Dagstuhl Seminar No. 02111
Report No. 337

Concurrency and Dynamic Behaviour Modelling: Pragmatics & Semantics

10.03. – 15.03.2002

organized by

Gregor Engels (University of Paderborn, Germany),
Rob van Glabbeek (Stanford University, USA) and
Ursula Goltz (TU Braunschweig, Germany)

Edited by Jochen Küster, University of Paderborn, April 2002

Preface

A topic which has gained increasing interest in the past years is the modeling of distributed and concurrent systems. Typical applications are for example in the area of real-time, embedded, and component-based systems, Web-based and multi-agent systems. The complexity of such systems in combination with high demands on their reliability call for adequate design methods.

Concurrency theory provides a formal basis for specifying such systems, consisting of approaches such as process algebra and Petri nets for modeling, logics for expressing properties of concurrent systems, and methods for analysis and verification; Pi-calculus, ambients and control structures provide mobility concepts. Semantic models underlying these concepts were investigated, for example transition systems with various notions of equivalence and event structures. Coalgebras and hidden algebras provide a uniform framework for modeling dynamic behavior and modularization. However, the impact of these developments on practical software development has been limited. One reason is the lack of integration of specification techniques for different aspects of software development, and the missing support for specific application domains and methodologies. Another reason lies in the difficulties of practitioners in reading and writing formal specifications.

Software engineering methods are being developed which specifically address these issues. For example, the Unified Modeling Language (UML) integrates design notations for specifying the logical and physical structure of a system, its dynamic behavior, the interaction with other systems, etc. Being a general-purpose language, the UML provides mechanisms for defining domain-specific profiles of the language. An intuitive diagrammatic notation allows its use by application developers without background in formal methods. However, as UML lacks a formal foundation, models are often

ambiguous, and there is no satisfactory support for analysis and verification of models.

The goal of this seminar was to bring together people from both areas of research for the mutual benefit of

- discussing the technology transfer from concurrency theory to (in particular) object-oriented modeling, and
- deriving new challenges for concurrency theory from problems in practical software development.

In particular, the following topics have been addressed:

- Semantics of behavioral models, including problems of under-specified and open systems.
- Consistency between between non-orthogonal sub-models.
- Support for methodologies and specific application domains.
- Adequacy and expressiveness of behavior models, abstraction levels in modeling.
- Analysis and verification (model checking, etc.), code generation.
- Advanced concepts like time and mobility.

The discussion of these and other issues between experts from the research fields outlined above led to a better understanding of the semantics of models for dynamic behavior of concurrent systems. In a working group, perspectives on further developments both from the theoretic and pragmatic point of view have been discussed.

The Organizers

Gregor Engels *Rob van Glabbeek* *Ursula Goltz*

Contents

Abstracts in alphabetical order:

Specifying Evolving Concurrent Information Systems Using Object Petri Nets

Nasreddine Aoumeur

University of Magdeburg

For the development of complex information systems we have been proposing a new variant of object Petri Nets. Its aspects include a true concurrency semantics based on rewrite logic, a component-oriented conceptualization and a meta-level for dealing with run-time changes. This presentation overviews the main aspects and features of this model using a simplified banking systems example.

Process Algebra with Pointers

Jos Baeten

TU Eindhoven

Joint work with Jan Bergstra and Loe Feijs

We present a process algebra for mobile processes without bound or free variables. Instead, pointers are used, that refer back to an action executed in the history of a process. The situation is comparable to a presentation of the *lambda*-calculus with DeBruijn indices. This may further work on the *pi*-calculus, e. g. for axiomatization, comparison of different notions of equivalence, implementation.

Modal Logics for Mobile Agents

Rocco De Nicola

University Firenze

Joint work with Michele Loreti

I briefly presented KLAIM (a Kernel Language for Agents Interaction and Mobility) and showed how it can be used to program applications distributed over sites of wide area networks. I then presented

a new logic, tailored on KLAIM, that can be used for describing properties of nets of processes modelled in KLAIM. The logic is an adaptation of Hennessy-Milner modal logic but is geared toward the descriptions of accesses, resources and mobility. Indeed, the new modalities have a richer structure: actions are replaced by predicates over transitions that permit describing both the nature of the actions and the sites involvements. After introducing the logic, I described the possibility of having located formulae, i.e. formulae that are restricted to specific sets of sites. These formulae can be exploited to set up a methodology for proving properties of open net and for developing stepwise implementations from abstract contextual specifications to concrete nets while guaranteeing preservation of properties.

Modeling Flexible Workflows

Jörg Desel

Katholische Universität Eichstätt-Ingolstadt

Flexible workflows are process specifications that are partly structured and partly unstructured. The structured parts have few alternatives and are modeled by Petri nets as usual. In contrast, unstructured parts capture exceptions and provide more freedom to the user to execute activities in an arbitrary way. We suggest to specify these parts by sets of simple activities, that can be used as building blocks, and additional declarative specifications, that can be formulated graphically. We discuss how flexible workflows can be obtained and validated from a set of runs given as occurrence Petri nets. The approach is partly impemented in the VIP-tool. Finally, different application scenarios for flexible workflows are discussed.

Avoiding State Space Explosion in Distributed Checking

Hans-Dieter Ehrich

TU Braunschweig

Joint work with Ralf Pinger

When model checking a concurrent system, the model conventionally represents concurrency either by global clock synchronization or by interleaving of its sequential components. This leads to the well-known "state space explosion problem" which has been tackled in several ways, including symbolic model checking, partial-order reduction, abstraction, and especially a number of compositionality methods. Very large systems have been successfully checked, demonstrating the power of the techniques.

We propose a compositionality method separating global conditions automatically into local conditions and communication requirements such that satisfaction of local conditions entails the global condition, provided that the communication requirements are fulfilled. Moreover, we use the more general "perspective concurrency" composition semantics, allowing for synchronization or interleaving in any combination.

The method is based on distributed logic D1 and its translation to D0 as described by Ehrich and Caleiro in Acta Informatica 2000. The method has been completed by Pinger (PhD thesis, 2002), providing an algorithm for matching communication requirements, implementing the method, and showing its practicality with an example of realistic size. The improvements over the only fully automatic method known so far (not implementing assume-guarantee methods) are quite promising.

The Modeling Triangle

Gregor Engels

University of Paderborn

Software engineering proposes the usage of a model as intermediate step on the long path from a given problem to a program, solv-

ing this problem. Such a model abstracts from irrelevant details from the problem domain on the one hand, and from implementation level details on the other. A model itself should be defined by a modeling language with a (hopefully precisely defined) syntax as well as with a (hopefully precisely defined) semantics. Unfortunately, current practice in (industrial) software development shows that used modeling languages lack sometimes a precisely given syntax and lack very often a precisely defined semantics. The talk discusses this problem and illustrates it by the use of a so-called modeling triangle. This modeling triangle, consisting of the three interrelated dimensions aspects, syntax, and semantics, makes clear that syntax and semantics of a modeling language should be based on a commonly agreed set of aspects, which could be termed generic conceptual model. Such a generic conceptual model provides means to abstract from the real world domain, it forms the base of a corresponding modeling language and it helps stakeholders like language designers and users, to deploy the modeling language with the same intuitive understanding.

Using UML Models for Dynamic Behaviour

Ursula Goltz

TU Braunschweig

Joint work with Karsten Diethers, Michaela Huhn and Martin Mutz

Two applied projects and resulting research issues are presented:

The first project is a part of a collaborative project, funded by the DFG (SFB 562), concerned with building high speed "parallel" robots. We are using UML for specifying the architecture for the robot control and we validate crucial parts by formal analysis (based on existing tools). For the latters, we consider semantical issues for the UML models we are using, including the questions of consistency between system views and integration of timing considerations.

The second project (STEP-X) is a joint enterprise with Volkswagen and the institutes of Prof. Varchmin and Prof. Schnieder of the TU Braunschweig. Topic is to develop a structured development process for automotive systems. We are considering the comfort

system to be implemented on ECUs (electronic control units). We suggest a development process (based on commercial tools), leading from informal structured requirements to an architecture design, from which it is possible to generate code. In the intermediate analysis, we propose to use different versions of Statecharts.

Behaviour Behaviour

Luuk Groenewegen

Leiden University

Behaviour descriptions in terms of alternating sequences of states and transitions between them are quite common. In addition to such a (detailed) description, a global view of behaviours in terms of alternating sequences of phases and overlaps between them is presented. As the phases consist of behaviours themselves, such a global view is behaviour (of) behaviour.

Phases and their overlaps are formalized in Paradigm (PARallelism, its Analysis, Design and Implementation by a General Method) through the notions of subprocess and trap.

Based on the phases and their overlaps, communication and its coordination can be understood and analysed surprisingly easily. Moreover, process change or process evolution, even on the fly, can be similarly formulated, understood and analysed on the basis of phases and their overlaps.

Integration of Incomplete Behaviour Specifications

Martin Große-Rhode

TU Berlin

Viewpoint models of a software system specify some aspects of the system, seen from a specific point of view. The decomposition of the development process achieved in this way yields a reduction of the complexity by separation of concerns, orthogonal to the (horizontal) decomposition of the system and its models into components. By definition, viewpoint models are heterogeneous and incomplete,

thus an integration is required that defines how the models correspond to each other, how they yield a global specification of the whole system, and whether they are consistent with each other.

In the talk an integration approach is presented that is based on an abstract common semantic domain. Basically, the interpretation of all specifications in this domain yields the possibility for their integration. The domain is well structured in that it provides - beyond the extended labelled transition systems for the representation of entities like objects, components, processes, or systems - schemes for composition operations and development relations. Corresponding compositionality results are also shown.

Different applications of the integration approach to formal specification techniques are discussed. As a more detailed example UML statecharts and sequence diagrams are considered and the specific problems of the integration of incomplete (semi-formal) behaviour specifications are discussed.

Specifying and Executing Behavioral Requirements: The Play-In Play-Out Approach

David Harel

The Weizmann Institute of Science

Joint work with Rami Marelly

A novel requirements methodology for reactive systems is described, in which scenario-based requirements are "played in" directly from the system's GUI or some abstract version thereof, and behavior can then be "played out" freely, adhering to all the requirements. The approach is supported and illustrated by a tool we have built, which we call the play-engine. As the requirements are played in, the play-engine automatically generates a formal version in the language of live sequence charts (LSCs). As behavior is played out, the engine causes the application to react according to the universal ("must") parts of the specification; the existential ("may") parts can be monitored to check for successful completion. Play-in is a user-friendly high-level way of specifying behavior and play-out is a rather surprising way of working with a fully operational system directly from its inter-object re-

quirements. We have also implemented a "smart" play-out mode, whereby a successfully terminating superstep is computed on the fly by using model-checking. Thus, we employ formal verification techniques for driving the execution. The entire approach appears to be useful in many stages in the development of reactive systems, and could also pave the way to systems that are constructed directly from their requirements, without the need for intra-object or intra-component modeling or coding at all.

Detection of Conflicting Functional Requirements in a Use Case-Driven Approach - A Static Analysis Technique based on Graph Transformation

Reiko Heckel

University of Paderborn

Joint work with Jan Hendrik Hausmann and Gabi Taentzer

In object-oriented software development, requirements of different stakeholders are often manifested in use case models which complement the static domain model by dynamic and functional requirements. In the course of development, these requirements are analyzed and integrated to produce a consistent overall requirements specification. Iterations of the model may be triggered by conflicts between requirements of different parties.

However, due to the diversity, incompleteness, and informal nature, in particular of functional and dynamic requirements, such conflicts are difficult to find. Formal approaches to requirements engineering, often based on logic, attack these problems, but require highly specialized experts to write and reason about such specifications.

In this paper, we propose a formal interpretation of use case models consisting of UML use case, activity, and collaboration diagrams. The formalization, which is based on concepts from the theory of graph transformation, allows to make precise the notions of conflict and dependency between functional requirements expressed by different use cases. Then, use case models can be statically analyzed, and conflicts or dependencies detected by the

analysis can be communicated to the modeler by annotating the model.

An implementation of the static analysis within a graph transformation tool is presented.

A Probabilistic Extension of UML Statecharts

David N. Jansen

Universiteit Twente

Joint work with Holger Hermanns and Joost-Pieter Katoen

We introduce means to specify system randomness within statecharts. (In system randomness, the system's behaviour itself asks for a probabilistic description. The opposite is environment randomness, where the environment of the system generates input for the system according to a probability distribution.) To achieve this, we develop a general recipe to extend a statechart semantics with discrete probability distributions. The semantic structure we use is (a subset of) Markov decision processes. Properties of interest for probabilistic statecharts are expressed in PBTL, a probabilistic logiv for MDPs, and checked using the model checker PRISM. We apply the recipe to the semantics of (Eshuis and Wieringa; Requirements-level semantics for UML statecharts. FMOODS 2000), but it could also be applied to many other semantics.

Semantics of Statecharts: Steps

Sebastian John

TU Berlin

Is it not surprising that despite of the common use of the graphical language of the statechart formalism, the reactive systems which are modelled by them very differ in most cases? Even more surprising would be the fact that the differences arise even if only basic features of statecharts are employed. This talk reports varieties of plain statecharts interpretation and their fundamental concepts

introduced by the given semantics in the literature. The study is based on 3 principles: the analysis of conflicts, priority and basic steps - which are iterated with and without cumulation concepts. As a resume 72 varieties describing dynamic behaviour are figured out waiting to be further compared and structured to display their nature. This talk proposes in a stage of ongoing work, the notion of normal reactive systems to bring the behaviours modelled by steps in a hierarchy of abstraction relations.

The Enterprise Application Integration (EAI) Profile of the UML

Stuart Kent

University of Kent at Canterbury

The EAI UML profile is soon to be ratified as an OMG standard. Its purpose is to provide a language with a UML-like syntax for designing the architecture of message queuing systems . MQ systems are typically used for Enterprise Application Integration. The profile attempts to capture a paradigm which is prevalent in industry, but which is not a natural fit with UML, namely autonomous components communicating via (asynchronous) passing of messages through input and output ports connected by channels. The profile clearly illustrates a common practice in the use of UML: choose a subset of the notation; specialise (the syntax of) that notation with stereotypes and additional well-formedness constraints; give the syntax a meaning which suits the domain of interest, but might not be compatible with the declared meaning of UML, as far as that is described in the standard. The EAI profile itself is something which could benefit from formal treatment and seems to have many of the concepts already considered by theories of concurrency. It may also contain some surprises that could lead to the development of new theory. It could do with some good analysis tools to support it. A formal treatment may also lead to improvement in the language. The concept of profile poses a challenge to language engineers: how to define languages in families and generate/configure modelling and analysis tools from the definitions.

A Methodology for Specifying and Analyzing Consistency of Object-Oriented Behavioral Models

Jochen M. Küster

University of Paderborn

Joint work with Gregor Engels, Reiko Heckel and Luuk Groenewegen

Object-oriented modeling favors the modeling of object behavior from different viewpoints and the successive refinement of behavioral models in the development process. This gives rise to consistency problems of behavioral models. The absence of a formal semantics for UML models and the numerous possibilities of employing behavioral models within the development process lead to the rise of a number of different consistency notions. In this talk, we discuss the issue of consistency of behavioral models in the UML and present a general methodology how consistency problems can be dealt with. According to the methodology, those aspects of the models relevant to the consistency are mapped to a semantic domain in which precise consistency tests can be formulated. The choice of the semantic domain and the definition of consistency conditions can be used to construct different consistency notions. We show the applicability of our methodology by giving an example of a concrete consistency problem of concurrent object-oriented models.

Handling Change in OO-Systems

Ruurd Kuiper

TU Eindhoven

Joint work with Kees Huizing

In the setting of a Hoare-style proof system (pre/post/inv) for OO-systems we analyse the effect of changes to classes. We aim for two properties of the proof system:

- Data induction to maintain the invariants

- Compositionality in the sense that for proofs about one class only the specifications of other classes are needed, i. e., not their code.

We also aim for a property regarding the change: If the superclass is changes, keeping its contract, than the subclass also keeps its contract, i. e. verification does not have to be repeated for the subclass.

We approach this subject as a dynamic binding issue and provide a new notion of behavioral subtyping, reinforced behavioral subtyping, and a new notion of specification, cooperative contract. The fragile base class problem is shown to be a case in point; a concrete instantiation of it is shown as an example.

A Semantics First Approach to a Behavioural Subset of UML Statechart Diagrams

Diego Latella

CNR-CNUCE, Pisa

Joint work with Mieke Massink

In cooperation with S. Gnesi, F. Mazzanti, I. Majzik, I. I. Schieferdecker

In this talk a 'Semantics/Kernel first' approach to UML Statechart Diagrams has been presented. By 'semantics first' we mean an approach which is heavily based on a formal definition of the (semantics of) the notation. This way, interesting aspects of the notation can be and have been formally studied and verification/testing tools can be directly derived. This contributes in increasing the confidence on the notation and on its supporting tools. We proceeded by first formally defining the semantics of a behavioural subset of the diagrams, i.e. a 'kernel', and then using it for investigating interesting theories - like formal testing theory and formal conformance testing -, verification approaches - like LTL model-checking and ACTL model-checking - and useful extensions - like stochastic statecharts.

We defined our original semantics using hierarchical automata in a similar way as E. Mikk did, and we proved that the semantics

fulfills major requirements stated in the official UML definition. In each and every investigation we performed, the formal link to our original semantics as well as the correctness of the involved algorithms have been proven.

Once enough confidence will have been gained on the kernel, elements of the UML statecharts which are not currently included in our notation will be taken into consideration.

The 'semantics/kernel first' approach has already proven profitable in other fields of concurrency theory, like e.g. process algebra, abd we think it is worth using it also in the UML framework.

Hierarchical Automata as Model for Statecharts

Erich Mikk

Siemens AG

Joint work with Gerard Holzmann, Yassine Lakhnech and Michael Siegel

We suggest *extended hierarchical automata* (extended HA) as an intermediate format to facilitate the linking of new tools to the STATEMATE environment. The extended HA formalism uses single-source/single-target transitions as in usual automata (no inter-level transitions are admitted) and has a simple priority concept which facilitates computing the next step of an extended HA. So, the main idea is to devise a simple formalism with a more restricted syntax than statecharts which nevertheless allows to capture the richer formalism. Extended hierarchical automata, which are related to the Argos language, come with a simple operational semantics which simplifies the implementation of tools for this formalism.

The main technical problem is to devise a simple formalism which is nevertheless capable to model inter-level transitions. Inter-level transitions (possibly with multiple sources/multiple targets) are transitions which do not respect the hierarchy of states, i.e. those that cross borderlines of states. They can be understood as a powerful goto mechanism which allows to arbitrary change of control across the state hierarchy. The price of inter-level transitions is their intricate semantics in particular in combination with the priority mechanism of statecharts. Inter-level transitions spoil

a clean decomposition of a system into subsystems (since "dangling" transitions without source or target result) and thus denies a structural operational semantics for statecharts.

We introduce extended HA and their semantics. Then we present the translation of EHA into Promela/SPIN and SMV. This translation has been implemented in a tool-set which has been used for the verification of Production Cell and SAFER case study. The case studies show that the model checking Statecharts is feasible with SMV.

This talk gives an overview of the dissertation of Erich Mikk titled Semantics and Verification of Statecharts. This disseration was prepared while Erich Mikk was affiliated with Christian-Albrechts-University in Kiel/Germany.

Combining Specification Techniques for Processes, Data and Time

Ernst-Rüdiger Olderog

Universität Oldenburg

Joint work with Jochen Hoenicke

Complex computing systems exhibit various behavioural aspects, for example communication between components, state transformation inside components, and real-time constraints on the communications and state changes. Formal specification techniques for such systems have to be able to describe all these aspects. Unfortunately, a single specification technique that is well suited for all these aspects is not available. This observation has led to research into the combination and semantic integration of specification techniques.

We combine three well researched specification techniques for processes, data and time: Communicating Sequential Processes (CSP) [6, 10], Object-Z (OZ) [11] and Duration Calculus (DC) [12, 5]. In this talk the emphasis is on a smooth integration of the underlying semantic models and its use for verifying properties.

A class in the combined specification language CSP-OZ-DC is of the form $C = (I, P, D, T)$ with interface I, process part P, data part D, and timing part T. Its semantics is a timed process modelled

by a DC formula of two observables: tr for timed traces and Acc for timed acceptance sets. The definition proceeds in several steps: first the transformational semantics of the untimed combination CSP-OZ [2, 3] is reused for the parts I, P, D yielding an untimed CSP process, then this process is lifted to the timed setting yielding a DC formula in the observables tr and Acc, and finally the timing part T, being a DC formula, is conjoined.

This style of semantics definition can be used directly for a partially automatic verification of properties of CSP-OZ-DC specifications. The approach proceeds in two steps and succeeds for finite data types and certain patterns of timing restrictions. First the FDR model checker [9, 4] for CSP is used to generate for given class $C = (I, P, D, T)$ in this subset an untimed transition system with acceptance sets covering the parts I, P, D. Then this transition system is automatically transformed into a timed automaton $\mathcal{A}_C$ respecting the timing restrictions of the DC semantics. Properties of the class C can then be verified by applying the model checker UPPAAL [1, 8] to $\mathcal{A}_C$. This approach is illustrated with an example. It will be published in [7].

Acknowledgement. This research was partially supported by the German Research Council (DFG) under grant Ol/98-2.

[1] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. Uppaal – a tool suite for automatic verification of real-time systems. In R. Alur, T.A. Henzinger, and E.D. Sonntag, editors, *Hybrid Systems III – Verification and Control*, volume 1066 of *LNCS*, pages 232–243. Springer, 1997.

[2] C. Fischer. CSP-OZ: A combination of Object-Z and CSP. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-Based Distributed Systems (FMOODS'97)*, volume 2, pages 423–438. Chapman & Hall, 1997.

[3] C. Fischer. *Combination and Implementation of Processes and Data: From CSP-OZ to Java*. PhD thesis, Bericht Nr. 2/2000, University of Oldenburg, April 2000.

[4] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement: FDR 2*, Dec. 1995.

[5] M.R. Hansen and C. Zhou. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9:283–330, 1997.

[6] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[7] J. Hoenicke and E.-R. Olderog. Combining specification techniques for processes, data and time. In M. Butler and K. Sere, editors, *Integrated Formal Methods (IFM 2002)*, LNCS. Springer, 2002. To appear.

[8] K.G. Larsen, P. Pettersson, and Wang Yi. Uppaal in a nutshell. *Software Tools for Technology Transfer*, 1(1+2):134–152, 1997.

[9] A.W. Roscoe. Model-checking CSP. In A.W. Roscoe, editor, *A Classical Mind — Essays in Honour of C.A.R.Hoare*, pages 353–378. Prentice-Hall, 1994.

[10] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[11] G. Smith. *The Object-Z Specification Language*. Kluwer Academic Publisher, 2000.

[12] C. Zhou, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

The Temporal Logic of Distributed Actions

Wolfgang Reisig

Berlin

Modern Specification Languages, including OBJ, Z, LARCH, FOCUS, and TLA, describe a specification as a huge first order expression, decorated with grains of Temporal Logic. Some of them even more obey the very useful principles of "Composition is conjunction" and - even more important - "implementation (refinement) is implication". These principles require particular semantical models such as streams (as in FOCUS), or stuttering sequences (as in TLA).

We show that the well established notion of concurrent runs likewise supports the above principles. A minor extension of Lamport's TLA fits perfectly to concurrent runs. This new Temporal

Logic of DISTRIBUTED actions (TLDA) addresses notions such as locality, synchronization, and causality, hence fundamental notions of distributed behavior. At the same time, reasoning is frequently simpler (and never more difficult) than in TLA. For example, the notion of progress can be formulated without using (weak) fairness.

All this is gained at a very reasonable price: Just extend TLA by some few Boolean variables.

Model Checking Birth and Death

Arend Rensink

University of Twente

Joint work with Dino Distefano and Joost-Pieter Katoen

We propose Allocational Linear Temporal Logic (ALTL) as a formalism to express properties concerning the dynamic allocation (birth) and de-allocation (death) of entities, such as the objects in an object-based system. The logic is interpreted on so-called History-Dependent Automata (Developed by Montanari and others), extended with a symbolic representation for certain cases of unbounded allocation. We also present a simple imperative language with primitive statements for (de)allocation, with an operational semantics, to demonstrate the kind of behaviour that can be modelled. The main contribution of the paper is a tableau-based model checking algorithm for ALTL, along the lines of Lichtenstein and Pnueli's algorithm for LTL.

Inheritance and Mining of WF-nets

W.M.P. van der Aalst

Eindhoven University of Technology

Inheritance is one of the cornerstones of object-oriented programming and object-oriented design. The basic idea of inheritance is to provide mechanisms which allow for constructing subclasses

that inherit certain properties of a given superclass. In our case a class corresponds to a workflow process definition (i.e., a routing diagram) and objects (i.e., instances of the class) correspond to cases. In most object-oriented methods a class is characterized by a set of attributes and a set of methods. Attributes are used to describe properties of an object (i.e., an instance of the class). Methods symbolize operations on objects (e.g., create, destroy, and change attribute). The structure of a class is specified by the attributes and methods of that class. Note that the structure only refers to the static aspects of the interface. The dynamic behavior of a class is either hidden inside the methods or modeled explicitly (in UML the life-cycle of a class is modeled in terms of state machines). Although the dynamic behavior is an intrinsic part of the class description (either explicit or implicit), inheritance of dynamic behavior is not well-understood. In recent years, we have developed four notions of inheritance. On top of these notions we have developed transformation rules, transfer rules, and tools (most notably Woflan).

Inheritance is about comparing models. This is related to the topic of mining since in mining models are not compared with other models but with concrete traces of behavior. Contemporary workflow management systems are driven by explicit process models, i.e., a completely specified workflow design is required in order to enact a given workflow process. Creating a workflow design is a complicated time-consuming process and typically there are discrepancies between the actual workflow processes and the processes as perceived by the management. Therefore, we have developed techniques for (re)discovering workflow models. Starting point for such techniques are so-called workflow logs containing information about the workflow process as it is actually being executed. Unfortunately, it is not possible to (re)discover every workflow process. Therefore, we explore the class of workflow processes which can be discovered. Theoretical results demonstrate that most practical workflow processes fit into this class. The tool MiMo supports the (re)discovery of these processes.

Behavioural Subtyping Relations for Integrated Specification Formalisms

Heike Wehrheim

University of Oldenburg

Behavioural Subtyping Relations are concerned with behavioural conformance relationships between classes. They have to satisfy the *substitutivity* requirement imposed on types and their subtypes: a supertype object should be replacable by a subtype object without a client of the supertype noticing a difference.

There are a number of proposals for behavioural subtyping relations. They can broadly be classified as *state-based* and *behaviour-oriented* approaches. State-based subtyping relations are defined for state-based specification formalisms describing data and operations on data whereas behaviour-oriented subtyping relations are defined for specification languages describing the dynamic behaviour of systems. For *integrated* specification formalisms combining state-based and behaviour-oriented languages the question is then "which relation to use"? Ideally, one would like to be able to use the existing relations for the separate parts of a combined specification, and still be sure that an appropriate relationship also holds for the combination. This can in fact be achieved since state-based subtyping relations can be shown to induce behaviour-oriented relations when the state-based formalism is equipped with a behavioural semantics.

The investigation is carried out within the context of the specification language CSP-OZ, combining the state-based specification language Object-Z with the process algebra CSP.

Verification Support for Workflow Design with UML Activity Graphs

Roel Wieringa

University of Twente

Ph.D. Work of Rik Eshuis

We describe a tool that supports verification of workflow models specified in UML activity graphs. The tool translates an activity graph into an input format for a model checker according to a semantics we published earlier. With the model checker arbitrary propositional requirements can be checked against the input model. If a requirement fails to hold an error trace is returned by the model checker. The tool automatically translates such an error trace into an activity graph trace by high-lighting a corresponding path in the activity graph.

One of the problems that is dealt with is that model checkers require a finite state space whereas workflow models in general have an infinite state space. Another problem is that strong fairness is necessary to obtain realistic results. Only model checkers that use a special model checking algorithm for strong fairness are suitable for verifying workflow models. In the talk we show how we deal with these problems. We illustrate our approach with some example verifications.

Fujaba Statecharts

Albert Zündorf

Technical University of Braunschweig

This talk reports about the Statechart dialect of Fujaba, the UML case tool we have developed at University of Paderborn. These statecharts specify the reactive behavior of active objects running as concurrent threads (in a Java environment). These active objects have (a large) common memory/object structure. Due to the common shared memory, handling of sets of events in a single micro step is not feasible. Handling multiple events in parallel may cause multiple transitions to fire and execute concurrently. This

would require some kind of double buffering technique, in order to coordinate access to common data. Double buffering complex large object structures is unfeasable. Thus Fujaba Statecharts handle events strictly sequentially. This leads to a substantially different semantics of Fujaba Statecharts compared to Harel Statecharts.