WORKSHOP

INFORMATION SYSTEMS AS REACTIVE SYSTEMS

EDITORS
Hans-Dieter Ehrich
Ursula Goltz
José Meseguer

Contents

1 Preface

The specification, development and use of distributed information systems is an important research area with many practical applications. Examples for existing such systems are those owned by banks, airlines or governments. Moreover, the explosive growth of the Internet is evidence that a global information structure is developing, and expansion in the use of large distributed information systems can be predicted.

Information systems are *reactive* systems. Unlike transformational systems—that is, systems whose function is to transform some inputs into output results—reactive systems have an unlimited number of interactions with their environment. Basic notions of program correctness do not apply, and techniques for designing transformational systems do not easily carry over to reactive systems.

Non-distributed reactive systems, like conventional operating and database systems, are not easy to design, implement and restructure, a lot remains to be done to develop helpful high-level concepts, features, languages, methods and tools. However, existing products can already cope successfully with reasonably complex systems in practice. Formal methods from logic, algebra and process theory have been successfully applied, for instance in query and transaction processing, but they are not in general standard engineering practice yet.

Distributed reactive systems are still an order of magnitude more complex and difficult to specify, analyse, implement and reconfigure. They are not well understood, and methods for designing, implementing and reengineering them have not yet reached a level of mature and sound engineering practice.

A great challenge is to cope with *concurrency, synchronization, and communication* at an appropriate level of abstraction. Process theory provides useful theoretical concepts for concurrency and interaction. But those providing high-level declarative specification concepts, like temporal logic, tend to be sequential in nature, while those capable of expressing full concurrency, like Petri nets, tend to be operational: the level of abstraction is low. A possible approach is to adopt concepts from process algebras which provide compositional system descriptions and formalize dynamic behaviour. However, many questions then arise. These questions have been a central issue of this seminar.

In particular, the following topics have been adressed:

Modeling and design methods. Information systems involve data, objects and global processes. In conceptual modeling, these aspects have been addressed in isolation and various combinations, and the same holds for the conceptual modeling perspectives: system architecture, object classes, object behaviour, and object communication. Diagrammatic modeling approaches like OOA&D, OMT, OOSE and their recent amalgamation UML employ a multitude of specification concepts and techniques that are not always well integrated and sometimes not well understood. The lack of semantic consistency among UML diagrams has been criticized, and approaches to giving formal semantics to UML have been discussed.

Several contributions addressed the question of how to integrate formal specification techniques into pragmatic software development methods like UML or OOSE, respectively. Successful applications have been reported where the need arose to

augment UML by formal specification techniques like pre- and postconditions or temporal-logic constraints. OOSE has been integrated with object-oriented algebraic specification techniques by extending object and interaction diagrams with formal annotations.

Information systems are hardly ever built from scratch. One particular design issue that has been addressed is how to integrate existing databases into a federated database, concentrating on behaviour conflict issues.

Semantics of object-oriented specification languages. Describing the semantics of a formal language amounts to giving a map from syntactic constructs into a suitable semantic domain. For object-oriented specification languages, the domains under discussion were process algebras, Petri nets, event structures, the actor model, type algebra, and others. For example, it has been shown how an object-based language with intra-object concurrency can be translated into a process algebra.

Studies on how these semantic domains are interrelated have also been presented. The expressive power of process algebras and finite Petri nets has been compared by providing translations in both directions.

Among the language issues addressed were module composition, refinement and atomicity. The category-theoretic treatment of the parallel composition of objects with synchronous and asynchronous interaction has been discussed. Concerning refinement, it has been argued that a suitable notion of atomicity for method invocation allows the implementation of method invocation as action refinement in transition systems.

Logics for distributed systems. An attractive model for the execution of a distributed system is a *distributed process*, which has a time line for each location of the system and links the time lines by synchronization conditions like common events.

Rewriting logic, well known and quite successful in this field, has been augmented by adding synchronization conditions that are represented geometrically as the vertical sides of a tile. There is a multi-purpose specification language,Maude, that is based on rewriting techniques. Work is in progress to design an appropriate temporal logic which allows to formally verify the correctness of Maude programs.

Several contributions addressed the issue to bring distributed processes together with temporal logics which are widely accepted as an intuitive formalism for the description of the temporal order of events in the execution of a sequential program.

An overview was given of several generalisations of linear temporal logic for distributed processes and related models. A particular such logic has been presented which has the same expressive power as monadic second order logic or asynchronous automata, thus marking the boundary for automatic verification. Specific approaches were presented where local temporal logics are used for specifying object behaviour, and communication facilities are added for specifying interaction between concurrent sequential objects.

A notion of refinement for one of these logics has been presented and shown to be compatible with action refinement in process algebras.

A somewhat different approach was presented that is based on finite observations as communications between an observer and a system.

Application-oriented concepts in specification languages. Among the issues addressed were modules, real time, fault tolerance, software features, and internet applications.

For system development and design, concentration on objects as modules was criticized on the grounds that it only deals with system statics: parallel statements ought to be also considered as a kind of modules.

Real-time constraints set limits to when an action may or must occur or terminate, and how long it may take from a triggering event to the corresponding reaction. One real-time formalism was presented as it is used to specify systems in a Codesign approach. Another presentation gave a complete axiomatization of fully decidable propositional real-time linear temporal logics with past.

It has been shown that fault tolerant distributed services can be shown correct more easily by exploiting the causalities between actions. Related problems (database consistency, transaction protocols) can be treated similarly.

Features are software supplements providing additional functionality. The feature-interaction problem is the problem of detecting and resolving undesired interactions between features. It was argued that the feature-interaction problem is not solvable in general.

Mobility. For modelling large systems of communicating objects, it is indispensible to deal with a dynamically changing communication structure. In process algebras, this issue has been formally dealt with in the π-calculus. Results on characterising equivalences for the π-calculus have been presented. The fusion calulus, which was also presented at the seminar, has recently been developed starting from the π-caluculus; it is at the same time conceptually simpler and more expressive. A kernel programming language for mobility, called KLAIM, was presented.

A central topic of concern in talks as well as in many informal discussions was the newly developped specification formalism UML (Unified Modelling Language). Both its usefulness for applications and the problems of giving semantics to various constructs of UML were intensively discussed.

Until this seminar, there was little interaction between the communities concerned with information systems and concurrency theory. One outcome of the seminar is that there is a definite need for interaction; new contacts have been established which will turn out to be fruitful in the future. 34 participants have accepted the invitation, among them scientists from China, the US, Israel and several European countries. The seminar was sponsered by the TMR program. This allowed to invite several young scientists giving very interesting input to the seminar; the seminar yielded valuable new contacts for them.

The program of the seminar was intense and stimulating; it comprised 28 talks, the abstracts are recorded in this report in alphabetical order.

Hans-Dieter Ehrich
Ursula Goltz
José Meseguer

Acknowledgements

Many thanks to the Dagstuhl team and to Thomas Gehrke and Juliana Küster Filipe for their help in the preparation and organisation of the seminar. On behalf of all the participants, the organisers would also like to thank the staff of Schloß Dagstuhl for providing an excellent environment.

2 Workshop Programme

Monday 16

Chair: Ursula Goltz

9.00 Hans-Dieter Ehrich: Integrating Specification Techniques for Information Systems
9.40 Carlos Caleiro: Combining Logics for Distribution

10.20 COFFEE BREAK

Chair: Hans-Dieter Ehrich

10.50 Rocco De Nicola: KLAIM: A Kernel Language for Agents Interaction and Mobility
11.30 Thomas Gehrke: A Semantics for a Language with Intra-Object-Concurrency

12.10 LUNCH BREAK

Chair: Rocco De Nicola

14.10 Ralf-Detlef Kutsche: Object-oriented Modeling and/or/vs. Formal Specification of Dynamics in Information Systems
14.50 Marcus Klar: Precise Metamodeling for the Definition of Object-Oriented Modeling

15.30 COFFEE BREAK

Chair: Gunter Saake

16.10 Joachim Parrow: The Fusion Calculus
16.50 Arend Rensink: Atomicity in Object-Based Systems

Tuesday 17

Chair: Shmuel Katz

9.00 Martin Wirsing: A Formal Approach to Object-Oriented Software Engineering
9.40 Pedro Resende: Observational Logic for the Specification of Concurrent Systems

10.20 COFFEE BREAK

Chair: Carolyn Talcott

10.50 Chi-Song Tang: Modules for Large-scale Programs from Semantic Point of View
11.30 Stefan Beroff: Real-Time Specifications Formalisms with States: Application to VHDL Semantics

12.10 LUNCH BREAK

Chair: Pierre-Yves Schobbens

14.10 Peter Niebert: Location-based Temporal Logics
14.50 Michaela Huhn: Location-based Action Refinement

15.30 COFFEE BREAK

16.10 DISCUSSION

Wednesday 18

Chair: Joachim Parrow

9.00 Mark Ryan: Integration and Verification of Software Features
9.40 Antonio Ravara: A Type Algebra for Concurrent Objects

10.20 COFFEE BREAK

Chair: Martin Wirsing

10.50 Shmuel Katz: Convenient Executions for Fault Tolerant Distributed Services
11.30 Ulrike Lechner: Towards a Computational Media Metaphor

12.10 LUNCH BREAK

EXCURSION

Thursday 19

Chair: Rob van Glabbeek

9.00 Jose Meseguer: ¿From Process and Term Tile Logic to Rewriting Logic
9.40 Grit Denker: On Protocol Specification with Maude

10.30 COFFEE BREAK

Chair: Mark Ryan

10.50 Juliana Küster Filipe: Category-Theoretic Semantics of Modular Concepts
11.30 Rob van Glabbeek: On the Relative Expressiveness of Petri Nets, Process Algebras and Event Structures

12.15 LUNCH BREAK

14.10 DISCUSSION

15.30 COFFEE BREAK

Chair: Bernhard Thalheim

16.10 Carolyn Talcott: Composable Semantics Models for Actors
16.50 Michael Schrefl: A Taxonomy of Behavior Heterogeneity in Federated Database Systems

---

Friday 20

Chair: Gilles Bernot

9.00 Pierre-Yves Schobbens: Specification of Real-Time Reactive Systems

9.40 Gianna Reggio: A Method to Capture Formal Requirements: An Application to the INVOICE Case Study

10.20 COFFEE BREAK

Chair: Jose Meseguer

10.50 Paola Quaglia: Name-Dependency and Congruences in the π-Calculus

11.30 Jan Friso Groote: Proving Modal Properties for μCRL

12.10 LUNCH

END

3 Collected Abstracts

Real-Time Specifications Formalism with States: Application to VHDL Semantics

Stefan Beroff
Université d'Evry
beroff@lami.univ-evry.fr

We introduce a notion of time and implicit states into the first-order specification framework, both to deal with temporal constraints and to specify functions whose semantics are time and state dependent. This real-time formalism is then used to specify systems in a Codesign approach, that is, systems which can be realized by overlapped description of hardware parts and software parts. Actually, specifications are used as input for an automatic partitionning tools which realizes the implementation choices and produces a system composed by VHDL descriptions (for hardware) and C code (for software).

In a first part, we focus on the real-time specification formalism with states, describing its syntax and its semantics. We show how concurrency is handled during term evaluations. We also take into accout a maximal allowed degree of concurrency during term evaluation. In a second part, we focus on an axiomatic semantics for the subset of the Hardware Description Language VHDL used for simulation. We then define the translation of a VHDL program to a real-time specification. The aim of this translation is to switch from the VHDL world to our real-time logic world, in order to prove that the hardware implementation is correct with respect to the abstract specification of the system. A short example is given to illustrate the approach.

Joint work with Marc Aiguier

Combining Logics for Distribution

Carlos Caleiro
Lisbon Institute of Technology
ccal@math.ist.utl.pt

We take the abstract view of a distributed system as a collection of independent subsystems that can be assessed by localities. Each subsystem has its own local behaviour and associated logic, but reference to properties of neighbour subsystems is possible due to communication.

We briefly review a distributed temporal logic (dtl) for synchronously communicating sequential systems where, at each locality, a same version of weakly branching temporal logic is assumed. We then generalize this idea by allowing an arbitrary logic of behaviour at each locality. We also drop the sequentiality condition for subsystems thus paving the way for hierarchical composition. As a general model of behaviour we consider prime event structures.

We discuss alternative forms of combination coping also with asynchronous communication. These would generalize the combination mechanisms that underlie the logics for communicating agents by Lodaya, Mukund, Ramanujam and Thiagarajan.

We conclude with a conjecture on generalizing to the mechanism for synchronous combination an equiexpressibility result for a calling-only restriction of dtl.

KLAIM: A Kernel Language for Agents Interaction and Mobility

Rocco De Nicola

Università di Firenze
denicola@dsi.unifi.it

We investigate the issue of designing a kernel programming language for Mobile Computing and describe Klaim, a language that supports a programming paradigm where processes, like data, can be moved from one computing environment to another. The language consists of a core Linda with multiple tuple spaces and of a set of operators for building processes. Klaim naturally supports programming with explicit localities. Localities are first-class data (they can be manipulated like any other data), but the language provides coordination mechanisms to control the interaction protocols among located processes. The formal operational semantics is useful for discussing the design of the language and provides guidelines for implementations. Klaim is equipped with a type system that statically checks access rights violations of mobile agents. Types are used to describe the intentions (read, write, execute, etc.) of processes in relation to the various localities. The type system is used to determine the operations that processeswith the declared intentions and whether they have the necessary rights to perform the intended operations at the specific localities. Via a series of examples, we show that many mobile code programming paradigms can be naturally implemented in our kernel language. We also present a prototype implementation of Klaim in Java.

(To appear in IEEE Transactions on Software Engineering)

On Protocol Specification with Maude

Grit Denker

SRI International Computer Science Laboratory
denker@csl.sri.com

Maude is a multi-purpose specification language that integrates functional and object oriented approaches among others. Maude is based on rewriting techniques. As such its underlying logic is concurrent term rewriting logic. Along with Maude comes a prototyping environment which allows to execute Maude specifications. Due to the reflectivity of rewriting logic it is possible to express strategies for executing Maude specifications in Maude itself. Using strategies one can program a "breadth-first-search" to exhaustively determine all possible runs for a given initial state using the rewrite rules of the Maude specification. This way model checking (in case of finite systems) as well as narrowing and analysis up to a certain depth are possible.

We take advantage of this framework to formalize the Needham-Schroeder Public-Key Protocol (NSPK). NSPK is a cryptographic protocol for the authentication of pairs of agents in a distributed computer system. Our Maude specification allows multiple sessions for an agent. Moreover, an agent may play different roles (e.g., initiator or responder) in its sessions. We used the Maude engine to validate our specification. Additionally we give the most general specification of an intruder. An intruder may observe, intercept or fake messages. Introducing a fake message into the system either means replaying a previously observed or intercepted message or creating a new message using the nonces known by the intruder. There is well-known attack to the NSPK-protocol proposed by Gavin Lowe in 1995. Running our enriched specification "NSPK+Intruder" on the Maude Machine delivers this attack.

Currently we are working on designing an appropriate temporal logic which allows us to formally verify the correctness of the repaired NSPK. Moreover, we intend to apply the Maude formalism to other protocols, including other areas such as reliable broadcast protocols for dynamically changing topologies.

Joint work with Jose Meseguer

Integrating Specification Techniques for Information Systems

Hans-Dieter Ehrich
Technische Universität Braunschweig
HD.Ehrich@tu-bs.de

Information systems involve data, objects and global processes. Objects have local attributes and actions. In conceptual modeling, these aspects have been addressed in isolation and various combinations. The same holds for the conceptual modeling perspectives: system architecture, object classes, object behaviour, and object communication.

Diagrammatic modeling approaches like OOA&D, OMT, OOSE and their recent amalgamation UML employ a multitude of specification concepts and techniques that are not always well integrated and sometimes not well understood. These approaches are semiformal and incomplete. Thus, not all relevant aspects of an information system can be specified with the necessary precision and completeness.

Here is where the TROLL project pursued in Braunschweig takes a run. TROLL consists of languages (graphic and textual) with formal syntax and semantics, a method and a workbench. The approach integrates ideas from object orientation, abstract data types, conceptual modeling, behaviour modeling, specification of reactive systems, and concurrency theory.

TROLL differs from other languages and methods especially by its approach to communication: there is no explicit global process description, all communication is expressed by local calling of actions in other objects. In theoretical work on TROLL semantics, distributed temporal logics are being investigated using sequential linear temporal logic for specifying local object behaviour and adding communication expressions for specifying interaction. In a practical project, integration of the TROLL approach with Petri nets for specifying global system dynamics is being investigated.

A Semantics for a Language with Intra-Object-Concurrency

Thomas Gehrke
Universität Hildesheim
gehrke@informatik.uni-hildesheim.de

In this talk we introduce an abstract object-based language with intra-object-concurrency based on *Object REXX*. We define the formal semantics of the language by translating object systems into terms of a process algebra. Instead of identifying objects with processes, we distinguish in the semantics active processes from passive objects. Method invocation is translated into process calls. Furthermore, weak bisimulation is used as an equivalence relation on object systems.

On the Relative Expressiveness of Petri Nets, Event Structures and Process Algebras

Rob van Glabbeek
Stanford University
rvg@cs.stanford.edu

In this talk I consider mappings from expressions in system description languages like CCS and CSP to Petri nets and vice versa. I recall two methods for associating a Petri net to such a process expression: the standard compositional approach where recursion is dealt with using fixed point techniques, and a variant due to Ursula Goltz in which recursion is implemented by redirecting arcs backwards to initial places. The former approach always yields a so-called safe flow net; recursion typically gives rise to infinite nets. The latter approach works for the subclass of process expressions where in the scope of a recursion construct only prefixing, independent parallel composition and guarded choice is allowed; but always yields finite S/T-nets. The two approaches are identical for recursion-free process expressions; in general they agree up to fully concurrent bisimulation equivalence. Here I extend the latter approach to deal with unguarded recursion. I also show that for every finite S/T-net there exists a process expression in the mentioned subclass whose semantics, up to fully concurrent bisimulation, is that net. Moreover, every finite safe flow net is denoted up to event-structure isomorphism by a recursion-free process expression.

Proving Modal Properties for μCRL

Jan Friso Groote
CWI Amsterdam
JanFriso.Groote@cwi.nl

μCRL is process algebra in the ACP style together with abstract datatypes. This very basic language is sufficiently expressive to allow the description of almost any distributed system or protocol. Moreover, the language is sufficiently basic to function as the carrier of the development of verification techniques.

In recent years many systems have been verified, in the style *specification* equals *implementation* and as a result a theory, called the *cones and foci* method, has been developed that makes it relatively easy to prove properties of this kind.

One of the core elements turned out to be the development of so-called *linear process operators* or *linear process equations*. Before coming to the heart of every verification, the description of every system is transformed into such a linear form.

We (incl. Radu Mateescu) have recently extended this approach to prove modal formulas about linear process expressions. The modal formulas are given as formulas in the

action based modal mu calculus extended with data. From such a formula and an LPO a set of parameterized boolean fixed point equations are derived. There are two ways to show that these hold for the initial state of a system. The first way is by approximating the fixed point via a standard iteration algorithm. This boils down to repeatedly prove (automatically) identities about data. The other way is to use derivation rules (we found two rules, one for minimal and one for maximal fixed points).

Location-based Action Refinement

Michaela Huhn
Universität Karlsruhe
huhn@informatik.uni-karlsruhe.de

We consider location-based action refinement as a technique to formally capture the hierarchical design of systems with a fixed number of sequential agents. By restricting ourselves to this system class we gain – compared to other approaches known from the literature – a more liberal refinement notion. An abstract action shared by several components may be refined differently for each component. Moreover, the system architecture is respected by location based refinement, which is most relevant in later design stages. Location based action refinement is integrated in a process algebraic specification language for systems of sequential agents. We give congruence results for all operators of our language with respect to bisimulation semantics. In particular the refinement operator is compositional on the level of components, and coincides with sequential procedure call. Next we briefly discuss the restrictions needed to ensure compositionality also if location based refinement is considered as an operator on the system level.

In a second part of the talk we focus on the inheritance of temporal logic requirement specifications under location based action refinement. Since action refinement modifies the system interface, in general, the temporal properties have to be transformed according to the refinement. For regular refinement functions the transformation of temporal properties stated in νTrPTL, a location based linear time μ-calculus, can be done automatically. Therefore we substitute the abstract actions within the modalities by their local refinements, yielding PDL-like modalities, which are canonically decomposed into μ-calculus modalities. Under reasonable restrictions – in particular the refinement has to be deadlock-free and ensure all-or-nothing behaviour – a temporal property is satisfied for the abstract system if and only if the transformed formula holds for the refined system.

Convenient Executions for Fault Tolerant Distributed Services

Shmuel Katz
Technion, Haifa
katz@cs.technion.ac.il

Execution sequences can be considered equivalent for many properties if they differ only in that independent actions are executed in a different order. By exploiting this equivalence, correctness reasoning and refinement can be made easier for problems that involve overlap of concurrent activities. This is especially true for distributed fault-tolerant services, where even the specification can be complex. The group membership problem is taken as a prime example, where distributed processes each have a local view of the group of processes considered nonfaulty. The nonfaulty processes are required to have views that are consistent and are updated to identify and remove processes that become faulty.

The allowed faults in the system must be part of the specification, modelled as introducing additional failure actions, or extending the semantics of existing operations (e.g., a *send* operation that can either lose the message or properly deliver it). Assumptions on the frequency or number of faults are also common.

In refinements of high-level operations, such as updating the group views, first convenient lower level executions are shown to implement execution sequences of upper level operations. The convenient executions at the lower level are precisely those where the lower level operations that implement a higher level one are all done sequentially, with no other lower level operations interspersed. These are legal lower level executions, even if they are unlikely to occur in practice because the operations are distributed in a collection of asynchronously executing processors. A mapping function from each convenient execution to some abstract one is generally simple and iterative. The invariants needed are more natural and stronger than for all the lower level execution sequences, since they need hold only for states in the convenient executions.

Then in the second stage, all other execution sequences are shown to be equivalent to one of the convenient ones. The equivalence maintains the ordering of all causally dependent events, but allows independent events to occur in different orders. This stage could be considered as a 'loosening' of the ordering imposed by the convenient executions. A proof rule using a well-founded set and a measure function from sequences to the well-founded set is used to show that every nonconvenient sequence is equivalent to one with a smaller measure function, and convenient sequences have a minimal value under the measure.

The advantage of this separation is that different kinds of reasoning and induction can be used for the two aspects, resulting in simpler, more natural proofs. This approach is valuable for related problems such as database consistency, transaction protocols, and interactions of active agents.

Precise Metamodeling for the Definition of Object-Oriented Modeling Languages

Marcus Klar
Fraunhofer-Institut für Software- und Systemtechnik ISST, Berlin
Marcus.Klar@isst.fhg.de

A widely used and generally accepted technique in modern software engineering is the combination of different models (or views) for the description of software systems. The primary benefit of this approach is to model only related aspects (like structure or behavior). For this principle, called *Separation of Concern*, different specialized techniques mostly of diagrammatic nature have been developed. These models are not independent and they are semantically overlapping and thus it is necessary to state how these models are related. Moreover, it is appropriate to determine how these models are constituting the whole system.

In this talk we present a new approach for the investigation of the relationships between different models. We use a metamodel for this task and give a precise meaning to each model and the relation between them using the object-oriented specification language Object-Z. In contrast to existing approaches we investigate abstract syntax, static semantics, and dynamic semantics within one single formalism. Furthermore, the metamodel is extended by aspects of a programming language. We show how this approach can be used for the definition of the *Unified Modeling Language*.

Category-Theoretic Semantics of Modular Concepts

Juliana Küster Filipe
Technische Universität Braunschweig
j.kuester-filipe@tu-bs.de

We aim to provide a category-based semantics to specifications in-the-large. By enriching TROLL with a module concept we obtain a further structuring mechanism which is essential for specification in-the-large. A module should be understood as a parameterisable system part with intra-module concurrency and interfaces through which it can communicate with other modules. Module interaction can be synchronous or asynchronous.

Modules are modelled by (labelled) prime event structures. I present an extension of a morphism definition between prime event structures which lets total inclusions between event structures be morphisms. The event structures with the new definition of morphism define a category which is complete (has pullbacks) but is not cocomplete (pushouts do not always exist).

When modelling module synchronisation categorically we wish to have a pushout construction. Pullbacks can be seen as a constrained product and are far too much.

Pullbacks, however, are what we need in case of full synchronisation. The idea is now to use pullbacks whenever they provide the intended result, and special pushouts that are sure to exist.

By considering a restriction on the synchronisation part of the models, their synchronisation can be obtained by a pullback. By building the resulting two underlying pullbacks, and at the end a pushout we obtain the expected synchronisation of models.

This is ongoing work, and some details concerning the properties of the new category have to be checked. The application of such a construction to parameter actualisation and refinement is being investigated.

Object-oriented Modeling and/or/vs. Formal Specification of Dynamics in Information Systems

Ralf-Detlef Kutsche

Fraunhofer-Institut für Software- und Systemtechnik ISST, Berlin

Ralf.Kutsche@isst.fhg.de

Developing large, distributed information systems, one has to reflect many requirements w.r.t. the integration of legacy components and the construction of transparent federation and interoperation architectures, but also w.r.t. software quality and reliability. The paradigm of a 'continuous software engineering', i.e. consistent evolution of models, specifications, code and documentation in a continuous process of forward, reverse and re-engineering, gives a general setting for the methodologies needed.

This talks adresses requirements in the modeling and specification area, here focussing on the dynamic aspects in search for powerful, practical, scalable but also semantically well-founded languages in the analysis and design with appropriate tool support.

Based on several experiences in different application domains, the author encourages the application of the Unified Modeling Language UML within an object-oriented methodology for the development of such information system, emphasizing the following key aspects:

1. modeling 'use cases' for the description of typical scenarios in the business processes, applying different diagramming techniques according to the required levels of abstraction (use case, activity and interaction diagrams), in order to obtain a detailed insight into the operations;

2. using class diagrams in a classical way for structural modeling, including horizontal and vertical connection diagrams as a supplementary technique;

3. modeling the intra- and inter-object dynamics by a well-chosen combination of UML's state diagrams and interaction or collaboration diagrams û depending on the viewpoints taken during analysis or design.

For a more rigorous handling of particular properties of the system, e.g. critical aspects w.r.t. time behaviour or the assertion of integrity constraints, additional formal specifications of those parts can be useful, e.g. adding pre- and post-conditions, temporal constraints and others. UML provides means to enrich the diagrams by such (and other, less formal) annotations.

All these techniques have been demonstrated successfully in practical application. However, the semantical integration and consistency of the different UML diagram types, and their combination with logical formalisms, still is insufficient. Here, contributions from the theoreticians side will be welcome.

Towards a Computational Media Metaphor

Ulrike Lechner
University of St. Gallen
ulrike.lechner@mcm.unisg.ch

Computer science has changed and empowered the notion of a medium as a carrier of information. The computer is the first active, artificial medium and the medium Internet makes information ubiquitous. The information carried by the medium Internet changes and evolves along with the knowledge of agents attached to it. We establish a new notion of media, the computational media metaphor. We employ concepts from computer science, the technology that triggers and builds new media to describe those media. The descriptions are specifications according to which the platforms of the media can be constructed by computer science. Media are modeled as spheres of multi-agent systems. The computational media metaphor comprises five components: channels to transport information, agents to process information, logics to represent information, organization to describe behavior and interaction of agents and channels, possible worlds to model the way media are constructed from the components.

We instantiate the general concept computational media metaphor with a computational framework consisting of Rewriting Logics, *mu*-calculus and deontic logic and discuss the properties of the framework.

Our example is the computational medium NetAcademy, a medium for the scientific community. It provides facilities for scientific publishing as well as scientific discourse and it is designed to renew the processes of scientific publishing for the new media. It relies on the Internet technology and implements the scientific processes for quality management.

The computational media metaphor is determined to be a interface that allows management science to express requirements such that the media can be build and a interface that allows new concepts from computer science to be employed in the media. Our goal is to build media, based on a profound theoretical basis, such that the media can be created,

and need not to be programmed by their users.

Joint work with Beat Schmid

From Process and Term Tile Logic to Rewriting Logic

José Meseguer
SRI International Computer Science Laboratory
meseguer@csl.sri.com

Tile logic generalizes rewriting logic by adding synchronization conditions that are represented geometrically as the vertical sides of a tile. Model-theoretically, this is the generalization from 2-categories, where standard rewrites are represented as 2-cells, to double categories, where double cells correspond to tiles. However, in both cases there is additional algebraic structure. For rewriting logic, the usual added structure is the cartesian structure of finite 2-products, which accounts very naturally in a Lawvere-style way for terms and substitutions. This work proposes algebraic axioms for two similar algebraic structures for double categories, yielding the desired categorical models for important examples of tile logic. The algebraic structures in question are: symmetric monoidal double category, and cartesian double category. They respectively yield the process and term variants of tile logic. Another important topic studied is the mapping from these versions of tile logic into rewriting logic. The interest of mapping these formal systems into rewriting logic is the possibilities for executable specification and machine-supported formal analysis that become then available using rewriting-logic-based language implementations such as Maude. In the definition of the symmetric monoidal and cartesian algebraic structure for double categories, and also in the mapping to rewritign logic, the algebraic foundation we use is partial membership equational logic. This makes the definitions themselves and the different adjoint functor constructions involved easy to axiomatize and work with.

Joint work with Roberto Bruni and Ugo Montanari (U. Pisa)

Location-based Temporal Logics

Peter Niebert
Universität Hildesheim
niebert@informatik.uni-hildesheim.de

Distributed systems are characterised by the interaction of otherwise independent local components. An attractive model for the execution of such a system is a *distributed process*, which has a time line for each location of the system and links the time lines by common events. On the other hand, temporal logics are accepted as intuitive formalism for the description of the temporal order of events in the execution of a sequential program. The goal of this work is to bring distributed processes together with temporal logics.

In the talk we give an overview of several generalisations of linear temporal logic for distributed processes and related models. In particular, the logic νTrPTL from the speaker's thesis is presented, which is the only known modal temporal logic for distributed processes with the same expressive power as monadic second order logic or asynchronousaus automata. Thus, this logic marks the boundary of logics suitable for finite-state methods for distributed processes.

We conclude with remarks on the limited prospects of generalising our results to branching time logics.

The Fusion Calculus

Joachim Parrow
SICS, Stockholm
joachim@sics.se

We present the fusion calculus as a significant step towards a canonical calculus of concurrency. It simplifies and extends the pi-calculus.

The fusion calculus contains the polyadic pi-calculus as a proper subcalculus and thus inherits all its expressive power. The gain is that fusion contains actions akin to updating a shared state, and a scoping construct for bounding their effects. Therefore it is easier to represent computational models such as imperative and concurrent constraints formalisms. It is also easy to represent the so called strong reduction strategies in the lambda-calculus, involving reduction under abstraction. In the pi-calculus these tasks require elaborate encodings.

The dramatic main point is that we achieve these improvements by simplifying the pi-calculus rather than adding features to it. The fusion calculus has only one binding operator where the pi-calculus has two (input and restriction). It has a complete symmetry between input and output actions where the pi-calculus has not. There is only one sensible

variety of bisimulation congruence where the pi-calculus has at least three (early, late and open). Proofs about the fusion calculus therefore are shorter and clearer.

Name-dependency and Congruences in the π-Calculus

Paola Quaglia
BRICS, Aarhus
quaglia@brics.dk

Late and early π-calculus bisimulations are not preserved by name instantiation. Full congruences are defined by requiring the corresponding ground bisimilarity under all name substitutions.

We show that it is possible to improve on those infinitary definitions by sticking to a finite number of carefully identified substitutions, and hence, by resorting to only a finite number of ground bisimilarity checks.

Finitary alternative characterizations of π-congruences are then investigated within both the ground and the non-ground $\pi\xi$-calculus, a CCS-like process algebra whose ground version has been proved to coincide with ground π-calculus. The $\pi\xi$-calculus perspective allows processes to be explicitly interpreted as functions of their free names. As for the ground version of the $\pi\xi$-calculus, we exploit λ-closures of processes, so inducing the effective generation of the substitutions necessary to infer non-ground equivalence. In the non-ground case, a more promising call-by-need discipline for the generation of the wanted substitutions is used. This last strategy can also be adopted to show a coincidence result with open semantics.

A Type Algebra for Concurrent Objects

Antonio Ravara
Lisbon Institute of Technology
amar@math.ist.utl.pt

To capture the non-uniform service availability of concurrent objects we define a type system for TyCO, a name-passing calculus. A type reflects the dependency between the interfaces and the internal states of an object, thus partially specifying the behaviour of that object. This notion of type loosens the usual message-not-understood communication

error by demanding only weak fairness in the treatment of messages, in the sense that messages that can be accepted sometime in the future do not cause errors. The type system ensures that typable processes are not locally deadlocked, and do not run into errors.

Joint work with Vasco Vasconcelos (U. Lisbon)
Available from: http://www.cs.math.ist.utl.pt/s84.www/cs/amar.html

---

A Method to Capture Formal Requirements: An Application to the INVOICE Case Study

Gianna Reggio
Universita' di Genova
reggio@disi.unige.it

In the last years myself in cooperation with other people have developed various specification formalisms for systems which can be named concurrent, parallel, reactive, ..., and applied to various case studies also in projects in cooperation with industry. While the people forwarding the case studies have found the formal specifications produced with our cooperation satisfactory, they were asking "how can we produce such specifications?" and "how can be such specifications integrated into a development process?" That has lead us to try to derive from a specification formalism for systems, satisfactory from a formal point of view a method for supporting the development process. Some tasks have been already worked out; here we present a first proposal for a method for giving requirement specifications (thus requirements capture and specification) and a graphic presentation for such specifications, applied to the INVOICE case study.

Requirement specifications should express the fundamental characteristic of something that we have to develop; and here we consider only the development of products that are changing along the time, and call them *systems*. Our requirement specifications follow an axiomatic (or better property oriented) style, and use a variant of the branching-time temporal logic. But here the emphasis in on "HOW" to get the formal specification. The method provides precise instructions guiding the user to find, after some analysis of the system, in an exhaustive way all sensible properties. The resulting specification will have a very precise format, and that could be positive, since it helps

- to read the specification;
- to easily modify the specification (and that could be used to support the evolution of the system);

- to develop techniques for verification, if we know exactly the subset of formulae that will be used, then the verification task may be simpler.

The guidelines are derived from our experience in specifying; the application to this on and to other simple case studies seems to be positive.

In this paper we present only the part of our method concerning requirement specifications of simple (non-structured) systems using a basic specification formalism, because that is what we need for INVOICE.

Atomicity in Object-based Systems

Arend Rensink

Universität Hildesheim
rensink@informatik.uni-hildesheim.de

The aim of this research is to establish a theory of specification and verification for object-based systems, based on the assumption that on an abstract level, method invocations are *atomic*. Atomicity has two aspects: all-or-nothing behaviour (any invocation is either refused or erminates successfully) and interference freedom (concurrent invocation has the effect of sequential execution). The atomicity assumption enables us to describe object behaviour using labelled transition systems, where the labels are methods.

The vertical decomposition of an object involves the implementation of methods as procedires calling lower-level objects. We study a otion of correctness for such decompositions, and present an abstraction algorithm that, if it runs successfully, is conjectured to guarantee correctness.

Observational Logic for the Specification of Concurrent Systems

Pedro Resende

Lisbon Institute of Technology
pmr@math.ist.utl.pt

A finite observation is understood to be a communication between an observer and a system, performed in finite time and involving a finite amount of information. We follow

Abramsky and Vickers in taking finite observations to form quantales, and introduce observational logic, whose models are labelled transition systems and whose formulas are equations of the form $a = b$ or $a =' b$, meaning that the (finite) observation a is the same as the observation b, or that the (finitely) observable property a is the same as the observable property b, respectively. The talk consists of three parts. In the first part the basic notions of finite observation, quantale and observational logic are introduced. The second part is devoted to three examples of specification of labelled transition systems, related respectively to interleaving semantics, causal semantics and real-time. Finally, in the third part we study vertical and horizontal modularity both at the syntactic and semantic levels. In particular, horizontal modularity is based on a notion of parallel composition of independent systems, vertical modularity is based on a notion of implementation with refinement of observations, and several compositionality results are presented.

Integration and Verification of Software Features

Mark Ryan
University of Birmingham
M.D.Ryan@cs.bham.ac.uk

The *feature-interaction problem* is the problem of detecting and resolving undesired interactions between features of systems. We argue that the feature-interaction problem is not solvable in general, since there is no *a priori* distinction between desired and undesired interactions. We propose an approach to the feature-interaction problem with the following characteristics:

- Features are treated as first-class citizens during the development phase;
- A method for integrating a feature into a system description is described. It allows features to override existing behaviour of the system being developed;
- A prototype tool has been developed for performing the integration;
- Our approach allows interactions between features to be witnessed.

In principle, our approach is quite general and need not be tied to any particular system description language. In the talk, however, we develop the approach in the context of the SMV model checking system.

As a case study, we analyse the *lift system* and its features.

Specification of Real-Time Reactive Systems

Pierre-Yves Schobbens
Universiti Notre-Dame de la Paix
pys@info.fundp.ac.be

We present a complete axiomatization of fully decidable propositional real-time linear temporal logics with past: the Event Clock Logic (ECL alias SCL) and the Metric Interval Temporal Logic (MITL). The completeness proof consists of an effective proof building procedure for ECL. It is structured to yield a similar axiomatization and procedure for interesting fragments of this logic: the linear temporal logic of the real numbers (LTR), the fragment of ECL with only one clocks, with only past clocks.

A Taxonomy of Behavior Heterogeneity in Federated Database Systems

Michael Schrefl
Johannes-Kepler Universitaet Linz
pys@info.fundp.ac.be

The integration of component databases into a federated database requires to resolve semantic heterogeneities (or conflicts) between the schemata of the component databases. Systematic classifications of such conflicts have been established for the relational data model and for object structure (e.g. by W. Kim). Based on these works we introduce a taxonomy of behavior heterogeneity for the integration of object-oriented databases. Two comparable object-types may exhibit behavioral conflicts of the following kinds:

A. Method-vs-method conflicts
    1. One-to-one method conflicts
        (a) Method name conflicts,
        (b) Method interface conflicts (missing parameters, missing but implicit parameters),
        (c) Pre- and postcondition conflicts
    2. Many-to-many method conflicts;

B. Parameter-vs-parameter conflicts
    1. One-to-one parameter conflicts
        (a) Parameter name conflicts,

(b) Parameter constraint conflicts

2. Many-to-many parameter conflicts;

C. Method-vs-parameter conflicts

D. Life cycle conflicts.

Composable Semantic Models for Actors

Carolyn Talcott
Stanford University
clt@sail.stanford.edu

We are interested in developing a semantic foundation that supports specifying, composing, and reasoning about components of open distributed systems. The actor model provides the basic elements for open distributed computation: encapsulation of state; independent concurrent units of computation; interaction; and dynamic creation and interconnection. In order to provide for composability, and for reasoning about properties at many different levels of abstraction we introduce the notions of actor component, actor component algebra, and actor component algebra morphism. Morphisms from syntactic to semantic algbras give composable semantics. These concepts are illustrated through a series of simple examples illustrating features of the actor model and a variety of component algebras and morphisms.

Modules for Large-scale Programs from Semantic Point of View

C.S. Tang
Chinese Academy of Sciences, Beijing
cst@ox1.ios.ac.cn

There are structural differences between modules for programming in the small and programming in the large. The representatives of the former are procedures and processes of flow chart structure while that of the latter is the modules or its composable structure is based on data structures.

This paper proposes two points:

1. for the modules of large-scale programs, oo modules alone is not sufficient, because it is a kind of module oriented toward the domain (declaration of variables and operations), it only concerns with the static semantics. Message communication in this kind of modules only plays a supplementing role to implement the operation calls in concurrent situation. Beside oo modules, parallel statements ought to be also considered as a kind of modules. It's structure is also characterized with composability from its subparallel statements based on communication, e.g., $\| \; [\| \; [P_1; P_2]; P_3; \| \; [P_4; P_5; P_6]]$. This is a kind of module oriented toward the process of communication and dynamic semantics.

2. for parallel statement, the composability shown by its structure is only syntatical, its validity must presuppose the condition of semantic composability. This condition can be represented as follows:
   Let pre and post condition of the parallel statement

   $$\| \; [P_1; ...; P_k]$$

   be PRE and $POST$ respectively and those of its items $P_i, i = 1, ..., k$ are PRE_i and $POST_i$, the semantic composability condition is:

   $$PRE \equiv PRE_1 \wedge \ldots \wedge PRE_k$$

   $$POST \equiv POST_1 \wedge \ldots \wedge POST_k$$

   Because the dynamic message communication can destroy the post condition of a process, it is a difficult problem to garantee the validity of this composability condition. Some language (e.g., Unity) adopts the approach to put the restrictions on the language in order to make this condition hold, but it results in making the language too restrictive to be acceptedd as a system programming language.

In this paper another approach is suggested. We treat each input predicate, say $chan?x$, as an entry of the process and the input variable, say x, as a special kind of input parameter. As pre condition of the process is, in fact, a kind of constraint to the input parameters in order to garantee the validity of the post condition, there must be some restrictions to be found for the input variable x of the predicate $chan?x$.

We have given answers to following questions concerning this problem:

(a) How to find the restriction to each input variable of the input predicate? By means of Hoare logic, a verifier of Hoare can automatically generate this restriction at its proper place.

(b) How to arrange this restriction in the program? After handshaking of the input predicate, to arrange a command to check the message received by the input variable, whether the restriction holds. If it does, then go on, otherwise to throw it off and return to the original position to wait for a new message. It means that only non destructive messages are really accepted.

(c) How to arrange this restriction in the specification? It is difficult because in essence, this restriction is different from precondition because it is a dynamic property, it could contain variables of dynamic values. However, these variables can be replaced in some way.

A Formal Approach to Object-Oriented Software Engineering

Martin Wirsing
Ludwig-Maximilians-Universität München
wirsing@informatik.uni-muenchen.de

The goal of this talk is to show how formal specifications can be integrated into pragmatic object-oriented software development methods. We use Jacobson's approach OOSE ("Object-Oriented Software-Engineering") and combine it with object-oriented algebraic specifications by extending object and interaction diagrams with formal annotations. The specifications are based on Meseguer's Rewriting Logic and are written in an extension of the language Maude by process expressions. As a result any such diagram can be semi-automatically associated with a formal specification, proof obligations ensuring invariant properties can be automatically generated, and the refinement relations between documents on different abstraction levels can be formally stated and proved.