Hans-Dieter Ehrich, Yulin Feng,
David Kung (organizers)
Grit Denker (editor):

Object-Oriented Software Development

Dagstuhl-Seminar-Report
7.-11.4.97 (9715)

DAGSTUHL SEMINAR

ON

Object-Oriented Software Development

Organized by:

Hans-Dieter Ehrich (Technische Universität Braunschweig, Germany)
Yulin Feng (Chinese Academy of Sciences, Beijing, China)
David Kung (The University of Texas at Arlington, USA)

Schloß Dagstuhl, April 7 - 11, 1997

Contents

1 Preface

The purpose of the seminar was to bring together scientists working in the field of object-oriented software development. This includes object-oriented requirements analysis, specification and design, programming, testing and maintenance. The topics ranged from theoretical foundations to practical development projects.

In the paradigm of object-orientation, software systems are considered to be dynamic collections of autonomous objects that interact with each other. Autonomy means that each object encapsulates all features needed to act as an independent computing agent: individual attributes (data), methods (operations), behavior (process), and communication facilities. And each object has a unique identity that is immutable throughout its lifetime. Coincidentally, object-orientation comes with an elaborate system of types and classes, facilitating structuring and reuse of software.

The object paradigm is widely discussed in software technology, there are object-oriented programming languages, database systems, and software development methods. The basic idea is not new, essential concepts were already present in the programming language Simula, in the end sixties. Wider acceptance, however, only came with Smalltalk in the beginning eighties. But still, in many application areas, object-oriented methods and systems are not state of the art yet.

While the object paradigm is fairly successful in practice, it finds more scepticism than enthusiasm among theoreticians. But there is growing interest in clean concepts and reliable foundations. In fact, object-orientation badly needs theoretical underpinning, for improving practice and facilitating teaching.

The seminar organizers came from three continents: Europe, Asia and America. This indicates another purpose of the seminar, namely to bring together scientists who do not meet easily because they work in distant parts of the world.

27 participants accepted the invitation and came to the seminar, among them scientists from China and Japan, the US, and several European countries.

The seminar program covered four days, from Monday to Thursday. Each morning had two talk sessions. Each afternoon except Wednesday had another talk session followed by a working group session. On Wednesday afternoon, there was the obligatory excursion.

Three working groups were set up, one on requirements engineering, one on concurrency, and one on object-oriented models, formalisms and methods for component software design. The first two working groups were organized and chaired by Roel Wieringa and Barbara Paech, respectively. The third working group was coorganized and cochaired by Yulin Feng and Zhenyu Qian. Participation was well balanced among the working groups, and discussion was lively. The findings of the working groups were presented and discussed in a plenary session. the chairpersons prepared summaries that are included in this report.

The talks showed a broad spectrum of topics around object-orientation and software, from requirements engineering via modelling, specification and design towards programming and testing. Interest is shifting towards distributed systems, bringing

concurrency and communication problems into focus. Evidently, object-oriented concepts and methods are fruitfully applied to a variety of areas. Although there is still a way to go, confusion about fundamental concepts and notions is diminishing and mutual understanding is growing, due after all to a growing body of object theory.

There was beautiful weather during the seminar week, quite unusual for that time of the year. The workshop atmosphere was favorably influenced by much sun and warmth, and the excursion on Wednesday afternoon was a highly welcomed opportunity to enjoy a foretaste of summer.

The Organizers

Hans-Dieter Ehrich *Yulin Feng* *David Kung*

2 Final Seminar Program

Monday, April 7, 1997

9:00 Opening Remarks
Hans-Dieter Ehrich, Germany

Session 1, 9:45 - 12:30

Chair: Hans-Dieter Ehrich

9:45 *Object State Testing and Fault Analysis for Reliable Software Systems*
David Kung, USA

10:30 BREAK

Chair: Alfs Berztiss

11:00 *From Structured Analysis to Object-Oriented Design: Choosing the Best of Both Worlds*
Roel Wieringa, The Netherlands

11:45 *Modelling Business Rules with Situation/Activation Diagrams*
Micheal Schrefl, Austria

Session 2, 14:00 - 18:00

Chair: Gerhard Goos

14:00 *Component Software Engineering in Object Orientation*
Yulin Feng, China

14:45 *A Combined Reference Model- and View-Based Approach to System Specification*
Gregor Engels, The Netherlands

15:30 BREAK

16:00 Parallel Working Groups:

WG 1: Requirements Engineering
Chair: Roel Wieringa, The Netherlands

WG 2: Concurrency
Chair: Babara Paech, Germany

WG 3: Object-oriented Models, Formalisms and Methods for Component Software Design
Chair: Yulin Feng, China and Zhenyu Qian, Germany

Tuesday, April 8, 1997

Session 3, 9:00 - 12:30

Chair: Reind Van de Riet

9:00 *Problems Related to Combination of Dynamic Semantics with Static Semantics in Programming*
Chi-Song Tang, China

9:45 *Interactive Foundations of Computing*
Peter Wegner, USA

10:30 BREAK

Chair: Gilles Bernot

11:00 *Tools and Object-Oriented Methods - from the viewpoint of software development process -*
Masao Ito, Japan

11:45 *Analysing Object Specialization Hierarchies for Database Integration*
Gunter Saake, Germany

Session 4, 14:00 - 18:00

Chair: Gunter Saake

14:00 *Simple Functional Programming and Parameterized Classes for Java*
Zhenyu Qian, Germany

14:45 *Functional Object-Oriented Programming: A Gentle Introduction to Object-Gofer*
Wolfram Schulte, Germany

15:30 BREAK

16:00 Parallel Working Groups:

WG 1: Requirements Engineering
Chair: Roel Wieringa, The Netherlands

WG 2: Concurrency
Chair: Babara Paech, Germany

WG 3: Object-oriented Models, Formalisms and Methods for Component Software Design
Chair: Yulin Feng, China and Zhenyu Qian, Germany

Wednesday, April 9, 1997

Session 5, 9:00 - 12:30

Chair: Peter Wegner

9:00 *Object orientation in domain analysis for reuse*
Alfs Berztiss, USA

9:45 *A Seamless Transition from Processes to Object Models*
Barbara Paech, Germany

10:30 BREAK

Chair: Grit Denker

11:00 *An Object Classifier Based on Galois Approach*
Hele-Mai Haav, Estonia

11:45 *Web-Based Object Animation*
Martin Gogolla, Germany

∗ ∗ ∗ AFTERNOON EXCURSION ∗ ∗ ∗

Thursday, April 10, 1997

Session 6, 9:00 - 12:30

Chair: Yulin Feng

9:00 *Specification of Objects in Cyberspace using Linguistically Oriented Tools*
Reind Van de Riet, The Netherlands

9:45 *A Way of Specifying with Transactions Using Linear Temporal Logic*
Grit Denker, Germany

10:30 BREAK

Chair: Hele-Mai Haav

11:00 *ETOILE-Specifications and Real-Time-Issues*
Gilles Bernot, France

11:45 *Formal Semantics of Basic Message Sequence Charts: an Algebraic Approach*
Piotr Kosiuczenko, Germany

Session 7, 14:00 - 18:00

Chair: Arne Sølvberg

14:00 *Multi-Aspect System Descriptions for Object-Oriented Programming*
Keijiro Araki, Japan

14:30 *ALBERT at the age of five: a progress report*
Pierre-Yves Schobbens, Belgium

15:00 *Temporal Object Specification*
Hans-Dieter Ehrich, Germany

15:30 BREAK

16:00 Working Groups: Plenary Session

18:00 Closing Remarks
Hans-Dieter Ehrich, Germany

Friday, April 11, 1997

no program

3 Abstracts of Presentations

The following abstracts appear in alphabetical order of the speakers.

Multi-Aspect System Descriptions for Object-Oriented Programming

Keijiro Araki[1]
Kyushu University
Fukuoka, JAPAN

We report our case study of a system development with a variety of descriptions and analyses using Z, ML and Smalltalk. We get a set of system descriptions from various viewpoints including abstract functional aspects, system structural aspects, implementation feasibility aspects, and so on. We need not necessarily start from an abstract formal specification and refine it to a final concrete program, but we may start wherever easy to start. We discuss the roles of such descriptions and their interrelationships. Especially, we use ML as an executable specification language. By describing a system in ML, we would get insigts for abstract formal specifications as well as for system architectures and design issues. We intend to accumulate much experience in system development with a set of various descriptions and build up a road map for system development based on formal methods.

ETOILE-Specifications and Real-Time issues

Gilles Bernot[2]
Université d'Evry
Evry France

ETOILE is an object based formal specification theory. After a short introduction about the way ETOILE-specifications are structured, we describe the syntax of the specification of object types and we show how they can be combined to specify object systems.

An object type specification can be described as a sort of "star", the center of which is the type of interest, the branches being types of objects that can be used by an object of the center. ("etoile" is the French translation of "star"). A system of objects is then obtained by putting together several such stars. We follow the principle to match each branch of a star with the center of another one.

Then we show that adding a new object to an already existing system can entirely modify the system properties. Thus, if we want to establish formally the properties of

[1]This work has been done with Han-Myung Chang and Toshiyuki Tanaka.
[2]This is joint work with Marc Aiguier and Stefan Beroff.

an object system, we cannot proceed by establishing some lemmas on a small system, and complete the lemmas incrementally after adding objects one by one to the system, until we reach the full system under interest. Consequently, an incremental proving method cannot be obtained this way.

To allow to establish properties on an incremental way, we propose instead a method based on "object refinements". Within our ETOILE theory, we have defined a theory of refinement which has the interesting property that: if a system SYS1 correctly implements an object type O, and if SYS2 is a system that contains O and satisfies a property phi, then the system SYS3 obtained by replacing O by SYS1 in SYS2 still satisfies phi. This allow to start from a small, very abstract system with a small number of very abstract object types; and to make the system incrementally bigger and more precise, by successive refinements of its object types.

Real-time aspects are also currently an important topic for ETOILE. ETOILE specifications are used since 5 years to specify hardware/software systems for the co-design of some telecommunication systems and we often need to introduce statements about some delays in actual nanoseconds.

We have only defined such real time formulas for systems made of a unique object (i.e. systems with a unique global state). The idea is to add a special data type which represents time durations, with some built-in operations and predicates. This allows to specify methods that modify dynamically their behaviours according to their own execution time for example. An example is fully described to illustrate the approach.

Object orientation in domain analysis for reuse

Alfs Berztiss
University of Pittsburgh
Pittsburgh, USA and
University of Stockholm
Stockholm, Sweden

We consider domain analysis in the context of reuse-based development of software systems. Domain models are expressed in terms of processes, and a process is defined as an ordered set of tasks. A generic task is first formulated as a cliche in natural language, and reuse is achieved by adapting it for several specific applications. We follow an object-oriented approach to the definition of processes, with the understanding that object orientation has two aspects. One relates to data, but there is also a process aspect, and this latter aspect is our primary concern here. We consider process models and object orientation in reuse-driven development of information systems and control systems. Our method of defining application domains in terms of generic processes, where a process is regarded as a collection of tasks, is used to define the domain of repairs. This domain includes repair of machinery, road repirs, surgical procedures, and software debugging. We specialize the generic process for the last application. We also consider situations, by which we mean tasks that are so general that they arise in several domains.

A Way of Specifying With Transactions Using Linear Temporal Logic

Grit Denker[3]
Technische Universität Braunschweig
Braunschweig, Germany

Our concern is the high level specification of reactive software systems such as information systems. We adopt an object oriented, temporal logic based approach to specification. The notion of transaction incorporates various application domains, for instance transactions as abstractions from processes as known from refinement theory, transactions as abstractions from business processes as known in business process modelling or database transactions. We investigate object specifications with transactions. We illustrate the use of transactions by examples given in an object oriented specification style. A linear temporal logic with transactions (TOSL) serves as denotional model for object specifications. We explain how TOSL is semantically defined in terms of life cycles. Semantics is given to object specifications via translation to TOSL. We illustrate the translation by means of an example.

We use temporal logics as a semantical basis for object oriented specifications. Temporal logic is appropriate for specifying system dynamics, integrity constraints of the system and its components, relations between objects of the system, etc. But there exists the following problem: Temporal operators such as next, previous, etc. assume a specific time scale, i.e., a specific granularity of actions they refer to. But the granularity of an action may change, e.g., through application of refinement. Thus, a specification should be formulated in terms of transactions. But since the length of a transaction is not known a priori because its instantiation may be state-dependent, we formulate specification requirements relative to the begin or end, respectively, of the transaction. TOSL is a logic that provides concepts for specifying in terms of transactions. This way, we can specify system requirements as sets of TOSL formulae which are independent from the level of granularity. Thus, we may specify requirements of a system which will hold true on different levels of abstraction.

Temporal Object Specification

Hans-Dieter Ehrich
Technische Universität Braunschweig
Braunschweig, Germany

The TROLL and OMTROLL languages and the method supported by them aim at specifying distributed information systems on a high level of abstraction. The approach

[3]This is joint work with Jaime Ramos, Carlos Caleiro, and Amílcar Sernadas, Technical University Lisbon, Portugal.

integrates ideas from object orientation, abstract data types, conceptual modeling, behaviour modeling, specification of reactive systems, and concurrency theory. The talk gives an overview of the state of development of the TROLL language and method and its underlying semantic foundations. One of the highlights of the most recent version, TROLL 3, is to incorporate true concurrency and interaction issues. Specifications are based on distributed temporal logics, using sequential linear temporal logic for specifying local object behaviour, and adding communication expressions for specifying interaction. Different styles for the interaction parts of the logics and their interrelationships are being investigated.

A Combined Reference Model- and View-Based Approach to System Specification

Gregor Engels[4]
Leiden University
Leiden, The Netherlands

The idea of a combined reference model- and view-based specification approach has been proposed recently in the software engineering community. We present a specification technique based on graph transformations which supports such a development approach. The use of graphs and graph transformations allows to satisfy the general requirements of an intuitive understanding and the integration of static and dynamic aspects on a well-defined and sound semantical base. On this background, formal notions of view and view relation are developed and the behaviour of views is described by a loose semantics. View relations are shown to preserve the behaviour of views. Moreover, we define a construction for the automatic integration of views which assumes that the dependencies between different views are described by a reference model. The views and the reference model are kept consistent manually, which is the task of a model manager. In case of more than two views more general scenarios are developed and discussed. We are able to show that the automatic view integration is compatible with the loose semantics, i.e., the behaviour of the system model is exactly the integration of the behaviours of the views. All concepts and results are illustrated at the well-known example of a banking system.

[4]This is joint work with Hartmut Ehrig, Reiko Heckel, Gabi Taentzer, Technical University of Berlin, Germany.

Component Software Engineering in Object Orientation

Yulin Feng
Chinese Academy of Sciences
Beijing, China

The development of object oriented technology causes a great changes to traditional methods of application software design and such changes make it possible that application systems can be constructed through some software components. A component framework is not simply a set of prepackaged solutions from which a customer may pick and choose, it involves an actual custom tailoring and reengineering during the later 1990's mass customization. The tolk is prepared to a brief description of our research on component software engineering in object orientation, including object models, specification languages, design methodology and OO trends and perspectives etc.

Web-Based Object Animation

Martin Gogolla[5]
University of Bremen
Bremen, Germany

The current activities of our group comprises concrete case studies arising from practical projects: (1) Reengineering of a tram simulation system and (2) development of a material information system. Both projects use object-oriented description techniques on a semi-formal (OMT-like) and formal (TROLL-like) level. Semi-formal techniques are also the topic of a student project (18 students, 2 years). The aim of this student project is the development of an OMT design system.

The formal basis for the above activities is the object description language TROLL light. Our group already has experience with implementations of this language on different platforms. The current implementation uses persistent Java developed recently in a project at Glasgow University.

The talk explains the user-interface of the current animation system which is based on HTML documents. Thus any Web browser can be employed for the validation of specifications. For demonstration purposes we use a small description of a car rentals case system.

[5]This is joint work with Mark Richter.

An Object Classifier Based on Galois Approach

Hele-Mai Haav
Institute of Cybernetics
Tallinn, Estonia

Conventional object-oriented database systems (OODBS) provide good facilities for creating objects according to the static class lattice, but lack classification mechanism for existing objects. We propose the application of Galois approach to dynamic classification of objects in OODB. First, we define an object model as the Galois lattice of binary relationship between objects and classes they belong to. Second, we show that an object classifier can be built on the basis of construction of such a model. We argue that the object classifier can be used for recovering class lattice, controlling object migration, maintaining OO views as well as for retrieval of meta-information concerning context of object base.

Tools and Object-Oriented Methods - from the viewpoint of software development process -

Masao Ito
Nil Software Corporation
Tokyo, Japan

There is no single software development method (SDM) applicable to every problem domain. If a problem domain would change, we have to vary our development method and, at the same time, our software development environment (SDE) that is supporting it. Especially changing the SDE with commercial off-the-shelf (COTS) tools is not easy work. This is because of the clear differences in viewpoint between the developers who create the tools and the environment builders who use these tools. Developers usually try to make their tools as general-purpose as possible. On the other hand, environment builders and users want specific tools which will fit into their specialized environments.

In my presentation, I proposed a mechanism which is able to fill up this gap by using the concept of "prime modeling component(PMC)". I mean the word "prime" that the prime modeling component is the minimum but essential tool that cannot be divided. And the user can combine them in the process-centered way in order to get their own SDE, that is, they only arrange those PMCs according to the process that a SDM needs.

I use some object-oriented methods to explain my idea. These are the "Object Modeling Technique (OMT)", and "Shlaer and Mellor Method (SMM)", "Fusion method (Fusion)".

Formal Semantics of Basic Message Sequence Charts: an Algebraic Approach

Piotr Kosiuczenko
LMU München
München, Germany

The aim of my talk was to present a new and complete semantics for basic Message Sequence Charts (MSC). MSC have been introduced to provide a trace language for description and specification of communication behaviour of system components and their environment by means of message interchange. They have been recommended as a standard by International Telecommunication Union [ITU 93]. MSC present communication behaviour of a distributed system in a very intuitive and transparent manner. They support many different methodologies for specification, design, testing, simulation, and documentation of systems. MSC may be independently used for requirement specification, interface specification, validation and simulation, test case specification, and documentation of real-time systems [ITU 96]. There are two MSC standards: MSC-92 [ITU 93] defining basic MSC and its extended version MSC-96 [ITU 96] defining also higher level MSC (see also [RGG 96]). In this paper we consider only basic MSC, but this semantics can be extended to MSC 96. This will be a subject of further research. There are some formal semantics for basic MSC but they are unsufficient, and neither MSC-92 nor MSC-96 possess a satisfactory semantics. In general, the existing semantics do not formalize notions like decomposition, environment, local actions (c. f. [LaLe 95a]). It is worth of noticing, that the high level MSC, and in general most of the new features of MSC-96 still do not have any satisfactory semantics either. In this talk we proposed a new semantics for MSC-92 based on multiset algebras and asynchronous transition relations. Our model simplifies Maude language (c. f. [Mes 92], [MeWi 92]) and can be understood as a special case of a Simple Maude specification. Maude is an algebraic specification language based on term rewriting [Mes 92]. Like in Maude we describe a configuration of a distributed system as a multiset of objects and messages. A configuration can be understood as a snapshot of the system. Each object has a unique name and certain attributes. Each message has a unique address, and signature. The model makes computational progress by executing actions like reading or sending a messages, creating or deleting an object. Actions of a specified system are modeled by transition relations. The model provides an asynchronous composition operator allowing to describe a system composed of independent, asynchronously communicating agents. We introduceed a natural notion of conditions which are assumed to define sets of configurations. We propose here two kinds of composition operators. The first one is a vertical composition allowing to compose different system components to work in parallel. The second one is a horizontal composition corresponding to week sequencing. We discussed properties of these composition operators and their relation to conditions.

References:

[**ITU 93**] ITU-TS Recommendation Z.120. Message Sequence Charts (MSC). ITU-TS, Geneva, 1993.

[**ITU 96**] ITU-TS Recommendation Z.120. ITU-TS Recommendation Z.120. Message Sequence Charts (MSC). ITU-TS, Geneva, 1996.

[**LaLe 95a**] P. Ladkin, S. Leue. Comments on a Proposed Semantics for Basic Message Sequence Charts. The Computer Journal, 37(9), January 1995.

[**Mes 92**] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. Theoretical Computer Science 96, 1992, 158-200.

[**MeW 92**] J. Meseguer, T. Winkler. Parallel Programming in Maude. In J. Banatre and D. le Metayer (eds.): Research Directions in High-Level Parallel Programming Languages, Springer LNCS 574, 1992, 253-293.

[**RGG 96**] E. Rudolph, J. Grabowski, P. Graubmann. Tutorial on Message Sequence Charts. Computer Networks and ISDN Systems. Vol. 22(12), Elsevier, 1996.

Object State Testing and Fault Analysis for Reliable Software Systems

David Kung[6]
The Univ. of Texas at Arlington
Arlington, USA

Object state behavior implies that the effect of an operation on an object may depend on the states of the object and other objects. It may cause state changes to more than one object. Thus, the combined or composite effects of the object operations must be analyzed and tested. We show that certain object state behavior errors cannot be detected readily by conventional testing methods. We describe an object state test method consisting of an object state model, a reverse engineering tool, and a composite object state testing tool. The object state test model is an aggregation of hierarchical, concurrent, communicating state machines envisioned mainly for object state testing. The reverse engineering tool produces an object state model from any C++ program. The composite object state testing tool analyzes the object state behaviors and generates test cases for testing object state interactions. We show the detection of several composite object state behavior errors that exist in a well-known thermostat example.

[6]This is joint work with Yao Lu, Neena Venugopalan, Pei Hsia.

A seamless Transition from Processes to Object Models

Barbara Paech
TU München
München, Germany

In this talk we propose a description technique which serves as an intermediate between process descriptions und object models. Process descriptions are used during requirements engineering to capture the information of how the system is going to be used (e.g. workflow descriptions) and object models describe the (conceptual) design of a system. In the transition from processes to object models a lot of decisions have to be taken. Processes describe exemplary, isolated behaviour and the behaviour is given in term of data dependencies between activities. Object models describe the behaviour in terms of services and the definition of the services has to take into account the integration of different processes on the data of the object.

As an intermediate we propose to use roles, which also structure the behaviour in services, but only describe the behaviour relevant to a given context. The services are described by input/output - state transition diagrams and the integrated behaviour of the services (related to the context) is derived by interleaving of the state transition diagrams.

Our approach relates to Jacobsons' OOSE, but we use process diagrams instead of text for the use cases and, because of the concurrency of services, we can capture the control part of the use cases in the design within services and do not need to create control objects.

Simple Functional Programming and Parameterized Classes for Java

Zhenyu Qian
Universität Bremen
Bremen, Germany

There exist several suggestions for additional features for Java. However, all the suggestions we know are either inconsistent with some existing Java features or require nontrivial extensions for Java Bytecode. We suggest a new approach, which extends Java with parametrized classes, algebraic types, higher-order functions, binary operations, ML-polymorphism and self types. The main features of the approach are that it is consistent with Java arrays and the type theory is much simpler than the existing approaches. In addition, a translation of our extension to Java has been provided, so that no extensions for Java Bytecode are necessary.

Analysing Object Specialization Hierarchies for Database Integration

Gunter Saake[7]
Otto-von-Guericke-Universität Magdeburg
Magdeburg, Germany

Analysing and merging of object specialization hierarchies is an important step in the process of building a database federation. Federated database systems enable a homogeneous access to distributed, heterogeneous databases still allowing local applications. For building an integrated view on the stored data, the local schemata have to be analysed and integrated into a global database schema. The local schemata often contain specialization hierarchies. Such hierarchies are often encompassed by local schemata even implicitly by a relational schema via foreign keys. Therefore, different specialization hierarchies including extensional and intension aspects have to be analysed and integrated into one minimal but complete specialization hierarchy.

We present an integration approach using the database integration model GIM. Based on an extensional and intensional analysis the local classes are partitioned into disjoint extensions (potential class populations) and presented in a two-dimensional GIM matrix showing the intensional and extensional properties of database classes. Using this representation, we discuss algorithms based on formal concept analysis which allow to automatically generate an integrated specialization hierarchy. Our approach generalize and formalize other approaches (e.g. by S. Spaccapietra and by S. Navathe). The integrated hierarchy can produced automatically by an efficient algorithm.

ALBERT at the age of five: a progress report

Pierre-Yves Schobbens
Université Notre-Dame de la Paix
Namur, Belgium

The Albert language for expressing requirements is based on a structuration in 'agents' and on a real-time temporal logic.

We summarize here recent developments of this language and its environment.

1. Semantics

   In response to industrial case studies, we have slightly changed the formal semantics of Albert. The new semantics ensures: - time continuity: while in previous versions of Albert the state of the system was only known at "snapshots", now we ensure that it is known at any time. - invariance under stuttering: the introduction of a different division of time is now guaranteed to be irrelevant.

[7]This is joint work with Ingo Schmitt.

2. Tools

Several tools are currently developed: 1- a graphical and textual syntax editor; 2- a type checker; 3- a proof assistant based on PVS; 4- (in project) tools for abstracting specifications, and for real-time model checking.

Modelling Business Rules with Situation/Activation Diagrams

Michael Schrefl[8]
Universität Linz
Linz, Austria

Business rules are statements about business policies and can be formulated according to the event-condition-action structure of rules in active database systems. However, modeling business rules at the conceptual level from an external user's perspective requires a different set of concepts than currently provided by active database systems. This talk identifies requirements on the event language and on the semantics of rule execution for modeling business rules and presents a graphical object-oriented language, called Situation/Activation diagrams, meeting these requirements.

A paper on the topic of this talk appears in the Proceedings of the IEEE International Conference on Data Engineering 1997.

Functional Object-Oriented Programming: A Gentle Introduction to Object-Gofer

Wolfram Schulte
Universität Ulm
Ulm, Germany

Object-Gofer is a strict superset of the functional programming language Gofer which incorporates the following ideas from the object-oriented community: objects and classes, subtype and implementation inheritance, method redefinition, late binding and self-type specialization. Since Object-Gofer is a strict superset of Gofer, the benefits of functional programming are still available, that is, the language has type inference, algebraic datatypes, higher-order functions, lazy evaluation and most notably ease of reasoning. The semantics of Object-Gofer is defined by the translation into Gofer. Although this restricts the design space, it turns out that using a suitable framework of monads, higher-order polymorphism, and overloading, objects smooth well with functions.

[8]This is joint work with P. Lang and W. Obermair.

Problems Related to Combination of Dynamic Semantics with Static Semantics in Programming

Chi-Song Tang
Chinese Academy of Sciences
Beijing, China

XYZ/E is a TLL (Temporal Logic Language) to combine the dynamic semantics with the static semantics in an uniform TL framework. This paper is used to explain the meaning and impact of this approach historically and technically. In 2−4, some meaningful problems are discussed in more detail: (1) A method is introduced to evaluate a specification which is a mixture of the commands of state transition with the commands of pre-post assertion. This method can be considered as a new approach for rapid-prototyping for our TLL System. (2) A new concept "agent" is defined in XYZ/E for OOP, in which the static semantics of repository data module can be combined coherently with the dynamic semantics of message-passing. This concept can be used to solve the difficult problem of autonomous object concept in OOP. (3) In contrast to the linear decomposable modularity of OOP oriented toward the domain of a program, the linear decomposability of the semantics of parallel statement is considered to be the basis of a kind of modularity oriented toward the process to solve the problem. The author of this article believes that a more desirable approach of the modularity for programming in large is to combine these two kind of modularity coherently.

Specification of Objects in Cyberspace using Liguistically Oriented Tools

Reind Van de Riet
Vrije Universiteit
Amsterdam, The Netherlands

Objects in Cyberspace come in two (different) forms:

- objects as passive things, which can be inspected and retrieved by:
- subjects as dynamic things, representing human beings.

The latter can be simulated/implemented/realized by active objects (in the technical sense of the word) and can be seen as a combination of e-mail and Social Security Number; they are called: Alter-egos. To model or specify the static and dynamic properties of alter-egos a tool is being used, called COLOR-X. With this tool it is possible to specify the behaviour of alter-egos in a way close to a specification in Natural Language. A Lexicon is being used for this: WordNet, which gives the meaning of concepts (or words) such as :"to borrow". To specify more precisely the behaviour and static aspects of the alter-ego a language CPL is used which has been derived from another linguistic tool: Functional Grammar, in use by linguists to define meaning of words and of sentences. Using these semantically rich tools it is possible to automatically derive:

- Verbalizations of the model in NL sentences;
- State Transition Diagrams for all the objects involved, using again the lexicon to exploit the fact that "to borrow" is the antonym of "to lend";
- Mokum programs so that simulation of the processe is possible (Mokum is an object-oriented active database system, in use and developed in our group).

In ongoing work Work Flow Diagrams, similar to COLOR-X models are being used to derive Security and Privacy rules for and about the alter-egos.

Interactive Foundations of Computing

Peter Wegner
Brown University
Providence, USA

Interactive systems are shown to have richer behavior than algorithms. The proof that Turing machines cannot model interaction machines is surprisingly simple: interaction is not expressible by a finite input string. Interaction machines extend the Chomsky hierarchy, are modeled by interaction grammars, and precisely capture fuzzy concepts like open systems and empirical computer science.

We examine extensions to interactive models for algorithms, machines, grammars, and semantics, and consider the expressiveness of different forms of interaction. Interactive identity machines are already more powerful than Turing machines, while noninteractive parallelism and distribution are algorithmic. The extension of Turing to interaction machines parallels that of the lambda to the pi calculus, but the ability to model shared state allows interaction machines to express more powerful behavior than calculi. Asynchronous and nonserializable interaction are shown to be more expressive than sequential interaction (multiple streams are more expressive than a single stream).

It is shown that interaction machines cannot be described by sound and complete first-order logics (a form of Godel incompleteness), and that incompleteness is inherently necessary to realize greater expressiveness. In the final section the robustness of interactive models in expressing open systems, programming in the large, graphical user interfaces, and agent-oriented artificial intelligence is compared to the robustness of Turing machines. Applications of interactive models to coordination, objects and components, patterns and frameworks, software engineering, and AI are examined elsewhere in [We5, We6]. The main results of this work are embodied in propositions P1 to P36 below:
P1 (interaction machines): Interaction machines cannot be modeled by Turing machines.
P2 (nonenumerability): The interaction histories of an interaction machine are nonenumerable.

P3 (diagonalization): Diagonalization proofs express nonenumerability by interactive processes.
P4 (on-line algorithms): On-line processes with the closed-system property are on-line algorithms.
P5 (dynamic interaction): Dynamic interaction is more expressive than on-line algorithms.
P6 (complexity): Some problems with algorithmic complexity NP have interactive complexity P.
P7 (constraints): Constraints can specify nonalgorithmic noncompositional emergent behavior.
P8 (grammars): Interactive listening machines can express richer behavior than generative grammars.
P9 (inclusion): Dynamic inclusion refines set inclusion as a measure of expressive power.
P10 (expressiveness): Bisimulation, dynamic inclusion, and game semantics are equally expressive.
P11 (irreducibility): Extensional behavior cannot express intensional behavior and vice versa.
P12 (concurrency): Interleaving models, enhanced operational semantic models, and true concurrency have progressively greater expressive power.
P13 (nonmonotonicity): Openness and interactiveness are nonmonotonic system properties.
P14 (duality): Observation/control duality in control theory mirrors algorithm/interaction duality.
P15 (noncompositionality): Process and object behavior is not compositional.
P16 (frameworks): Frameworks can be specified by constraints on constituent components.
P17 (identity): Interactive identity machines can express richer behavior than Turing machines.
P18 (agents): Agents interacting with nonalgorithmic systems have nonalgorithmic behavior.
P19 (management): Interactive management is more expressive than rule-based management.
P20 (orthogonality): Interaction, parallelism, and distribution are orthogonal forms of behavior.
P21 (transactions): Interactive correctness conditions are more natural than serializability.
P22 (unicasting): Unicasting models computational, chemical, biological, and sexual interaction.
P23 (processes): Process calculi have algorithmic reduction rules and interactive control rules.
P24 (pi calculus): The pi calculus has the expressive power of serializable interaction machines.
P25 (asynchrony): Asynchronous is more expressive than synchronous interacton.

P26 (nonserializability): Nonserializable is more expressive than serializable interaction.
P27 (physics): Distinctions between Newtonian, relativistic, chaos, and quantum-theory models of physics can be characterized by styles of interactive computing.
P28 (openness): Openness in mathematics, physics, and computing has a common foundation.
P29 (logic): Logic is too weak to model interactive computation.
P30 (models): Sound and complete models have an enumerable number of true statements.
P31 (incompleteness): Interaction machines have no sound and complete first-order logic.
P32 (errors): Systems for finding errors in programs are neither sound nor complete.
P33 (robustness): Interaction has many alternative models with the same expressive power.
P34 (systems): Software engineering systems have interactive, nonalgorithmic models.
P35 (empiricism): The intuition that empiricism extends rationalism can be proved for computing.
P36 (empirical CS): Interaction machines precisely characterize empirical computer science.

From Structured Analysis to Object-Oriented Design: Choosing the Best of Both Worlds

Roel Wieringa
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Software system specification can be partitioned into the specification of external interactions and the specification of an internal decomposition. By and large, structured and object-oriented analysis methods offer comparable techniques and heuristics for the specification of external interactions but differ in the techniques and heuristics used for the specification of an internal decomposition. This talk explains this claim and makes it as precise as possible (but not more precise, because that would be pedantic as well as useless - assuming it would be possible).

We assume a simple framework for specification methods borrowed from systems engineering and from other work in software engineering methodology. The framework distinguishes external interactions from internal decomposition, and distinguishes for kinds of properties of external interactions that we can specify: functions, behavior, communication and all other properties. We can describe these properties of external system interactions, but also of each system component.

In structured analysis, the system components are themselves functions, or more precisely, data processes, control processes and data stores. In object-oriented analysis,

the components are objects. In the talk we discuss the techniques used by Yourdon structured analysis and by the UML for the specification of external system interactions, of a system decomposition, and of the interactions of components.

The comparison leads us to conclude that there is no fundamental incompatibility between techniques for specifying interactions - be they interactions of the system or of its components. Examples of these techniques are the statement of purpose, a function refinement tree, event-response pairs, a context diagram, use case diagrams, message sequence charts and state transition diagrams. This does not imply that all of these should be used in one specification, but it does imply that the specifyer has the freedom to choose from these techniques as he or she sees fit.

The specification techniques for the internal decomposition are incompatible in that structured techniques separate the representation of memory (data stores), data processes (functions) and control processes (state machines), whereas object-oriented techniques encapsulate all of these into objects. Other than that, we can find some unexpected compatibilities: the decomposition criteria of functional decomposition, event-partitioning, device partitioning and domain partitoning, can all be used in structured models or in object-oriented models. There is no fundamental incompatibility here and the choice is up to the designer.

The talk ends with a non sequitur, viz. a plea for formal foundations of semiformal (diagram-based) techniques.

4 Summary of Working Groups

Working Group 1: Requirements Engineering

Chair: Roel Wieringa, The Netherlands

The working group on requirements engineering started with defining two of the key terms, "requirements engineering" and "object-orientation". Two views on requirements engineering came forward:

- Requirements are desired properties of systems. These may be properties of external interactions or of internal decomposition, so this turns out to be a white box view of requirements: Any desired property of an external interaction or of a component is a requirement.

- The second view that came forward was that a requirement is a contract between customer and software engineer in which they agree that the software will be delivered according to the requirements. This view is more of the black box kind, because the customer will not generally state required properties of the internal decomposition.

Despite their apparent difference, these two views were compatible enough to proceed as if there were agreement.

Discussion on the second key term, "object-orientation", was considerable shorter. We agreed to follow Peter Wegner's characterization and took the encapsulation of actions with state as the minimal feature of object-oriented software. This differs from the encapsulation of operations with data, for that would merely be a definition of an abstract data type. It is essential that objects have dynamics, hence they have local actions and a local state. This also implies that they have an identity which persists through change.

After settling on our terminology, we discussed three questions.

1. What is the contribution of object-orientation to requirements engineering?

2. Is an object-oriented software model a requirements specification or a design specification?

3. Can we move beyond object-orientation in requirements engineering?

In answer to the first question it was remarked that the use of scenarios and message sequence charts is definitely not a contribution of object-orientation. These techniques have been in use for a long time in the telecommunications area and are not invented by object-oriented methodologists. Object-orientation did make two closely related contributions.

- Requirements are structured into classes. This allows the collection of data about the implementation of classes, and hence cost and schedule predictions on the basis of the characteristics of class specifications.

- Requirements are structured into classes on the basis of the heuristic of domain modeling. The objects found in the application domain of the software are used to model software objects, on top of which software functions are defined. It was observed that this idea is more than 30 years old, since it arose in Simula67, and is also closely related to JSD, which is 15 years old. Nevertheless, the idea reached widespread acceptance through the object-oriented methods.

The second question, whether an object-oriented software model is a requirements specification or a design specification, provoked a renewed discussion about what requirements are. If we restrict ourselves strictly to external interactions of a system, then requirements do not refer to the system state. It is as if we view the entire system as an object and only refer to its external interactions. In this restricted view of requirements, object models have no place because they structure the state of the software into objects. After some discussion we reached some kind of an agreement that object models are part of requirements in the wider sense and can be called conceptual design, which is distinct from physical design. They structure the software, but do so in terms of the application domain, which should be meaningful to the user.

The third question, whether we can move beyond object-orientation, quickly led to the observation that in order to advance the state of the art, we need domain-specific techniques. Once we deal with, say, traffic regulation systems, we can build domain models and reuse them in the specification and design of traffic regulation systems. This observation ties in well with the observation that the contribution of object-orientation to requirements engineering is domain modeling.

In a second discussion round, we considered agent-orientation as a way of advancing the state of the art. Realizing that the concept of agent is vague and that there is no single widely accepted definition, we settled on the provisional definition that an agent is an object with intentions, by which it tries to reach goals, and beliefs, by which it maintains information about its external world. Requirements engineerin could incorporate this by viewing the system and its environment as agents. The implication is that requirements engineers should consider the goals of the environment and as well as of the system, including priorities of these goals.

The discussion then took an unexpected turn because the question was raised why we should use formal logic for the specification of agents. Why not use what we know from psychology and sociology about intentions, goals and beliefs? This quickly led to the question whether everything can be formalized, or whether there are certain phenomena, such as intention and belief, that resist formalization. We retreated from this question to another one, namely whether it is useful to formalize the formalizable. We reached agreement about the facts that customer wishes may be vague and inconsistent, but that product requirements must be made precise. There are however two different interpretations of the word "precise":

- Testable
- Formal

These interpretations are not the same. It seems reasonable enough to demand that requirements can be tested so that the customer can know why he or she should accept the delivered system. One argument for formalization upon which we agreed is that software tool support requires formalization. An example of this is the execution semantics of statecharts, which is formal. Having strayed this far from our original working group goal, and given the fact the the time was up, we decided that it would be nice to report our results back to the plenary session.

Working Group 2: Concurrency

Chair: Barbara Paech, Germany

The aim of this working group was to identify and discuss topics related to concurrency and relevant to the work of all the participants. A first overview on that work showed the following structure:

Most of the participants were concerned with _inter-object_ concurrency (message passing), while _intra-object_ concurrency (shared state) was only a minor issue. The concurrency between the objects was due to the _problem domain_ (e.g. workflow), but also to the _solution domain_ (e.g. hardware). All the participants were using specification languages for concurrency, but no concurrent object-oriented programming languages. Therefore issues like synchronization constraints were not discussed in details.

Following this overview, the discussion concentrated on the following two issues:

- composition of specifiations and
- specification of communication patterns.

Composition of Specification Influenced by Peter Wegners talk, the group concentrated on the specification of _open_ systems whose execution environment can not be determined at specification time. To deal with this problem, according to the approach of Abadi/Lamport assumptions about the environment can be incorporated into the specification. When composing such assumption/commitment specifications, these assumptions have to be verified. If no assumptions are made about the environment, the behaviour _emerging_ through composition has to be constrained in order to allow for the composition of the specifications. An example of such constraints is given in Gilles Bernods talk on ETOILE.

Specification of Communication Pattern The experience of the participants showed that in the specification of distributed object systems the main effort has to be invested into the specification of the communication behaviour, while the functional behaviour of the objects can be specified straightforwardly. This problem can be alleviated through so called _communication structures_ which capture typical communication behaviour in the same sense that data and control structure capture typical functional behaviour. Connectors like Pipes and Filters which are identified in the area of software architectures are an example of such communication structures. While the question for good communication structure still remains (and has to be solved in the context of specific application domains), the group discussed the question of specification languages for such communication structures. There is an overwhelming variety of specification languages for distributed systems. The group looked at the distinction between specification languages which describe communication from a _global_ viewpoint in contrast to a _local viewpoint_. The latter being exemplified by TROLLs distributed temporal logic. The first one allows to introduce special language constructs for communication structures, while in the latter such a structure would be captured in some kind of controller object enforcing the communication structure between the other objects. In both cases, it would be desirable to have a collection of structures (similar to design patterns, but concentrating on communication) allowing for reuse. However, the issue of communication seems to be so fundamental that a further structuring of such a collection into basic primitives and results on substitutability between communication structures is important.

Working Group 3: Object-oriented Models, Formalisms and Methods for Component Software Design

Chair: Yulin Feng, China, and Zhenyu Qian, Germany

Component Software Design supports the independent production of software components for a component market and/or later composition. The components should be allowed to interoperate. The user of the components cannot change the components. Open research questions in this area include formal and informal methods for

1. Interface specifications: How to specify a component? What informtion should be provided and in which style? How can this information be used to compose the design?

2. Implementations: How to implement a component based on a specification? How to test, verify, etc.

3. Architecture: What is a proper architecture for Component Software Design.
4. Designing process: How to combine composition actitivies with usual OO design methods?
5. Supporting tools: visualization, code generation, etc.
6. Standards: COM/OLE 2, Corba/SOM/OpenDoc, Java/JavaBeans.
7. Consistency problems between components.
8. Protocols

The results

The discussion shows that the problem with Component Software Design is very complex. The main reason is that the complete usability of a software component does not seem to be expressible by the currently known formal approaches. The participants come to the following conclusions.

The behavior of a software component may be specified by an interface, which consists of a type specification and a protocol. The type specification specifies the static properties and the protocol the dynamic ones. Specification of the dynamic behavior is still a research topic. The protocols can be specified either formally or informally. A protocol is a set of operation sequences that the user of the component can execute.

It seems that the existing mathematical approaches could not describe the complete dynamic behavior of a software component independently of the environment, where the component is intended to be used. Practice is leading theory.

The participants discussed about the *Interaction Machine* (IM) proposed by P. Wegner. IM's are more powerful than Turing Machines and useful for specifying today's software components. But a formal definition of IM is still to be developed.

Sometimes implementions lead to unavoidable errors w.r.t. specifications. Verification of an implementation against a specification is not always possible, although it is very important in safety-critical applications. The reasons are that most practical specifications are not formal, and that verification may sometimes be very difficult. Testing and validation is more important than verification. It seems that we could not achieve safety, efficiency and usability at the same time. Tradeoffs should be made for concrete applications.

COM, CORBA and JavaBeans are useful technology for the time being. However, they are still too restricted in one or the other way for the real interoperability. The participants are expecting big changes.

Dagstuhl-Seminar 9715 List of Participants

Keijiro **Araki**
Kyushu University
Dept. of Computer Science and
Communication Engineering
6-1 Kasuga-Koen
Kasuga City - Fukuoka 816
Japan
araki@csce.kyushu-u.ac.jp
tel: +81-92-583-7624
fax: +81-92-583-1338

Gilles **Bernot**
Université d'Evry
La.M.I.
Cours Monseigneur Romero
91025 Evry Cedex
France
bernot@lami.univ-evry.fr
tel: +33-1 69 47 74 66
fax: +33-1 69 47 74 72

Alfs **Berztiss**
University of Pittsburgh
Dept. of Computer Science
Pittsburgh PA 15260
USA
alpha@cs.pitt.edu
tel: +1-412-421-6960 / 412-624-8401
fax: +1-412-624-8854

Stefan **Conrad**
Otto-von-Guericke-Universität
Institut für Techn. Informationssysteme
Postfach 4120
D-39016 Magdeburg
Germany
conrad@iti.cs.uni-magdeburg.de
tel: +49-391-67-1 80 66
fax: +49-391-67-1 20 20

Grit **Denker**
Technische Universität Braunschweig
Informatik, Abt. Datenbanken
Postfach 3329
38023 Braunschweig
Germany
G.Denker@tu-bs.de
tel: +49-531-391-3103
fax: +49-531-391-3298

Hans-Dieter **Ehrich**
Technische Universität Braunschweig
Abteilung Datenbanken
Postfach 3329
38023 Braunschweig
Germany
HD.Ehrich@tu-bs.de
tel: +49 531 3271
fax: +49 531 3298

Gregor **Engels**
University of Leiden
Department of Computer Science
Niels Bohrweg 1
NL-2300 RA Leiden
The Netherlands
engels@wi.leidenuniv.nl
tel: +31-71-527-7063
fax: +31-71-527-6985

Yulin **Feng**
Chinese Academy of Sciences
Institute of Software
P.O.Box 8718
Beijing 100080
P. R. China
feng@ios.ac.cn
tel: +86-10-6257-23 28
fax: +86-10-6256-25 33

Martin **Gogolla**
Universität Bremen
Fachbereich 3
AG Datenbanksysteme
Postfach 33 04 40
D-28359 Bremen
Germany
gogolla@informatik.uni-bremen.de
tel: +49-421-218-74 69
fax: +49-421-218-30 54

Gerhard **Goos**
Universität Karlsruhe
Fakultät für Informatik
Postfach 6980
D-76128 Karlsruhe
Germany
ggoos@ipd.info.uni-karlsruhe.de
tel: +49-721-608-47 60
fax: +49-721-69 14 62

Hele-Mai **Haav**
Institute of Cybernetics
Department of Software
Akadeemia tee 21
EE0026 Tallinn
Estonia
helemai@cs.ioc.ee
tel: +372-2-527-314
fax: +372-6-397-042

Masao **Ito**
Nil Software Corporation
E-Space 302
Kinuta
Setagaya-ku
Tokyo 157
Japan
nil@nil.co.jp
tel: +81-3-3749-8601
fax: +81-3-3749-8602

Piotr **Kosiuczenko**
LMU München
Institut für Informatik
Oettingenstr. 67
D-80538 München
Germany
kosiucze@informatik.uni-muenchen.de
tel: +49-89-2178-21 33
fax: +49-89 2178-21 75

David **Kung**
University of Texas at Arlington
Department of CS & Engineering
Box 19015
Arlington TX 76019
USA
kung@cse.uta.edu
tel: +1 817 272-3627
fax: +1 817 272-3784

Welf **Löwe**
Universität Karlsruhe
GMD-Forschungsstelle
Vincenz-Prießnitz-Str. 3
D-76128 Karlsruhe
Germany
loewe@ipd.info.uni-karlsruhe.de
fax: +49-721-69 14 62

Barbara **Paech**
TU München
Institut für Informatik
Arcisstr. 21
D-80290 München
Germany
paech@informatik.tu-muenchen.de
tel: +49-89-289-2-81 86]
fax: +49-89-289-2-81 83

Zhenyu **Qian**
Universität Bremen
Fachbereich Mathematik/Informatik
Postfach 33 04 40
D-28334 Bremen
Germany
qian@informatik.uni-bremen.de
tel: +49-421-218-22 39
fax: +49-421-218-30 54

Isidro Ramos **Salavert**
Universidad Politecnica de Valencia
Dept. de Sistemas Informaticos y Computacion
Apartado 22012
Cami de Vera s/n
E-46071 Valencia
Spain
iramos@dsic.upv.es
tel: +34-6-387 7350
fax: +34-6-387 7359

Gunter **Saake**
Otto-von-Guericke Universität Magdeburg
Fakultät für Informatik
Inst. für Technische Informationssysteme
Postfach 4120
D-39016 Magdeburg
Germany
saake@iti.cs.tu-magdeburg.de
tel: +49-391-67-18800
fax: +49-391-67-12020

Pierre-Yves **Schobbens**
Université Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage 21
B-5000 Namur
Belgium
pys@info.fundp.ac.be
tel: +32-8172-4990

Michael **Schrefl**
Universität Linz
Institut für Wirtschaftsinformatik
Abt. für Data and Knowledge Engineering
Altenbergerstr. 69
A-4040 Linz
Austria
schrefl@dke.uni-linz.ac.at
tel: +43-732-2468-9480
fax: +43-732-2468-9471

Arne **Sølvberg**
The Norwegian Institute of Technology
Dept. of Electrical Engineering and Computer Science
N-7034 Trondheim
Norway
asolvber@idt.unit.no
tel: +47-7-593438
fax: +47-7-594466

Chi-Song **Tang**
Chinese Academy of Sciences
P.O. Box 8718
Beijing 100080
P.R. China
cst@ox1.ios.ac.cn
tel: +86-10-255-6910
fax: +86-10-6256-2533

Reind **Van de Riet**
Vrije Universiteit
Dept. of Mathematics and Computer Science
De Boelelaan 1081 a
NL-1081 HV Amsterdam
The Netherlands
vdriet@cs.vu.nl
tel: +31-20-4447757

Peter **Wegner**
Brown University
Department of Computer Science
P.O. Box 1910
Providence RI 02912
USA
pw@cs.brown.edu
tel: +1-401-863-7632
fax: +1-401-863-7657

Roel **Wieringa**
Vrije Universiteit Amsterdam
Faculteit Wiskunde en Informatica
De Boelelaan 1081a
NL- 1081 HV Amsterdam
The Netherlands
roelw@cs.vu.nl
tel: +31-20-444-7771
fax: +31-20-444-7653