.g. UML) is the art of defining the transition function Δ depending on the current state and on the input sofar that moreover observes certain rules.

Having defined what a system model is, the challenge now is to establish when such a system model satisfies a message interchange specification (expressed by means of UML interactions).

Keywords: System model, UML, interaction

Partial Behavioural Models for Requirements and Early Design

Marsha Chechik (University of Toronto, CA)

The talk will discuss the problem of creation, management, and specifically merging of partial behavioural models, expressed as model transition systems. We argue why this formalism is essential in the early stages of the software cycle and then discuss why and how to merge information coming from different sources using this formalism. The talk is based on papers presented in FSE'04 and FME'06 and will also include emerging results on synthesizing partial behavioural models from temporal properties and scenarios.

Keywords: Requirements behavioural models

Joint work of: Chechik, Marsha ; Brunet, Greg ; Fischbein, Dario ; Uchitel, Sebastian

Writing formal specifications (models): a method for different languages (notations)

Christine Choppy (Université Paris-Nord, F)

While the issue is (as for many MMOSS participants) how to help writing specifications, we address here the issue of finding out "detail"/properties and expressing them in a formal specification.

C. Choppy and G. Reggio addressed this issue in a JLAP paper (2006) where the target language is CASL-LTL, an extension of CASL (Common Algebraic Specification Language) for dynamic systems (LTL = Labelled Transition Logic). They also show that the proposed method could be used in conjunction with problem frames.

The ideas developed there are general enough that they could be applied to Petri nets, and a first attempt in this direction is presented.

Keywords: Specification method, formal specifications, algebraic specifications, CASL, Petri nets

Joint work of: Choppy, Christine; Reggio, Gianna; Petrucci, Laure

See also: Christine Choppy and Gianna Reggio, A Formally Grounded Software Specification Method, Journal of Logic and Algebraic Programming, Elsevier, 67(1-2):52-86, 2006.

A Taxonomy of Aspects in Terms of Crosscutting Concerns

Jorge Fox (TU München, D)

Aspect-orientation provides support for " Separation of Concerns" by means of techniques that first isolate and then weave concerns.

Most work in aspect-orientation has achieved such goals at the programming level, even also at the modeling level. Though, in some cases the application of these techniques is independent of the problem itself. In other words, the techniques for weaving either code or models are in principle applicable to a number of problems without a clear criterion to answer questions like: in what software processes we may actually discuss aspect-orientation? This also brings other questions: what do we consider an aspect?, how do we deal with it?, are aspects crosscutting concerns? The first notions of aspect-orientation relate to crosscutting in code. We consider this a bottom-up approach. We believe though, that aspect-orientation can be better understood from an architectural perspective.

We call this a top-down approach. We explore the question of "what makes an aspect an aspect" and "when do aspects arise" from a top-down perspective. This work relates to a definition of aspects in terms of requirements traceability, proposes a classification, and altogether a taxonomy.

Keywords: Aspect-orientation, Software Engineering, Taxonomy

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/860

A Framework for Analyzing Composition of Security Aspects

Jorge Fox (TU München, D)

The methodology of aspect-oriented software engineering has been proposed to factor out concerns that are orthogonal to the core functionality of a system. In particular, this is a useful approach to handling the difficulties of integrating non-functional requirements such as security into complex software systems. Doing so correctly and securely, however, still remains a non-trivial task. For example, one has to make sure that the "weaving" process actually enforces the aspects needed.

This is highly non-obvious especially in the case of security, since different security aspects may actually contradict each other, in which case they cannot be woven in a sequential way without destroying each other.

To address these problems, this paper introduces a framework for the aspect-oriented development of secure software using composition filters at the model level. Using an underlying foundation based on streamprocessing functions, we explore under which conditions security properties are preserved when composed as filters. Thanks to this foundation we may also rely on model level verification tools and on code and model weaving to remedy security failures. Our approach is explained using as case-studies a web banking application developed by a major German bank and a webstore design.

Keywords: Aspects in software engineering, aspect interference, verification, semantics, formal methods

Joint work of: Fox, Jorge; Juerjens, Jan

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/859

Model Development in the UML-based Specification Environment (USE)

Martin Gogolla (Universität Bremen, D)

The tool USE (UML-based Specification Environment) supports analysts, designers and developers in executing UML models and checking OCL constraints and thus enables them to employ model-driven techniques for software production. USE has been developed since 1998 at the University of Bremen. This paper will discuss to what extent and how USE relates to the questions and topics (Model quality, Modelling method, Model Effectiveness, Model Maintainability) raised for this seminar.

Keywords: UML, OCL, Model-Driven Development, Validation, Animation, Model Execution

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/861

Model-Based Software Development for Embedded Systems

Ursula Goltz (TU Braunschweig, D)

Model-based Software Development is currently an important issue in the area of embedded systems. In this talk, modelling techniques, methods and tools are presented and discussed using two larger applications.

The first application is the development and validation of a control system for parallel robots, which we carry out together with electrical and mechanical engineers in the context of a large research project (SFB 562) at the TU Braunschweig. We show how we develop a generic modular control system in a model-based approach, and which validation techniques we have developed and used. For modelling, we use in particular interaction diagrams and state diagrams (UML 2.0) for the representation and for the simulation of system behaviour.

The second application project (STEP-X) is a cooperation in the automotive area. With partners from electrical in mechanical engineering and the car manufacturer VW, we work on the development of a seamless development process for electronic control units. A huge part of nowadays innovations in the automotive area is obtained by enlarging the software functionality; this requires new qualities in the development processes. Aim of the project STEP-X was to show a seamless tool-based development process for the whole range from the requirements down to automatic code generation for ECUs. In the upper part of this process, down to the level of architecture design, we propose to use UML. We show the modelling approaches and discuss the experiences. An important concern is again the modelling of system behaviour and executable specifications.

Finally, we discuss a light weight approach for modelling and simulating behaviour using Harel/Marelly's Play Engine, which allows to generate a prototype for a system directly from the requirements, using Life Sequence Charts. We present results of a case study from the automotive area.

Keywords: Model-based development, embedded systems, UML, automotive software development

Joint work of: Goltz, Ursula; Florentz, Bastian; Huhn, Michaela; Knieke, Christoph; Mücke, Tilo; Steiner, Jens

The Formal Specification Language mCRL2

Jan Friso Groote (Eindhoven Univ. of Technology, NL)

We introduce mCRL2, a specification language that can be used to specify and analyse the behaviour of distributed systems. This language is the successor of the mCRL specification language. The mCRL2 language extends a timed basic process algebra with the possibility to define and use abstract data types. The

mCRL2 data language features predefined and higher-order data types. The process algebraic part of mCRL2 allows a faithful translation of coloured Petri nets and component based systems: we have introduced multiactions and we have separated communication and parallelism.

Keywords: Specification language, abstract data types, process algebra, operational semantics

Joint work of: Groote, Jan Friso; Mathijssen, Aad; Reniers, Michel; Usenko, Yaroslav; van Weerdenburg, Muck

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/862

Are Specification Methods Problem-Dependent?

Anthony Hall (Oxford, GB)

How can we classify problems and methods so that we can find a good method for solving a particular kind of problem?

Problem Frames and Solution Composition

Maritta Heisel (Universität Duisburg-Essen, D)

To make use of problem frames, complex problems have to be decomposed into simple ones. We have defined architectural patterns corresponding to Jackson's problem frames, which provide solution structures for these simple problems. Now the question arises how to combine the solution structures of the simple subproblems to obtain a solution structure for the complex problem. The presentation addresses this question.

Different subproblems of a complex problem can be related in various ways. They can be independent of each other, they can exclude each other, or they may have to be solved in a specific order. Such information can be used to combine the solutions structures of the subproblem to a solution structure of the overall problem.

Joint work of: Heisel, Maritta; Choppy, Christine; Hatebur, Denis

Modeling Embedded Systems

Jozef Hooman (Radboud University of Nijmegen, NL)

We report about experiences with the modeling of embedded systems in the context of the Boderc project.

In this project, a number of academic and industrial partners collaborate to improve multi-disciplinary development of mechatronic systems. Important part of the approach is the use of high-level models. We present results and observations of two activities in this project.

Part of a car radio-navigation system from Siemens VDO has been used to experiment with a number of performance modeling techniques, such as MPA, SymTA/S, Uppaal, POOSL, and VDM++. (Joint work with Marcel Verhoef.)

To support the design of printer/copiers at the company Océ, which is the carrying industrial partner of the Boderc project, we made a coupling between a UML tool (Rose RealTime) and Matlab/Simulink. To improve the introduction at Océ, the framework has been adapted to a software-based simulation which can be maintained more easily by software engineers. Currently, this frame is used to test real-time software before the hardware is available. (joint work with Nataliya Mulyar and Ladislau Posta, Lou Somers, and Sebastiaan van der Hoest)

Keywords: Embedded systems, Modeling, Multi-disciplinary, UML

Structural Relationships among Models

Michael Jackson (London, GB)

The basis of a problem-oriented view of software-intensive systems is the recognition of three fundamental roles to be played by models in a development: R, the requirement; W, the given properties of the problem world; and S, the specification of the machine's behaviour at its interface with the problem world. These are (ideally) related by the entailment S,W |= R.

Decomposition of a problem into subproblems is analogous (but not identical) to decomposition of a system into components. A subproblem has R, W and S, its machine interacting with a subset of the problem world. Conceptually, subproblems are initially considered in isolation, composition concerns being deferred.

Composition of subproblems involves relationships among all three of their models R, W and S; in general different subproblems use different models, even of the same parts of the problem world. Problem world properties are not in general compositional. Relationships among subproblems, and hence among their models, may be complex.

Keywords: Composition, model, problem, requirement, specificaiton, structure, subproblem

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/863

First ideas on how to use UML Sequence Diagrams in probabilistic model checking

David N. Jansen (RWTH Aachen, D)

Probabilistic model checking is a method to prove or disprove that a given probabilistic system had a given property. The property is mostly expressed in a formal language like PCTL or CSL (probabilistic extensions of CTL). Software engineers find using these formal languages difficult.

In this talk, I will try to give first ideas about how UML Sequence Diagrams could provide means to express a desired property, and I would like to receive comments.

Modeling and Aspect Weaving

Jean-Marc Jezequel (IRISA - Rennes, F)

A model is a simplified representation of an aspect of the world for a specific purpose. Complex systems typically give rise to more than one model because many aspects are to be handled. For software systems, the design process can be characterized as a (partially automated) weaving of these aspects into a detailed design model. While verification is usually feasible on each of the aspects, it is seldom possible on the resulting detailed design because of the size explosion. Hence we need weaving processes that exhibit good composition properties from the point of view of verification. We present an example of such a weaving process for behavioral models represented as scenarios.

Keywords: Model aspect weaving scenarios

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/864

MOCA project: Patterns for modeling workpieces problems

Wouter Kuijper (University of Twente, NL)

Within the MOCA project we focus on the process of modelling embedded systems for the purpose of formal verification.

I will present a case study on the modeling and verification of a control program for a brine plant. I will discuss the patterns that emerge when modeling this kind of workpiece problems. Using these identified patterns I propose a method leading towards efficient and truthful models, while preventing certain common modeling errors.

Verification models for embedded systems

Angelika Mader (University of Twente, NL)

Our goal is to derive verification models for embedded systems, i.e. models that can be analysed with a tool. In a stepwise manner of model derivation decisions have to be taken concerning the environment, abstractions, decompositions, etc. Lists containing possible decisions can help to make the modelling process more efficient and transparent. In the talk I will discuss a number of such possible decisions.

MOCA project: Using problem framing technique in modelling for formal verification – a case study

Jelena Marincic (University of Twente, NL)

Within the MOCA project we are focused on the process of modelling for formal verification. More particularly, we are interested in embedded control systems, which models consist of environment and software descriptions. When designing a model, a modeller describes a part of the real, informal world and starts from informal descriptions like, for example, domain experts diagrams and documents. The process of model design is also informal, because we do not have a formal proof that the model is truthful. Our goal is to find techniques that will guide the modelling process in order to make it more efficient, systematic and that will help to enhance quality of the models.

I will describe a case study - an example of a simple control embedded system for which we designed a verification timed-automata (Uppaal) model. Starting from the informal description, we used Michael Jackson's problem framing technique to classify the problem and to decompose the system. This was used as a basis for the next step, state charts description, which was helpful for deriving a Uppaal model. I will conclude with some observations about the techniques used on this concrete example and ideas for future work.

Keywords: Modelling for formal verification, problem frames

Maintainable Semantic Modelling of Programming Languages

Peter D. Mosses (University of Wales - Swansea, GB)

A formal semantics of a programming language defines an abstract model of the language's implementations. The model determines the primary functional requirements regarding the observable effects of compiling and running programs, independently of any details of compiler design.

Semantic modelling is supported by solid theoretical foundations, and compilers are reasonably simple and well-understood examples of software systems. In some major semantic modelling frameworks, however, the maintainability of models can be quite poor.

This talk presents some semantic modelling techniques which significantly enhance maintainability, and which allow the development of libraries of reusable semantic model components.

Perhaps these techniques might be applicable also when modelling more complex software systems?

Keywords: Semantics, programming languages

Please draw me a model

Pierre-Alain Muller (IRISA - Rennes, F)

This talk presents modeling as the way to parameterize an hypothetical all-purpose generic machine, which can transmute itself into the system to be built.

The machine is supposedly able to handle all aspects of a system, including hardware and software. The parameterization is achieved by a multi-layer representation, that includes data, models of these data, and metamodels (models of the languages used to express the models).

The talk concludes by motivating the need for multiple metamodels, and explains how metamodeling environments such as www.kermeta.org along with programmable hardware such as FPGA might support this vision.

Keywords: Model-driven engineeering, language engineering, metamodeling

Template Semantics : A Parameterized Approach to Structuring Semantics of Modeling Notations

Jianwei Niu (Univ. of Texas at San Antonio, USA)

Template semantics is a formalism to represent the semantics of a family of modeling notations. Template semantics focuses on the similarities and differences among notations: notations' common semantics is represented by a predefined parameterized template, and a notation's distinct semantics is specified as template parameters. The basic computation model of template semantics is a non-concurrent, hierarchical transition system (HTS). Concurrency, synchronization, and communication among HTSs are achieved via composition operators. The definitions of these operators use the template parameters to preserve the notation-specific behavior in composition. Template semantics can be used as an input to a tool that automatically produces an analyzable model.

Keywords: Modeling notations, Semantics, Concurrency, Synchronization, Automated analysis

Joint work of: Niu, Jianwei; Atlee, Joanne; Day, Nancy

See also: 1. Jianwei Niu, Joanne M. Atlee, and Nancy A. Day."Template Semantics for Model-Based Notations", IEEE Transactions on Software Engineering, vol. 29, no.10, pages 866-882, 2003. 2. Jianwei Niu, "Template Semantics: A Parameterized Approach to Semantics-Based Model Compilation", PhD thesis, University of Waterloo, 2005.

Model Transformation Technologies in the context of Modelling Software Systems

Óscar Pastor (Univ. Politèc. de Valencia, E)

Object-Oriented Methods, Formal Specification Languages, Component-based Software Production... During the last two decades, a lot of research and industrial work has been oriented to the objective of generating code from a higher-level system specification, normally represented as a Conceptual Schema. Anyway, many failures to reach the goal has created strong doubts to accept any new proposals offering a "just press the key, and get all the code" strategy.

But currently, the apperance of proposals as MDA, Extreme Non-Programming, Conceptual Schema-Centered Software Development and so on have given a new push to all these strategies. New methods propose sound model transformations, that have to cover all the different steps of a rigorous software production process from an Information Systems Engineering point of view. This must include Organizational Modeling, Requirements Engineering, Conceptual Modeling and Model-Based Code Generation techniques, all of them properly integrated. In any case, the required conceptual primitives must be precisely, formally defined, and the conversion between the different involved models and their corresponding software counterpart must be done in a well-defined way, making possible even the full automation of the process through the use of the corresponding Model Compilers.

This is going to be central topic of the talk/colloquium. What conceptual primitives should be present in a system specification will be discussed. How to use UML to represent them will be analyzed, reducing the current complexity of the proposal by identifying just those diagrams and those modeling constructs really required to create a correct and complete Conceptual Schema. Concretely, how to accomplish the transformation process between the problem space and the solution space will be presented. Finally, some tool support will be also included in order to make more practical the discussion.

Model Transformation Technologies in the Context of Modelling Software Systems

Óscar Pastor (Univ. Politèc. de Valencia, E)

Programming technologies have improved continuously during the last decades, but from an Information Systems perspective, some well-known problems associated to the design and implementation of an Information Systems persists. Object-Oriented Methods, Formal Specification Languages, Component-Based Software Production... This is just a very short list of technologies proposed to solve a very old and, at the same time, very well-known problem: how to produce software of quality. Programming has been the key task during the last 40 years, and the results have not been successful yet. This work will explore the need of facing a sound software production process from a different perspective: the non-programming perspective, where by non-programming we mainly mean modeling. Instead of talking about Extreme Programming, we will introduce a Extreme Non-Programming (Extreme Modeling-Oriented) approach. We will base our ideas on the intensive work done during the last years, oriented to the objective of generating code from a higher-level system specification, normally represented as a Conceptual Schema. Nowadays, though, the hip around MDA has given a new push to these strategies. New methods propose sound model transformations which cover all the different steps of a sound software production process from an Information Systems Engineering point of view. This must include Organizational Modeling, Requirements Engineering, Conceptual Modeling and Model-Based Code Generation techniques. In this context, it seems that the time of Model Transformation Technologies is finally here...

Keywords: Information Systems Design, Software Engineering, Model-Based Code Generation

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/865

A UML-Based Approach for Problem Frame Oriented Software Development

Gianna Reggio (University of Genova, I)

We propose a software development approach that combines the use of the structuring concepts provided by problem frames, the use of the UML notation, together with our methodological approach for well-founded methods. Problem frames are used to provide a first idea of the main elements of the problem under study. Then we provide ad hoc UML based development methods for some of the most relevant problem frames together with precise guidelines for the users. The general idea of our method is that, for each frame, several artifacts have to be produced, each one corresponding to a part of the frame. The description

level may range from informal and sketchy, to formal and precise, while this approach is drawn from experience in formal specifications.

Thus we show how problem frames may be used upstream of a development method to yield an improved and more efficient method equipped with the problem frames structuring concepts.

Joint work of: Choppy Christine, Reggio, Gianna

Modelling Software Product Lines by Feature Diagrams

Pierre-Yves Schobbens (University of Namur, B)

Whatever the method you use to model software systems, you will have to face the evolution and variability of your models.

We propose to use the notion of *feature*. It originates from the telecommunication industry, but we advocate to apply it in any such domain, and to integrate it with any refinement-based development method.

A feature is formalized as a set of model transformations.

We use two techniques to support feature interference detection:

We check redundancy given by refinement between the abstraction levels after transformation, and we detect dependencies on the ordering of transformations.

To record the variabilities offered to the customer, we propose to use *Feature Diagrams*. They currently exist in many variants. We gave them a formal products as set of features, and compared the semantics, that lead to a simple and powerful variant based on multiplicity nodes only.

Keywords: Features, feature diagrams, software evolution

Joint work of: Bontemps, Yves; Heymans, Patrick; Ryan, Mark; Plath, Malte; Schobbens, Pierre-Yves; Trigaux, Jean-Christophe

A Model-Based Approach To Requirements Analysis

Bernhard Schätz (TU München, D)

A major task in designing embedded systems is the systematic elaboration of functional system requirements and their integration into the environment of the complete technical system. The main challenge is to handle the versatile tasks of coordinating a definition of behavior, which is appropriate to the problem. The problem- and design-specifications of the customer related product definition have to be adjusted with and integrated into the manifold requirements of the technical system design. Accordingly, the model-based requirements analysis and system-definition presented here defines a well-structured modeling approach, which systematically aids the goal-oriented formulation and adjustment of the different stakeholder-requirements with the aid of views onto the system and descriptive specification techniques. Thus it allows a clear specification of a

consistent and complete system design. The central steps of this approach are implemented in a requirements management (RM) tool prototype called AutoRAID.

Keywords: Requirements, model-based, tool support

Joint work of: Schätz, Bernhard; Geisberger, Eva; Grünbauer, Johannes

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/866

An Overview of CADP 2006 "Edinburgh"

Wendelin Serwe (INRIA Rhône-Alpes, F)

In this talk, we present the essential features of CADP 2006 Edinburgh, the next stable version of the CADP (Construction and Analysis of Distributed Processes) toolbox. CADP is rooted in concurrency theory and allows to design and manipulate formal models of asynchronous systems. CADP 2006 Edinburgh provides advanced verification and performance evaluation features and, above all, a unified framework in which the major state space reduction techniques can be combined (including compositional verification, on-the-fly verification, partial order reduction, static analysis, and massively parallel verification). We summarize the main evolutions of the toolbox since 2001 and present the new CADP tools as well as significant enhancements brought to existing tools.

Developing models incrementally and interactively?

Perdita Stevens (University of Edinburgh, GB)

Working with models involves managing the tension between two imperatives:

1) to ensure that the models are fully thought-through, internally consistent, and appropriately complete

2) to ensure that the models flexibly track external decisions and uncertainties, avoiding spurious accuracy.

I will review my recent work with Jennifer Tenzer on using games for UML software design, and will discuss future directions and challenges.

Analysis of Zeroconf Using Uppaal

Frits Vaandrager (Radboud University of Nijmegen, NL)

Formal methods have been applied frequently to analyze (critical parts of) standards for communication protocols and it has been demonstrated that their application may help to improve the quality of these standards.

Nevertheless, despite several decades of formal methods research, formal methods notations have rarely been included in the authoritative part of protocol standards. Also, the relationships between (abstract) formal models and informal protocol standards are typically obscure. It is our ambition to improve this situation. To establish the current state-of-the-art, we report in this paper on a case study where we use Uppaal to formally model parts of Zeroconf, a protocol for dynamic configuration of IPv4 link-local addresses that is defined in RFC 3927 of the IETF. Our goal has been to construct a model that (a) is easy to understand by engineers, (b) comes as close as possible to the informal text (for each transition in the model there should be a corresponding piece of text in the RFC), and (c) may serve as a basis for formal verification. Our conclusion is that Uppaal, which combines extended finite state machines, C-like syntax and concepts from timed automata theory, is able to model Zeroconf in a faithful and intuitive way, using notations that are familiar to protocol engineers. Our modeling efforts revealed some errors (or at least ambiguities) in the RFC that no one else spotted before. We also identify a number of points where Uppaal still can be improved. After applying a number of abstractions, Uppaal is able to fully explore the state space of an instance of our model with three hosts, and to establish some correctness properties.

Keywords: Protocol standards, formal methods, timed automata, zeroconf protocol

Joint work of: Vaandrager, Frits; Gebremichael, Biniam; Zhang, Miaomiao

Full Paper:
http://www.ita.cs.ru.nl//publications/papers/fvaan/zeroconf/

Modeling for Testing

Margus Veanes (Microsoft Research - Redmond, USA)

The use of models in testing is becoming more and more important in the context of industrial software development. We discuss an approach of modeling supported by a tool called Spec Explorer, developed at Microsoft Research, where a tester can provide a high-level description of the system behavior under test as a "model program". In this talk, we discuss the concept of model programs and demonstrate their use by using the tool.

We briefly discuss some internal customer experiences and issues, and mention some future and current work in addressing those issues, in particular by using composition of model programs.

Keywords: Model programs, model composition, conformance testing