Semi-Formal and Formal Specification Techniques for Software Systems

08.10. – 13.10.2000

organized by

Hartmut Ehrig (TU, Berlin), Gregor Engels (Paderborn),
Fernando Orejas (Barcelona) and
Martin Wirsing (Uni München)

Edited by Sebastian John, Technical University Berlin, Dezember 2000

Preface

During the last 20 years several different formal and semi-formal specification techniques have been successfully developed and used. Applications comprise the specification of simple programs, data types and processes as well as of complex hardware and software systems. The variety of specification techniques ranges from formal set theoretical, algebraic and logic approaches for specifying sequential systems and from Petri-nets, process algebras, automata, graph grammars for specifying concurrent and distributed behaviors to semi-formal software engineering methods such as UML which has become the de facto software engineering standard for developing large and complex systems.

Formal and semi-formal approaches have their advantages and disadvantages: the informal diagrammatic methods are easier to understand and to apply but they can be ambiguous. Due to the different nature of the employed diagrams and descriptions it is often difficult to get a comprehensive view of all functional and dynamic properties. On the other hand, the formal approaches are more difficult to learn and require mathematical training. But they provide mathematical rigor for analysis and prototyping of designs. Verification is possible only with formal techniques.

The talks and discussions of a previous seminar held in Dagstuhl in 1998 on these topics have shown that many researchers and research groups are putting more and more effort in closing this gap by integrating semi-formal and formal specification techniques. Their studies and experiences show the added value of combining semi-formal and formal techniques and at the same time open a whole range of new problems and questions which cannot be asked when studying formalisms in isolation. The talks and discussions of the previous seminar have shown particular interest and first results in studying the relationship between UML as de facto standard for diagrammatic modeling languages and CASL as the new standard algebraic specification language.

The goal of this seminar was to study possibilities and solutions for integrating and validating different formal and semi-formal specification techniques. More than 50 participants attended the seminar and more than 30 talks were presented. Most of the talks in this seminar did not concentrate on a single technique in isolation but analyzed, compared or integrated at least two techniques such as data type and process specifications, sequence charts and state charts, sequence charts and data manipulation, B and state charts, formal models and safety analysis, CSP and Object-Z, Petri nets and algebraic module concepts or tile systems and logic programming. In almost a third of the talks UML and its integration with different other specification techniques, such as ASM, graph transformations, CASL or new kinds of behavioral diagrams, was studied; particular emphasis was given to the analysis and foundation of the Object Constraint Language OCL and its integration in the software development process. One other major topic was the use of transformations of graphs or Petri nets for different applications such as the integration of heterogeneous software specifications, visual specification of diagrammatic modeling techniques, semantics of UML and OCL or dynamic meta modeling.

In addition to the talks several discussion sessions were organized on the applicability of formal methods in practice and on the integration with object-orientation. In order to give the many young participants of the seminar the possibility of expressing their opinions and expectations we divided the speakers in two groups: young versus experienced researchers. This lead to stimulating, controverse and inspiring discussions which continued in the evenings.

Our gratitude goes to the scientific directorate of Schloss Dagstuhl for giving us the possibility of organizing this workshop. The TMR program of the European Community allowed us to invite many young researchers; we gratefully acknowledge this support. However, the workshop would not have been possible without the help of the friendly and efficient staff of Schloss Dagstuhl. Our sincere thanks go to all of them.

The Organizers

Hartmut Ehrig *Gregor Engels* *Fernando Orejas* *Martin Wirsing*

Contents

Abstracts in order of presentations at Dagstuhl seminar:

Hartmut Ehrig
Towards a Formal Model and a Component Concept for the Integration Paradigm 1

Martin Große-Rhode
Integration of Heterogeneous Software Specifications based on Transformation Systems 1

Stefan Gruner
A New Tool for the Simulation And Visualization of Distributed Algorithms 2

Maike Gajewsky
Transformations of Petri Net in the context of Software Engineering 3

Wolfgang Reif
Combining Formal Models and Safety Analysis 3

Lutz Schröder
Semantics of Architectural Specifications in CASL . . . 4

Roswitha Bardohl
Visual Specification of Diagrammatic Modeling Techniques 6

Jochen Klose
Using Extended Message Sequence Charts for the Verification of Statemate Designs 6

Ursula Goltz
High-Level Sequence Charts with Data Manipulation . 7

Gregor Engels
Object-Oriented Modeling: A Roadmap 8

Harald Störrle
Conference Report "UML 2000" 8

Maritta Heisel
Structuring the First Steps of Requirements Elicitation 9

Andy Schürr
Adding Graph Transformation Concepts to UML and OCL 10

Albert Zündorf
Formalizing UML Collaboration Diagrams as Graph Rewrite Rules 11

Kevin Lano
Using Restricted Statecharts and B for Reactive Systems Development 11

Reiko Heckel
Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML 12

Elvinia Riccobene
Using ASM to Formalize and Integrate UML 13

Roel Wieringa
Formalizing the tools in TRADE 15

Jürgen Ebert
Z-based Description of Semantics for Visual Modeling Languages 16

Marta Simeoni
Formal Specification of Visual Languages via Algebra Rewriting and Constraint Solving 16

Barak Baruch Chizi
Clustering Methods for improving System Analysis and Design with DfD & OOA 17

Jörg Desel
Non-sequential Petri net semantics and partial algebra 18

Hubert Baumeister
eXtreme Programming and eXtreme Modeling 18

Ugo Montanari
An Interactive Semantics of Logic Programming 20

L. Jason Steggles
Rewriting Logic: A Prototyping Tool for Time Petri Nets 20

Eckehard Schnieder
Description means for Automation Systems: Classification, Evaluation, Application 21

Julia Padberg
Petri Net Modules Transfer of Algebraic Specification Modules 22

Dan Hirsch
Integrating Reconfiguration to Software Architecture Styles using Name Mobility 24

Peter H. Schmitt
Translating OCL 25

Heike Wehrheim
Modelchecking a data and behaviour integrating formal method 26

Gianna Reggio
An Extension of UML for Modelling the nonPurely-Reactive Behaviour of Active Objects 26

Carolyn Talcott
Plan in Maude: A Specification with Multiple Purposes 27

Martin Wirsing
A Hoare Calculus for Verifying Java Realizations of OCL-Constrained Design Models 28

Contribution of authors in alphabetical oder:

Bardohl, Roswitha 6
Baruch Chizi, Barak 17
Baumeister, Hubert 18
Desel, Jörg 18
Ebert, Jürgen 16
Ehrig, Hartmut 1

Engels, Gregor 8
Gajewsky, Maike 3
Goltz, Ursula 7
Große-Rhode, Martin 1
Gruner, Stefan 2
Heckel, Reiko 12
Heisel, Maritta 9
Hirsch, Dan 24
Klose, Jochen 6
Lano, Kevin 11
Montanari, Ugo 20
Padberg, Julia 22
Reggio, Gianna 26
Reif, Wolfgang 3
Riccobene, Elvinia 13
Schmitt, Peter H. 25
Schnieder, Eckehard 21
Schröder, Lutz 4
Schürr, Andy 10
Simeoni, Marta 16
Steggles, Jason L. 20
Störrle, Harald 8
Talcott, Carolyn 27
Wehrheim, Heike 26
Wieringa, Roel 15
Wirsing, Martin 28
Zündorf, Albert 11

Towards a Formal Model and a Component Concept for the Integration Paradigm

Hartmut Ehrig

Technical University of Berlin

(Joint work with Fernando Orejas, Technical University of Catulunya)

In previous papers of the authors an integration paradigm for data type and process specification techniques has been introduced on a conceptual level. A large variety of data type and process-driven specification approaches have been discussed as instances of this paradigm. The first aim of this paper is to present a formal model for the basic layers of this integration paradigm.

The second aim is to propose a generic component concept for integrated data type and process specification techniques corresponding to the architectural layer of the paradigm.

The third aim is to show how well-known integrated specification formalisms and module concepts, like algebraic high level Petri nets, attributed graph transformation and the algebraic module concept, and also new applications can be presented as instantiations of the formal model of the integration paradigm. Concerning new applications the UML diagram techniques are of special interest.

Integration of Heterogeneous Software Specifications based on Transformation Systems

Martin Große-Rhode

Technical University of Berlin

Following the viewpoint model of software systems development abstract models of the components of the systems have to be built representing their desired features appropriately. Separating different viewpoints each model only provides partial information about the component which, however, may be overlapping. The task of integrating these models (or specifications) that may be given in very different languages is then to establish correspondences between them and clarify how they can be considered as one (integrated) specification of the whole system.

The basic idea of the approach presented here is to provide a common semantic domain, given by transformation systems as basic elements, that allows the interpretation of all the different languages. By referring to the same semantic element the correspondence of the syntactic elements can then be deduced.

The precondition for the application of this approach is that the domain of transformation systems provides enough structure. In order to show this its basic elements (categorically: objects), its types of development relations (categorically: morphisms) and its composition operations (categorically: by limits) are presented and some examples of specification languages and their interpretation are sketched. The categorical approach thereby also yields a classification of development relations and composition operations w.r.t. their compositionality.

A New Tool for the Simulation And Visualization of Distributed Algorithms

Stefan Gruner

LaBRI, Université Bordeaux

(Joint work with Mohamed Mosbah and Michel Bauderon)

We motivate and present a new tool for the visualization of distributed computations. Special attention is payed to certain distributed algorithms which have been coded as rewriting systems by Yves Metivier. In order to study the behaviour of algorithms and tool, several experiments have been performed the results of which are presented and discussed. Finally, some important properties of our tool are described and explained. An α-version of our tool will be made available to the public soon. A Technical Report describing details of our results can be found on the WWW pages http://dept-info.labri.u-bordeaux.fr/~stefan/.

Transformations of Petri Net in the context of Software Engineering

Maike Gajewsky

Technical University of Berlin

Enhancement of models within Petri net based process models in literature is covered by two distinguished levels of Petri net transformations. We propose "net class transformation" to add, resp. abstract from aspects like marking, data, roles etc. whereas "model transformation" is used to modify details like exceptions, modification of aspects, optimization etc. In this talk we give the formal foundations of net class transformations as catgorical functors and model transformation as rule-based modification of Petri nets. We present a compatibility condition between the different kinds of transformations, which ensures consistency among models on different layers of abstraction. A list of compatible net class transformations is given which additionally are compatible with structuring techniques.

Combining Formal Models and Safety Analysis

Wolfgang Reif

University Augsburg

The goal is to establish a method for the systematic development of formal models for high assurance systems. The aim is to reduce errors in requirement definitions of software based safety critical systems to get safe specifications. Here we mean 'safe' with respect to correct functioning as well as avoiding hazards because of failures. This is achieved by combining formal models with safety analysis techniques from engineering. Formal specifications are used to describe the system model and to formulate and verify safety properties. Design errors and safety flaws are detected by safety analysis techniques.

The benefit of combining both techniques is three-fold:

- Formal methods benefit from safety analysis because of its systematic way to construct safety requirements, and to break them down to single (software-) components.

- safety analysis benefits from a formal model because of its preciseness, and the option to formally prove correctness and completeness properties of fault trees.
- The resulting system specifications comply with two complementary aspects of safety: correct functioning and fault tolerant design

A formal semantics of fault tree analysis (FTA) was developed. A fault tree describes a top down analysis that breaks down a system hazards to elementary component resp. software failures (basic events). For fault trees two different verification conditions are generated and have to be proved with respect to the formal system model. First, for every gate a completeness condition guarantees, that no event had been overlooked which might cause the hazard. Second, it has to be proved, that the basic events do not occur in the system model. These proofs guarantee, that the hazard described by the fault tree does not occur in the formal system model.

Semantics of Architectural Specifications in CASL

Lutz Schröder

University of Bremen

(Joint work with Till Mossakowski, Andrzej Tarlecki, Bartek Klin and Piotr Hoffman)

A common feature of present-day algebraic specification languages is the provision of operations for building large specifications in a structured fashion from smaller and simpler ones. Much less usual in specification languages are features for describing the modular structure of software systems under development. This purpose is served by the architectural specifications that form part of the algebraic specification language CASL recently developed by the CoFI group.

The main idea is that architectural specifications describe branching points in system development by indicating units (modules) to be independently developed and showing how these units, once developed, are to be put together to produce the overall result. Semantically, units are viewed as given models of specifications, to be used as building blocks for models of more complex specifications, e.g. by amalgamating units or by applying parametrized units.

A crucial prerequisite for a semantics of architectural specifications is the amalgamation property, which allows smaller models to be combined into larger ones under statically checkable conditions. Somewhat informally, the amalgamation property ensures that whenever two models 'share' components in the intersection of their signatures, they can be unambiguously put together to form a model of the union of their signatures. However, while many standard logical systems admit amalgamation, this property fails to hold for the logical system underlying CASL.

We develop a three-step semantics that circumvents this problem. The first step is a purely model-theoretic semantics. Here, amalgamability is just *required* whenever it is needed. This makes the definition of the semantics as straightforward and permissive as possible, but leaves the user with the task of actually checking these model-theoretic requirements. Thus, the natural second step is to give a semantics of architectural specifications in terms of diagrams which express the sharing that is present in the unit declarations and definitions. This allows us to reformulate the model-theoretic amalgamability conditions in 'almost' static terms. A suitable amalgamation property is needed to make the static character of these conditions explicit. The trick used in the third step to achieve this is to embed the CASL logic into a richer logic that does have amalgamation. This makes it possible to restate the amalgamability conditions as entirely static factorization properties of signature morphisms.

Visual Specification of Diagrammatic Modeling Techniques

Roswitha Bardohl

Technical University of Berlin

Visual modeling techniques including the Unified Modeling Language (UML) as well as graph and net based techniques are of growing interest for software system specification and development. The GENGED approach developed at the Technical University of Berlin allows already the generic description of visual modeling languages based on formal graph transformation and graphical constraint solving techniques and tools.

In this talk, the GENGED approach is reviewed and extended in order to allow the description of dynamic behavior and animation of systems. The basic idea is to define visual behavior and animation rules on top of the rules defining the corresponding visual modeling language and to allow a domain specific layout for the animation view of the system. A simple version of the Dining Philosophers is used as running example, where the system view is given by a visual modeling language based on Petri nets. The animation view shows directly the dynamic changes of eating and thinking philosophers.

Using Extended Message Sequence Charts for the Verification of Statemate Designs

Jochen Klose

University of Oldenburg

Message Sequence Charts (MSCs) are widely used to describe the use-cases of a system. In order to also employ them to specify system requirements in an intuitive way — with the goal of formal verification — more expressive power has to be added to MSCs. Damm and Harel have proposed such an extension, called Live Sequence Charts (LSCs), which is presented here. The key concept of LSCs is the distinction between mandatory and possible behavior allowing — among other things — to enforce progress along instances and messages, thereby adding liveness to MSCs. LSCs are provided with a formal semantics which allows them to be used as requirements for modelchecking. Here the principle of the formal verification of Statemate designs against LSCs is presented.

High-Level Sequence Charts with Data Manipulation

Ursula Goltz

Technical University Braunschweig

(Joint work with T. Gehrke, Technical University Braunschweig)

We define socalled n-agent-diagrams for the graphical specification of distributed systems. These diagrams are based on sequence diagrams of the UML, however extend those by integrating the modelling of data transformations. To allow for problem-oriented specifications, we do not fix a specific language for the modelling of data, but rather choose a parametric approach where we just assume certain properties for the data part. Data handling is specified in diagrams by using annotations. We introduce gates as interfaces between instances and their environment, which define the messages which can be received by the instances. Gates are also used to specify the types of the message parameters.

Furthermore, we introduce high-level sequence diagrams, similar to High-level Message Sequence Charts. Here basic n-agent-diagrams (with data) are used as building blocks and composed by means of a general concatenation operator, such that alternatives and loops in system behaviour may be represented. This allows to specify generic behaviour of complex systems in a concise and modular form. The composition of behaviours is studied in combination with data transformations. The concatenation operator for loops and alternatives contains information about the responsibility of instances for decisions regarding the control flow.

We define a textual notation, which can be used as an alternative representation for diagrams. Based on this syntax, we give a formal semantics to n-agent-diagrams and their composition in a process algebraic setting with integrated functional handling of data transformations.

Object-Oriented Modeling: A Roadmap

Gregor Engels

University of Paderborn

(Joint work with L. Groenewegen, Leiden University)

Object-oriented modeling has become the de-facto standard in the early phases of a software development process during the last decade. The current state-of-the-art is dominated by the existence of the Unified Modeling Language (UML), the development of which has been initiated and pushed by industry.

The talk presented a list of requirements for an ideal object-oriented modeling language and compared it with the achievements of UML and other object-oriented modeling languages. This formed the base for the discussion of a roadmap for object-oriented modeling, which is structured according to a classification scheme of six different regions. These regions are (1) the language structure, (2) the model constituents, (3) the model composition, (4) the modeling process (in-the-small), (5) the model review, and (6) the modeling process (in-the-large). Within each region open issues are identified and possible solutions are discussed.

[Remark: The talk was firstly presented at the International Conference on Software Engineering (ICSE'2000) within a special track on the "Future of Software Engineering". More information (papers, slides) can be found at http://www.softwaresytems.org/future.html]

Conference Report "UML 2000"

Harald Störrle

Ludwig-Maximilians-Universität München

The UML is "the lingua franca of the Software Engineering Community", and thus of paramount importance to both informal and formal approaches to software construction. On the recent UML-conference in York, a few interesting trends have emerged:

- Many people now deal with formal semantics of UML. While earlier, the focus was on understanding and on the static models, it is now shifting towards operationalization and the behavioral models. Also, we have seen the first approaches to join individual semantics together. Still, however, most semantics are defined by example and by diagram only, and not as a rigorous transformation based on the metamodel.

- The importance of semantics-based tool-support is now broadly acknowledged, and will lead to greater appreciation of the metamodel, and of metamodeling in general.
- Little advances have been made in the field of using UML for software architecture. A panel discussion identified a number of problems, but compared to last year's UML-conference, the issue has been scaled down in favour of the future of UML.
- The version 2.0 of UML and the future development in general has attracted considerable attention. An issue of eminent consequence and economic interest, heated debate in this field will ensue - possibly behind the closed doors of the OMG, though. We may assume that UML 2.0, due some time late 2002, will bring considerable change in the metamodel.

Structuring the First Steps of Requirements Elicitation

Maritta Heisel

University of Magdeburg

(Joint work with Jeanine Souqui‘eres, LORIA Nancy, France)

We propose to use a system classification and associated diagrams to structure the first steps of requirements elicitation. With system diagrams, we introduce a set of components related by communication links, which represent the system to be developed including its environment. In this setting, requirements elicitation proceeds by first selecting a system class and then mapping the parts of the application domain to the components of the system diagram that corresponds to the chosen system class.

Adding Graph Transformation Concepts to UML and OCL

Andy Schürr

University of the German Armed Forces, Munich

For the next couple of years the Unified Modeling Language UML will be the "lingua franca" of the software engineering community and related disciplines. In their current form the OMG standard documents do a good job in defining the language's abstract syntax using a meta modeling approach, but provide only a very incomplete and imprecise definition of UML's static and dynamic semantics. This well-recognized deficiency of the standard is a major obstacle for the development of UML CASE tools which are more than "nice" diagram drawing tools. Many ad-hoc decisions have to be made to implement useful consistency checkers, code generators or interpreters for (executable) subsets of UML.

With respect to these problems UML could profit from the experiences of the graph transformation community with the formal definition of executable diagrammatic languages. Graph transformation concepts may and have already been used to identify an executable subset of UML collaboration diagrams, to provide an operational semantics definition for UML statecharts, and so forth. Furthermore, graph grammars are — compared with meta modeling concepts — a more "expressive" approach (from the practical point of view) to define the syntax and the static semantics of isolated UML diagram languages as well as consistency relationships across different diagrammatic sublanguages of UML.

Finally, the development of UML's object-oriented constraint language OCL may profit from the experiences made with the design and implementation of rather similar path expression languages, which are used for the construction of graph rewrite rules with complex application conditions. Their type checking rules, their treatment of partially defined subexpressions as well as their definition of different kinds of transitive closure operators may be adopted and lead to a more precisely defined and more expressive future version of OCL.

Formalizing UML Collaboration Diagrams as Graph Rewrite Rules

Albert Zündorf

Technical University Braunschweig

The aim of my current work is to identify a subset of UML class and behavior diagrams that may serve as a visual programming language for object oriented applications that employ complex object structures. As a basis for this work, this talk formalizes object diagrams as the visualization of "objectoriented graphs" and class diagrams as the visualization of graph schemas.

Next, we formalize the graphical parts of collaboration diagrams as graphical representation of graph rewrite rules. Such rewrite rules are embedded into activity diagrams and/or statecharts that provide controlstructures for the execution of graph rewrite rules. Such activity diagrams are used as method body specifications. Finally, we add some invocations to our language and, tra ra, there we have a full fledged visual programming language.

The Fujaba environment (www.fujaba.de) allows to edit such specifications and provides a "push button" generating a complete Java program from a complete specification. No manual Java coding is necessary, anymore.

Using Restricted Statecharts and B for Reactive Systems Development

Kevin Lano

King's College London

(Joint work with K. Androutsopoulos, D. Clark, P. Kan, King's College London)

We describe how a method, called RSDS, has been applied to reactive systems to support their specification, design and verification, using a combination of statechart notation and the B formal method. RSDS provides guidelines for the decomposition of a reactive control system into subsystems, based on the invariants of the system which define its behaviour. Decomposition can be hierarchical, mode-based, or can use template architectures for fault-detection or scheduling. Translation to SMV provides model-checking capabilities for temporal properties, whilst B provides animation and code-generation capabilities. The method has been applied to systems in the areas of process control, automated manufacturing and railway signalling systems.

Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML

Reiko Heckel

University of Paderborn

(Joint work with Gregor Engels, Jan Hendrik Hausmann and Stefan Sauer)

In this paper, dynamic meta modeling is proposed as a new approach to the operational semantics of behavioral UML diagrams. The dynamic meta model extends the well-known static meta model by a specification of the system's dynamics by means of collaboration diagrams. In this way, it is possible to define the behavior of UML diagrams within UML.

The conceptual idea is inherited from Plotkin's structured operational semantics (SOS) paradigm, a style of semantics specification for concurrent programming languages and process calculi: Collaboration diagrams are used as deduction rules to specify a goal-oriented interpreter for the language. The approach is exemplified using a fragment of UML statechart and object diagrams.

Formally, collaboration diagrams are interpreted as graph transformation rules. In this way, dynamic UML semantics can be both mathematically rigorous so as to enable formal specifications and proofs and, due to the use of UML notation, understandable without prior knowledge of heavy mathematic machinery. Thus, it can be used as a reference by tool developers, teachers, and advanced users.

Using ASM to Formalize and Integrate UML

Elvinia Riccobene

Università di Catania

UML (Unified Modeling Language) [10, 11] is a de-facto standard visual modeling language used to specify, visualize, construct and document the structure and the behavior of software systems. However, UML lacks of a formal semantics, and this leads to a more apparent than real understanding of models, makes difficult to perform rigorous analysis and to develop tools supporting mechanical validation and verification.

References [3, 4, 5, 6] report some results of our work [Joint work with Egon Börger (University of Pisa, Italy) and Alessandra Cavarra (University of Catania, Italy)], still in progress, concerning the application of the ASM (Abstract State Machine) [7, 2] formal method to formalize and integrate UML. We are proceeding our research along two directions:

1. Assign to the UML behavioral diagrams (statecharts, activity diagrams, interaction diagrams, etc.) a precise semantics which
   (a) makes explicit the "semantic variation points";
   (b) resolve some of the ambiguities arising from possible different interpretations of UML diagrams;
   (c) serves as a reference model for implementing tools for simulation, verification, and code generation;
2. Integrate UML with formal methods in order to
   (a) study consistency and compositionality of models;
   (b) make formal methods more *practical*, namely easier to use and to understand;
   (c) explore techniques for exploiting formal methods in software development processes which use semiformal languages.

During the seminar, an ASM model for UML State Machines [4] has been presented and a statecharts Simulator [6], developed in AsmGofer [1] (an interpreter for ASMs), has been showed through the concrete example of a state machine representing an ATM machine.

The guidelines for developing a multi-agent ASM model given a UML model have also been illustrated through the case study of the rail-car system modeled in UML by Harel and Gery in [8]. The advantages of having such formal model in order to discover incorrectness, inconsistencies, missing controls, and to help in the formalization of the missing parts and in the definition of the required constraints have been discussed.

[1] J. Schmid. Executing ASM specifications with AsmGofer. http://www.tydo.de/AsmGofer

[2] Abstract State Machines. http://www.eecs.umich.edu/gasm/

[3] E. Börger, A.Cavarra, E.Riccobene. An ASM Semantics for UML Activity Diagrams. In T. Rus (ed.) *Algebraic Methodology and Software Technology* (AMAST 2000), Springer LNCS 1816, pg.287–302, Maggio 2000.

[4] E. Böerger, A. Cavarra, E. Riccobene. Modeling the Dynamics of UML State Machines. In Y. Gurevich, P. Kutter, M. Odersky, L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, pp. 223–241, Springer LNCS 1912, 2000.

[5] A. Cavarra, E. Riccobene. Modeling the Dynamics of UML Behavioral Diagrams. Poster at FORTE/PSTV 2000 (Formal Description Techniques for Distributed Systems and Communication Protocols/Protocol Specification, Testing and Verification).

[6] A. Cavarra, E. Riccobene. Simulating UML Statecharts. Submitted to the ASM Workshop at EUROCAST 2001.

[7] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger (Ed.): Specification and Validation Methods, Oxford University Press, 1995.

[8] D. Harel, E. Gery. Executable Object Modeling with Statecharts. IEEE Computer, pp. 31–42, 1997.

[9] Rational Software Corporation, *Unified Modeling Language (UML)*, version 1.1, http://www.rational.com, 1997.

[10] *UML Notation Guide*, 1997. (Published as part of [9]).

[11] *UML 1.1 Semantics*, 1997. (Published as part of [9]).

Formalizing the tools in TRADE

Roel Wieringa

University of Twente

TRADE is a set of Techniques for Requirements and Architecture DEsign of reactive (software) systems, collected from an in-depth analysis of structured and object-oriented methods and techniques. Some of the techniques in TRADE are formalized, others are essentially informal. The goal of defining TRADE is to teach a simple and coherent set of software design techniques to academic students that they can use in practice later on. The techniques in TRADE also come with a set of guidelines for finding and validating models build by the techniques in TRADE. The basic idea of TRADE is that a reactive system is connected to a subject domain so that it can respond to subject domain events by producing desirable subject doman actions. Both the environment and the system are modeled according to three aspects: functionality, behavior and communication. Both the environment and the system can be decomposed, and the components again can be described according to the aspects of functionality, behavior and communication. (Quality attributes are not considered in TRADE.) The techniques used to describe this are the mission statement and function refinement tree (functionality), event list (behavior) and communication diagram (communication). Decomposition is described by simple class diagrams. For each of these techniques, there are simple and more advanced versions, but we stick to the simple ones to see how far we can go with those. The talk discusses the (in)formality of the semantics of these techniques. It turns out the the centerpiece of a formal semantics consists of certain aspects of behavior and comunication; all other pieces are essentially informal. I argue that especially for these informal parts, precision is possible and desirable, but that this precision must be reached by using natural language supplemented with simple diagrams.

Z-based Description of Semantics for Visual Modeling Languages

Jürgen Ebert

University of Koblenz-Landau

(Joint work with Roger Süttenbach, Compaq Computer International B.V., Galway Ireland)

A way how to specify the semantics of visual modeling languages (VMLs) was presented. The specification is based on the abstract syntax of diagrams. The class of abstract syntax graphs (ASGs) for a given language is defined by a corresponding extended entity relationship diagram (EER) and additional by constraints in the Z-like graph specification language GRAL.

The ASG is used as a given element in a Z-specification which describes an abstract automaton by a semantic basis (SB) consisting of four parts: global definitions, configurations, transitions, and inititial configurations. The analogous construction of semantics for alls VMLs allows for integration by composing EER/GRAL-descriptions and SBs.

Formal Specification of Visual Languages via Algebra Rewriting and Constraint Solving

Marta Simeoni

Università di Venezia

(Joint work with Roswitha Bardohl and Martin Große-Rhode, Technical University of Berlin)

The definition of Visual Languages by generating rules, as proposed by the GENGED approach (see [Bar00]), supports both the description and the manipulation of visual sentences.

Defining Visual Languages means dealing with two different aspects, namely the description of the logical stucture of the language and its graphical layout. We propose an approach for the formal definition of both the logical and graphical aspects base on algebra rewriting (according to [Gro99]) and constraint satisfaction techniques. More precisely, the specification of visual sentences is given by algebraic specifications while the transformation of a visual sentence is described by the integration of an algebra rewriting rule (which transforms the logical structure of the sentence) and a constraint satisfaction problem (which deals with the transformation of the graphical layout).

[Gro99] M. Große-Rhode "Specification of State Based Systems by Algebra Rewrite Systems and Refinements" *Tech. Report TU-Berlin 99–04* (1999)

[Bar00] R. Bardohl "Visual Definition of Visual Languages based on Algebraic Graph Transformation" *Phd Thesis, Kovak Verlag* (2000)

This research has been partially supported by the EC TMR Network GETGRATS (GEneral Theory of GRAph Transformation Systems) and Esprit Working Group APPLIGRAPH.

Clustering Methods for improving System Analysis and Design with DfD & OOA

Barak Baruch Chizi

University Tel Aviv

We develop a method to improve efficiency of information system structure. It also reflects on the organization performance.

We consider two approaches: the structured analysis (data flow diagrams) and the object oriented analysis. The methods applies to an MIS developed by one of the two approaches and improve the quality and productivity of the representation.

We use a defined case study in order to demonstrate the swift pass between the current state to a suggested state, using clustering techniques. The performance measures are: coupling, complexity, and modularity.

The algorithm was implemented on a and the system coupling was reduced by 28 % for the structured analysis and for the object oriented analysis by 17 %. We also reduced the system complexity by 53 %.

This work provides a unique overall view of information system analysis and points out systematically the weakness of an MIS, and the relation to organization performance.

Non-sequential Petri net semantics and partial algebra

Jörg Desel

Katholische Universität Eichstätt

In many different application areas, people need and invent new semi-formal notions based on Petri nets. In particular, the occurrence rule for transitions differs for these Petri net classes, i.e., there are different restrictions for the enabling of a transition.

In this talk I present a systematic approach to obtain a partial order semantics from the occurrence rule. This approach is based on concepts of partial algebra. Partially ordered runs are constructed via process terms that use operators for sequential and for concurrent composition. Concurrent composition is defined only partially, depending on the respective occurrence rule.

In the case of elementary net systems and of contextual nets, the obtained semantics coincides with known partial order semantics. In the paper:

J. Desel, G. Juhás and R. Lorenz: Petri nets over partial algebra, to appear in: H. Ehrig, G. Juhas, J. Padberg and G. Rozenberg (eds.), Unifying Petri Nets, Advances in Petri Nets, LNCS it is shown how a partial order semantics for many more Petri net classes can be obtained.

eXtreme Programming and eXtreme Modeling

Hubert Baumeister

Ludwig-Maximilians-Universität München

Recently a lightweight software development method called extreme programming (XP), developed by Kent Beck and Ward Cunningham, received a lot of attention. XP is build around 12 practices which are based on common software engineering practice, like small iterations, testing, and code review, put to the extreme. Small iterations leads to the XP practice of continuous integration, which requires a system integration several times a day. Testing leads to the 'test first' paradigm and to automatic tests that are run whenever the source code is changed. Code review leads to pair programming, where two programmers sit before the same computer and while one programmer programs, the other reviews constantly the first programmers work.

Each of these extreme practices alone may even be harmful; however, the disadvantages of one practice is counter acted by the advantages of another. Thus, these extreme practices taken together have a synergistic effect. The more practices are used together, the greater the benefit will be.

One characteristic of XP is based on the observation that documentation and diagrams are not kept up-to-date with the implementation. Therefore, once diagrams are used to get an understanding of what is to be implemented, they are thrown away and the code together with the tests are essentially the program documentation. To know something about the program one asks the program by either reading its source code or by writing tests and executing them. Of course, this only works if the code reflects the intention of the programmer, a simple design is used, and coding standards are enforced.

Since the beginning of this year, there are some approaches trying to combine modeling techniques using diagrammatic notations with XP programming pioneered by the paper "Extreme Modeling" by Marko Bogner and others presented at the XP 2000 conference in Italy. The basic idea is that a subclass of UML-models can be executed by suitable virtual machines. Then, instead of writing programs and tests in conventional programming languages, like Smalltalk or Java, executable UML-models are constructed. These models can be used to test other models and programs.

In the talk I give an overview over extreme programming and extreme modeling.

An Interactive Semantics of Logic Programming

Ugo Montanari

Università di Pisa

(Joint work with Roberto Bruni and Francesca Rossi)

We apply to logic programming some recently emerging ideas from the field of reduction-based communicating systems, with the aim of giving evidence of the hidden interactions and the coordination mechanisms that rule the operational machinery of such a programming paradigm. The semantic framework we have chosen for presenting our results is tile logic, which has the advantage of allowing a uniform treatment of goals and observations and of applying abstract categorical tools for proving the results. As main contributions, we mention the finitary presentation of abstract unification, and a concurrent and coordinated abstract semantics consistent with the most common semantics of logic programming. Moreover, the compositionality of the tile semantics is guaranteed by standard results, as it reduces to check that the tile systems associated to logic programs enjoy the tile decomposition property.

Rewriting Logic:
A Prototyping Tool for Time Petri Nets

L. Jason Steggles

University of Newcastle

In order to model and reason about real-time systems a variety of extensions of Petri nets with time have been proposed in the literature. However, problems arise due to the semi-formal description of these extensions and the sheer number of different semantic choices that can be made:

- timing information can be assigned either to places, transitions, arcs or tokens;
- time durations or intervals can be used;
- specified time can represent a period of inhibition or a period

when an activity can occur;

- variety of rules can be used to reset local clocks.

The tools currently available are unable to cope with these problems and thus practical investigation of these different time tensions is not feasible.

In this talk we will consider using RL and an associated support tool Elan as an environment for formalising, rapidly prototyping and analysing Petri nets with time. Rewriting logic (RL) is an extension of standard algebraic specification techniques which uses rewrite rules to model the dynamic behaviour of a system. The flexible approach we propose allows the wide range of possible time extensions presented in the literature to be formalised and investigated; thus it overcomes one of the major drawbacks of the current hardwired tools. We demonstrate our ideas by modelling one possible semantics for *time Petri nets*, a time extension in which transitions are associated with a firing interval. We conclude by considering what it means for an RL model to correctly simulate such a Petri net extension with time.

Description means for Automation Systems: Classification, Evaluation, Application

Eckehard Schnieder

Technical University Braunschweig

Due to the complex nature of automation systems, combining both control and plant, several kinds of notations have been developed to cover all aspects and views to be regarded for requirements, planning, design and operation. A general system understanding concerning state-space, structure, causal and temporal dynamics lead to a convenient classification framework for systems and their description means. The latter — in german "Beschreibungsmittel" — and their orthogonal dimensions for system development methodology and tool — german: "Werkzeug" — form the BMW-principle.

Following the system concept the conceptual terms structure, function and behaviour form a more detailed classification scheme, in which behaviour can be covered by the dimensions and metrics of state quantity resolution versus process evolution resolution, e.g. discrete, continuous and hybrid behaviour.

Due to different aspects of the individual phases during a system life cycle each classification item can be weighted differently. From an economical view, however, holistic notations in combination with homogeneous tool-chains are very attractive. In order to achieve high quality, especially the degree of formality should allow formal verification in early development phases, but at the same time provide a suited representation of all relevant aspects of the application domain. For this purpose generic frames for ressource models with notation means integration or data model integration or data exchange are required.

To illustrate all aspects specific examples from automation engineering are given.

[1] E. Schnieder: Methoden der Automatisierung. Vieweg Verlag 1999.

Petri Net Modules Transfer of Algebraic Specification Modules

Julia Padberg

Technical University Berlin

In the context of developing and investigating concepts for components and component-based systems [Web99] the question of a compatible concept for various formal methods comes into focus. An important and widely accepted concept are algebraic specification modules [WE86, EM90]. Based on algebraic specifications a modules consists of parameter, import, export, and body specifications. These are related by specification morphisms. The theory of algebraic specification modules is well developed and comprises a (contravariant) semantics based on the semantics of algebraic specifications. Various module operations, as union, composition, actualization are available and compatible with the module semantics. An adaption of this to Graph transformation systems and labeled action system s has been given in [Sim00]. There the parameter part has been dropped and the semantics base on transition systems of the corresponding graph transformation systems.

Here we have adopted this approach to Petri nets, and have given first some examples for various Petri net modules and their composition. In order to achieve such a module concept for Petri nets a generic approach that is feasible not only for Petri nets has been suggested. We have given a reformulation of *Cat*-modules (introduced in [Sim00]) and have introduced a set of conditions that allow the definition of *Cat*-modules and their operations. We have given two classes of different refinement morphisms for Petri nets, namely loose and substitution morphisms. The first are morphisms mapping place to places and transitions to transitions. There the pre and post domain of the mapped transition may have less adjacent arcs that the target transition. This class of morphisms satisfy the *Cat*-module conditions, so we have directly the above module operations union and composition. The class of substitution morphisms are given by the usual transition substitution, where one transition is substituted by a subnet. This class is most likely to satisfy the *Cat*-module conditions as well.

A detailed discussion of the possible semantics of these modules based on the semantics of the underlying specification formalism has been given. The two possibilities comprise a contravariant and a covariant semantics. the first is based on a contravariant semantics functor of the underlying specification formalism as e.g. for algebraic specifications. The second is analogously based on a covariant functor, that could be the case for graph transformation systems. We have sketched the possibilities of a contravariant semantics for Petri nets based on classes of transition systems.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, volume 21 of *EATCS Monographs on Theoretical Computer Science*. Berlin, 1990.

[Sim00] M. Simeoni. *A Categorical Approach to Modularization of Grap Transformation Systems using Refinements*. Università Degli Studi Di Roma "La Sapienza", Phd thesis, 1999.

[WE86] H. Weber and H. Ehrig. Specification of Modular Systems. *IEEE Transactions of Software-Engineering*, (SE - 12 (7)):784–798, 1986.

[Web99] H. Weber. Continuous Engineering of Communication and Software Infrastructures. volume 1577 of *Lecture Notes in Computer Science 1577*, pages 22–29. Berlin, Heidelberg, New York, 1999.

Integrating Reconfiguration to Software Architecture Styles using Name Mobility

Dan Hirsch

Universidad de Buenos Aires

(Joint work with Ugo Montanari, Universitá di Pisa and Paola Inverardi, Universitá dell'Aquila)

An important issue in the area of software architecture is the specification of reconfiguration and mobility of systems. This talk presents an approach for the specification of software architecture styles using hyperedge replacement systems and for their dynamic reconfiguration using constraint solving. A system architecture is represented as a graph where edges are components and nodes are ports of communication. Then, a style is represented as a graph grammar where the instances of the style are the graphs generated by the corresponding grammar. The construction and dynamic evolution of the style are represented as context-free productions and graph rewriting. To model reconfigurations we allow the declaration, creation and matching of new nodes (i.e. ports of communication) and use constraint solving over the productions of the style grammar for achieving synchronization. In this way complex evolutions can be specified in a more expressive and compact form than using pi-calculus style languages for mobility.

Translating OCL

Peter H. Schmitt

University of Karlsruhe

This talk is concerned with the problem of defining a semantics for the OCL. As in most previous attempts on this problem, we translate OCL expressions into a known laguage with well-understood semantics. In our case this is first-order dynamic logic. The talk includes a 3-slide intoduction to Dynamic Logic: DL is a multimodal logic that arises from Hoare triples (which we assume to be known to the audience) by closure under the logical connectives "or" and universal quantification. Given n OCL expression e in an UML diagram D we first fix the syntatical material that may be used in the translated DL formula. This follows, more or less, the suggestions in the literature: classes give rise to type symbols, an association gives rise to two function symbols — one for each direction of the association, pre-defined OCL-operations, attributes and query methods are translated into function symbols. The talk will concentrate on two details:

1. How to translate the iterate construct? This as a typical example, where pure first-order logic is not expressive enough. In DL it is of course simple to inculde the simple for-loop that evaluates an iterate expression as a program in an modal operator.

2. How do deal with the @pre construct in OCL postconditions? This problem is more delicate, since post conditions to an method m are in OCL (of course) evaluated in the system state after execution of m. In DL on the other hand one inteprets in the state before the execution of m an statement of the form <m>F for an appropriate formula F. A systematic solution to this problem is presented.

Modelchecking a data and behaviour integrating formal method

Heike Wehrheim

Universität Oldenburg

(Joint work with Clemens Fischer)

The integration of several different modelling techniques into a single formal method has turned out to be advantageous in the formal design of software systems. In this work we present an integrated formal method with the name CSP-OZ, an integration of the process algebra CSP with the state-oriented formalism Object-Z (an object-oriented extension of Z). CSP-OZ has a uniform formal semantics in the style of the failure-divergence model of CSP. The semantics preserves the refinement concepts of the two formalisms: separate refinements of the CSP and the Z part induce an overall failure-divergence refinement in the common semantics. We propose a method for checking correctness of CSP-OZ specifications via a translation into the CSP dialect of the model checker FDR. In this dialect CSP is combined with a small functional language which can be used for encoding the data part of CSP-OZ specifications. Thus verification of CSP-OZ specifications becomes possible.

An Extension of UML for Modelling the nonPurely-Reactive Behaviour of Active Objects

Gianna Reggio

Universita' di Genova, Italy

(Joint work with E. Astesiano)

Modelling nonpurely-reactive systems, such as agents and autonomous processes, does not find a direct support in the UML notation as it stands, and it is questionable whether it is possible and sensible to provide it in the form of a lightweight extension via stereotypes.

Thus we propose to extend the UML notation with a new category of diagrams, "behaviour diagrams", which are, in our opinion, complementary to statecharts for nonpurely-reactive processes (active objects). The proposed diagrams, to be used at the design level, also enforce localization/encapsulation at the visual level. Together with motivating and presenting the notation, we also discuss the various possibilities for presenting its semantics in a palatable way, depending on the reader.

Plan in Maude: A Specification with Multiple Purposes

Carolyn Talcott

Stanford University

(Joint work with Jose Meseguer and Mark-Oliver Stehr, SRI International)

Plan is a language being developed at UPenn for programming active networks. Plan programs are sets of active packets that travel through a network, executing code on specified nodes. The active nature of Plan programs makes it important to have a formal specification of the semantics. The multiple facets of Plan present several interesting specification challenges

- a language for programming mobile agents $\Longrightarrow$ concerns of security
- a language for programming networks $\Longrightarrow$ concerns of resource usage
- a scripting language for combining services provided by network nodes
$\Longrightarrow$ need for rely/guarantee style reasoning

Maude is a language and implementation based on rewriting logic that is well suited for specifying, simulating and analysing distributed systems. We describe the challenges and some progress towards a specification of Plan in Maude that can be used

- by programmers to understand language constructs and as a simulation tool
- implementors as a reference model

- algorithm designers to prove that Plan programs meet their specifications
- language designers to experiment with and reason about proposed language modifications and extensions.

Slides can be found at http://www-formal.stanford.edu/clt/talks.html

A Hoare Calculus for Verifying Java Realizations of OCL-Constrained Design Models

Martin Wirsing

Ludwig-Maximilians-Universität München

(Joint work with Rolf Hennicker and Bernhard Reus)

The Object Constraint Language OCL offers a formal notation for constraining the modelling elements occurring in UML diagrams. In this talk we apply OCL for developing Java realizations of UML design models and introduce a new Hoare-Calculus for Java classes which uses OCL as assertion language. The Hoare rules are as usual for while programs, blocks and (possibly recursive) method calls. Update of instance variables is handled by an explicit substitution operator which also takes care of aliasing. For verifying a Java subsystem w.r.t. a design subsystem specified using OCL constraints we define an appropriate realization relation and illustrate our approach by an example.