Hans-Dieter Ehrich, Gregor Engels,
Jan Paredaens, Peter Wegner (editors):

Fundamentals of Object-Oriented Languages, Systems, and Methods

Dagstuhl-Seminar-Report 95
22.-26.8.94 (9434)

DAGSTUHL SEMINAR

ON

Fundamentals of Object-Oriented Languages, Systems, and Methods

Organized by:

Hans-Dieter Ehrich (Technische Universität Braunschweig, Germany)
Gregor Engels (Leiden University, The Netherlands)
Jan Paredaens (University of Antwerp, Belgium)
Peter Wegner (Brown University, Providence, USA)

Schloß Dagstuhl, August 22 - 26, 1994

Contents

1 Preface

During the last decade, there has been progress within the object-oriented community in agreeing on what object-orientedness means. The role of object identity, specialization, inheritance, and dynamic binding have been examined. However, a careful look at articles written by people with a theory, programming language, database, and software engineering background shows that the semantical understanding of the same features still differs considerably. A well-known example is the feature of inheritance, which has quite a different meaning for persons from the database community with a semantic data model background and persons from the programming language community with a compiling technique background. This observation also applies to the literature on object-oriented software development methods. Here, object-oriented analysis and design methods are often heavily influenced by semantic data modelling concepts. An implementation of an object-oriented design by an object-oriented programming language usually causes more difficulties than expected although the "same" object-oriented paradigm is used.

Even when there is agreement on terminology, there are legitimate differences in perspective among different object-oriented research communities. It was the intention of this seminar to clarify differences in terminology as well as to clarify legitimate differences in perspective. The seminar brought together an approximately equal number of researchers from each of these four areas and was being jointly organized by four organizers representing each of the four areas:

- theoretical foundations of object-orientedness (H.-D. Ehrich)
- object-oriented programming languages (P. Wegner)
- object-oriented database systems (J. Paredaens)
- object-oriented software development methods (G. Engels)

This gathering of experts, working in the "same" field within computer science, offered an excellent opportunity to discuss and understand the differences, similarities, and commonalities of basic notions within the object-oriented world. In addition, the specific atmosphere of Schloß Dagstuhl gave new impulses to already existing cooperations and stimulated new "interdisciplinary" research in the whole range of object-orientedness.

This booklet summarizes the presentations and discussions during the seminar. It comprises the final seminar program, abstracts of all (morning) talks, summaries of all (afternoon) workshops, as well as a list of the participants.

Hans-Dieter Ehrich Gregor Engels Jan Paredaens Peter Wegner

2 Final Seminar Program

Monday, August 22, 1994

9:00 Opening Remarks
Gregor Engels, Germany

Session 1, 9:30 - 12:30
Chair: Gregor Engels

9:30 *What comes after Object-Oriented Programming*
Peter Wegner, USA
10:00 *Fundamentals of Adaptive Object-Oriented Software*
Karl Lieberherr, USA
10:30 *Pitfalls and Possible Solutions of Object-Oriented Development*
Gerti Kappel, Austria

11:00 BREAK

11:30 *Distribution and Concurrency without Tears*
Hans-Dieter Ehrich, Germany
12:00 *A Method for Specifying Conceptual Models of Data Processing Systems*
Roel Wieringa, The Netherlands

Working Groups, 14:00 - 18:00

14:00 Discussion about Topics
Chair: Peter Wegner

15:00 BREAK

15:30 Parallel Working Groups:

WG 1: Inheritance: Covariance vs. Contravariance
Chair: Roel Wieringa, The Netherlands

WG 2: Observing Behavior of Objects
Chair: Gunter Saake, Germany

Tuesday, August 23, 1994

Session 2, 9:30 - 12:30

Chair: Hans-Dieter Ehrich

9:00	*Coordination Constraints* Gregor Engels, The Netherlands
9:40	*ODMG-93 Standard vs. Object-Oriented Databases Theory* Andreas Heuer, Germany
10:20	*Fundamentals of Object-Oriented Database Models* Bernhard Thalheim, Germany
11:00	BREAK
11:20	*Incremental Specifications for Abstract Object-Oriented Modeling* Gianna Reggio, Italy
12:00	*Testing Equivalence and Containment of Database Queries Involving Object Identifier Creation* Jan van den Bussche, Belgium

Working Groups, 14:00 - 18:00

14:00	Plenum

Chair: Peter Wegner

15:00	BREAK
15:30	Parallel Working Groups:
	WG 3: Object Evolution Chair: Roel Wieringa, The Netherlands
	WG 4: Class Organization Chair: Karl Lieberherr, USA

Wednesday, August 24, 1994

Session 3, 9:30 - 12:30

Chair: Jan Paredaens

9:00	*The Chimera Object-Oriented Data Model* Giovanna Guerrini, Italy
9:40	*Typing Issues Related to Object-Oriented Queries* Herman Balsters, The Netherlands
10:20	*Structural and Behavioral Views on OMT-Classes* Jürgen Ebert, Germany
11:00	BREAK
11:20	*Inheritance Conditions for Object Life Cycle Diagrams* Gunter Saake, Germany
12:00	*Object Creation in Databases* Marc Gyssens, Belgium

∗∗∗ AFTERNOON EXCURSION ∗∗∗

Thursday, August 25, 1994

Session 4, 9:30 - 12:30

Chair: Gerti Kappel

9:00	*Method Engineering of Object-Oriented Analysis and Design Methods: Concept Analysis and Method Assembly* Sjaak Brinkkemper, The Netherlands
9:40	*On Reification in Object-Oriented Specification* Grit Denker, Germany
10:20	*A Meta-Language for Representation of Evaluation Invariants in OODB*
10:20	Hele-Mai Haav, Estonia
11:00	BREAK
11:20	*Inheritance of Object Behavior: Extension and Refinement of Object Life Cycles* Michael Schrefl, Germany
12:00	*Database Applications of Reflective Programming* Gottfried Vossen, Germany

Working Groups, 14:00 - 18:00

14:00 Plenum
Chair: Peter Wegner

15:00 BREAK

15:30 Parallel Working Groups:

WG 5: Database Models
Chair: Gottfried Vossen, Germany

WG 6: Software Engineering Models
Chair: Sjaak Brinkkemper, The Netherlands

Friday, August 26, 1994

Session 5, 9:30 - 12:00
Chair: Peter Wegner

9:00 *A New Type-theoretic Approach to Objects*
Radu Grosu, Germany
9:30 *Group-Oriented Architecture Development*
Wilhelm Schaefer, Germany
10:00 *Generic Update Operations for OODB*
Christian Laasch, Germany

10:30 BREAK

Working Groups, 11:00 - 12:00

14:00 Plenum

12:00 Closing Remarks
Gregor Engels

3 Abstracts of Presentations

The following abstracts appear in alphabetical order of the speakers.

Typing issues related to object-oriented queries

Herman Balsters
University of Twente
Enschede, The Netherlands

In the various proposals for O-O data models, there also occur those employing set-constructs, especially in the case that query results are of importance. In these data models there is no consensus –as yet– on how to type sets; i.e. each data model has its own way to type check correctness of set-membership. In some cases it is allowed to collect expressions that have different minimal types into a set. Such sets are called heterogeneous, as opposed to homogeneous sets where it it is allowed only to collect expressions of exactly the same minimal type. In the paper "A semantics of O-O sets" by Balsters & de Vreeze (cf. [DBPL'91], Nafplion, Greece) it is argued that the correct way to handle O-O sets is to only allow for homogeneous sets. In that paper we worked with greatest-lower-bound (GLB) and least-upper-bound (LUB) constructs, as given in the basic type system of Cardelli. In order to give further evidence that the homogeneous approach is indeed the correct way of dealing with O-O sets, we have taken up the issue again by looking into alternative type systems. In order to investigate whether it is in some way possible to define a type system allowing for heterogeneous sets, we employ the notion of disjunct sums (A + B, where A,B are types). The idea is that the A + B -construct could replace the LUB(A,B)-construct as a candidate for typing heterogeneous sets. It turns out that the more subtle construct A + B (w.r.t. the much coarser construct LUB(A,B)) cannot be included in the type system without also including certain additional axioms. These additional axioms, concerning the + -construct, seem to be rather ad-hoc and somewhat unnatural (both in form and content), thus making such a type system unappealing to use. There also remains a formal argument in favor of homogeneous sets (cf. [DBPL'91]) that also makes a type system employing + -constructs and sets hard to maintain.

Equivalences among conjunctive queries involving object creation

Jan Van Den Bussche[1]
University of Antwerp (UIA)
Antwerp, Belgium

Classical conjunctive queries on relational databases have been well-studied. The properties of equivalence and subsumption between conjunctive queries are decidable by testing for

[1]This is joint work with Riccardo Torlone, IASI-CNR, Rome, Italy

the existence of a homomorphism. These decisions have NP complexity. In a 1991 PODS paper, Hull and Yoshikawa considered the problems of testing equivalence of conjunctive queries, or more generally unions of conjunctive queries, extended with the possibility of creating new objects. This problem is considerably more intricate than the classical problem. In particular, the existence of a classical homomorphism is no longer necessary but only sufficient for subsumption between queries. Hull and Yoshikawa were able to establish decidability of equivalence in the special case of "isolated object creation". Their approach, however, is not based on the idea of homomorphisms and does not yield tests with NP complexity. In this work, we try to generalize the notion of homomorphism to take into account the intricacies involved in the creation of new objects in terms of existing objects. Our goal is to thus obtain NP tests for containment, and also equivalence, between conjunctive queries with object creation.

Method Engineering of Object-oriented Analysis and Design Methods: Concept Analysis and Method Assembly

Sjaak Brinkkemper[2]
Center of Telematics and Information Technology
University of Twente
Enschede, The Netherlands

Method Engineering is the engineering discipline of methods, techniques and tools for systems development. We report about an in-depth analysis of ten generally accepted O-O methods, available in textbooks. The comparison is performed by method modelling, resulting into detailed information on the concepts of the methods in Concept Structure Diagrams, and on the steps of the procedure of the methods in Task Diagrams. Extensive comparison tables of steps, concepts, techniques are presented. From the comparison it can be concluded that the contemporary methods, although diverse, possess a common kernel of basic object oriented concepts. The process models of the methods turn out to be very rudimentary exposing the limited experience with the methods. The complexity of the various graphical specification formalisms of the methods indicate the absolute necessity of advanced tool support when the methods are to be applied in practice.

Method fragments, being coherent parts of methods, can be stored in a so-called method base for method engineering purposes. We discuss the selection and assembly of method fragments into a situation specific method. The architecture of the Computer Aided Method Engineering tool (CAME tool) is presented.

Literature:

S. Hong, G. van den Goor, S. Brinkkemper, A Formal Approach to the Comparison of Object- Oriented Analysis and Design Methodologies, Hawaii International Conference on System Sciences (HICSS) (IEEE Computer Society Press, Hawaii) 1993, Vol. IV, pp. 689-698.

[2]This work is jointly performed with Frank Harmsen, Han Oei, Arjan Bulthuis (all Univ. Twente, NL), Shuguang Hong (Georgia State Univ, USA) and Geert van den Goor (Andersen Consulting, NL).

F. Harmsen, S. Brinkkemper and H. Oei, Situational Method Engineering for Information System Project Approaches, To appear in: T.W. Olle et.al. (Eds.), Proceedings of CRIS'94, North-Holland, September 1994.

F. Harmsen, S. Brinkkemper and H. Oei, A Language and Tool for Situational Method Engineering for Information System Project Approaches. To appear in the Proceedings of ISD'94 Conference, Slovenia, September 1994.

Transactions in Object-Oriented Specifications

Grit Denker[3]
Technische Universität Braunschweig
Braunschweig, Germany

The formal step by step development of implementations from specifications is necessary to allow the incremental description of large software systems and hence split the software development process in tractable portions. Due to the complex notion of objects as units of structure and behavior the refinement process has to be reconsidered in the object-oriented framework. Apart from refining structure the behavioral part of objects give rise to refine actions by transactions. As such the refinement step possibly comprises a change of granularity with respect to the design process. We will refer to this as (action) reification. Referring to information systems as application domain synchronization problems arise when different transactions access shared data. Because of this reification of actions by transactions forces to take care of concurrency control issues. We aim at introducing transactions in object-oriented specifications for supporting action reification and outline a semantics for it.

In particular, we have the most liberal composition of transactions resulting from a reification process in mind. More precisely, imagine two abstract actions each reified to a sequence of concrete actions, i.e, transactions. It is too strict to force the sequential execution of these abstract actions to be reified only through the sequential composition of the corresponding transactions. There may exist concrete actions in the transactions which do not access the same sources, and, therefore, they are independent and may be executed in arbitrary order without changing the overall effect. Our aim is to provide a semantic framework in which reification of actions is treated as liberal as possible with respect to serial composition of transactions. We propose a semantics based on event structures which appropriately explain the semantics of abstract objects as well as the one of reified objects, and, moreover, liberalizes the sequential composition of transactions.

[3]This is joint work with Hans-Dieter Ehrich, Technical University Braunschweig.

Structural and Behavioral Views of OMT-Classes

Jürgen Ebert[4]
Universität Koblenz
Koblenz, Germany

The dynamic behavior of objects of a given class may be described by a state diagram (STD), which specifies the allowed sequences of method calls. Such an STD, sometimes called life cycle diagram, has to be compatible with the STDs of its super- and subclasses.

It is shown that the existence of an homomorphism from the subclass STD to the superclass STD is a sufficient criterion for the compatibility of behavior descriptions with respect to inheritance.

Some examples are given including the case of restriction views to classes, which turn out to be (virtual) superclasses where the homomorphisms are a projections.

Distribution and Concurrency without Tears

Hans-Dieter Ehrich[5]
Technische Universität Braunschweig
Braunschweig, Germany

Fully concurrent models of distributed object systems are specified using linear temporal logic that does not per se cope with concurrency. This is achieved by employing the principle of local sequentiality: we specify from local viewpoints assuming that there is no intra-object concurrency but full inter-object concurrency. Local formulae are labelled by identity terms. For interaction, objects may refer to actions of other objects, e.g., calling them to happen synchronously. A locality predicate allows for making local statements about other objects. The interpretation structures are global webs of local life cycles, glued together at shared communication events. These interpretation structures are embedded in an interpretation frame that is a labelled locally sequential event structure. Two initiality results are presented: the category of labelled locally sequential event structures has initial elements, and so has the full subcategory of those satisfying given temporal axioms. As in abstract data type theory, these initial elements are obvious candidates for assigning standard semantics to signatures and specifications.

[4]This is joint work with Gregor Engels, Leiden University, The Netherlands
[5]This is joint work with Amílcar Sernadas, Cristina Sernadas, Gunter Saake and Grit Denker

Coordination Constraints

Gregor Engels[6]
Leiden University
Leiden, The Netherlands

Nowadays object-oriented analysis and design methods propose to split the specification of a system into several parts, where each part describes a certain perspective. We follow this line and present the object-oriented specification language SOCCA (Specification of Coordinated and Cooperative Activities). A SOCCA specification consists of

- a class diagram to describe the structural part of objects and their interrelations with other objects,
- state transition diagrams to describe the external behavior (”life cycle”) of objects,
- an interaction diagram to describe the ”uses” relationship between classes, and
- (once more) state transition diagrams to describe the internal behavior (”realization”) of operations.

In addition to most of the existing approaches, we also model the coordination of object behavior explicitly. This is done by splitting the internal behavior descriptions into subdiagrams and by controlling the transition from one subbehavior to another by coordination constraints added to the external behavior description of objects.

A New Type Theoretic Approach to Objects

Radu Grosu
Technische Universität München
München, Germany

In this talk we present a novel, implicitly typed λ–calculus for objects, by viewing these as extendible case–functions rather than as extendible records.

This novel view, allows to unify the concepts of function, object and process into one concept, that of a functional entity which is self contained and provided with a uniform communication protocol. We use this view to give a formal foundation for both sequential and concurrent object oriented languages. In the later case, we view objects as case–functions communicating asynchronously over unbounded channels.

Our calculus is a conservative extension of the polymorphic type system of Aiken and Wimmers, to include case–function extension and lazy data types. Its soundness is proven with respect to a semantical model based on ideals. Subtyping and case–function extension play a central role in our modeling of generalization/specialization and inheritance. To model self and self–class, our calculus includes recursive types. These are also necessary to model streams and provide the theoretical background for passing streams themselves as messages. We use higher order streams to express mobile systems.

[6]This is joint work with Luuk Groenewegen, Leiden University, The Netherlands

The implicitly typed λ–calculus is accompanied with a decidable type inference algorithm, which always delivers the least type of a term (if this exists). An implementation of this algorithm was written in Common Lisp.

The Chimera Object-Oriented Data Model

Giovanna Guerrini[7]
Università di Genova
Genova, Italy

Chimera is a novel database language that integrates object-oriented, deductive and active capabilities, developed as part of ESPRIT Project 6333 IDEA. Chimera supports all the concepts commonly ascribed to object-oriented data models, such as object identity, complex objects, user-defined operations, classes and inheritance. Moreover it provides capabilities for expressing deductive rules, that can be used to define view and integrity constraints, to express the implementation of derived features and to formulate queries. Finally, Chimera supports a powerful language for defining triggers.

In this talk, the Chimera language is presented and its innovative features are pointed out. A particular emphasis is given to the Chimera data model. The most relevant characteristics of this data model are (constrained) user-defined types, value classes, class features, implicitly populated classes, intensionally defined attributes and declarative method implementation. Then, the most relevant issues in the formalization of the Chimera data model are discussed. These issues are mainly related to heterogeneous structures, metaclasses, ISA hierarchies and typing. The main contributions of this formalization work are a clear definition of classes and schema, a precise characterization of type refinement and the definition of a set of typing rules for declarative expressions (terms, formulas and rules). Finally, our current directions in enriching the model are sketched, discussing features like objects belonging to several most specific classes, constraint and trigger redefinition and support for composite objects.

Object Creation in Databases

Marc Gyssens[8]
University of Limburg
Limburg, Belgium

Two properties that are often desired in object-oriented databases are the availability of a complete language to manipulate the data and the possibility to create new objects. Completeness issues have been investigated at length for data models in which all entries in the result of a query are already present in the input. In the presence of object creation, however, transformations become non-deterministic and require adapted completeness criteria. We discuss determinacy, constructiveness, semi-determinism, and swap-genericity, and point out

[7]This is joint work with Elisa Bertino, Università di Milano, and René Bal, University of Twente

[8]This is joint work with with Marc Andries, Leiden University; Jan Van den Bussche, University of Antwerp; and Dirk Van Gucht, Indiana University

the connections between these notions. We also discuss to which extent complete languages exist for these various notions.

A Meta-language for Representation of Evolution Invariants in OODB

Hele-Mai Haav[9]
Estonian Academy of Sciences
Talinn, Estonia

Class lattice design is the most important step of OO database design process. In order to keep the database schema correct and consistent during its evolution several constraints have to be checked dynamically.

We propose a logic based meta-language for representing constraints on class and object lattices and provide a set of corresponding inference rules. The meta-language [1] differs from other logic based languages for complex objects in its intention to be used on the top of the OO language NUT and its tight relationships with underlying OO specifications of objects and classes.

From the logical point of view the meta-language is a very simple language based on Horn clauses without function symbols in order to represent class and object lattices specified by the OO language NUT. Notions of class atoms and superclass atoms are introduced to the meta-language for mapping concepts from the OO language to the meta-language. These atoms are generated automatically during the compile time of class definitions using certain transformation rules defined in [2].

We provide a set of inference rules for class and object lattice constraints as well as for evolution invariants.

The meta-language will be prototypically incorporated to the OO language and environment NUT developed at the Institute of Cybernetics of the Estonian Academy of Sciences [3].

References:

1. Haav H-M , A Meta-language for Specification of Evolving Class and Object Lattices in OODB, In: Proceedings of the Second International East-West Database Workshop, Sept. 25-28, 1994, Klagenfurt, Austria (to appear)

2. Haav H-M. and Matskin M., Using Partial Deduction for automatic propagation of changes in OODB, In: Information Modelling and Knowledge Bases IV: Foundations , Theory and Applications, IOS Press, Amsterdam, 1993 pp 339-353

3. Tyugu E., Matskin M, Penjam J, Eomois P., NUT- An object-oriented language, Computer and AI 1986, 6:521-542

[9]This is joint work with M. Matskin, Tallinn

The ODMG-93 Standard vs. Object-Oriented Database Theory

Andreas Heuer
University of Rostock
Rostock, Germany

Since end of 1993, there is a standard for an object-oriented database model and associated object definition and query languages: the ODMG-93-Standard [3]. Since this standard was mainly initiated by vendors of existing OODBMS, the concepts of some of these existing systems (e.g. ObjectStore and O_2) are mixed up.

As one result of this process, the standard has become *inconsistent* (e.g. the definition of inheritance and keys), *too weak* (e.g. the query semantics and all dynamic aspects) and *not "safe"* (e.g. the usage of keys and a proposed extension for multiple type membership of objects may cause problems).

A formal OODB model (following the lines of Beeri [1] consists of classes and types containing objects and values which are organized in class and type hierarchies. While a class hierarchy is defined by subsetting of extents of classes, the type hierarchy is defined in the Cardelli-Wegner style [2]. A prerequisite for the definition of class hierarchy is the multiple membership of objects in class extents.

The ODMG-Standard mixes up the two definitions even though objects can reside only in one class up to now. This is one of the inconsistencies in the standard.

In a formal OODB model, a query language is closed and adequate [4], that is, given as an input extents of classes and types organized in hierarchies, the result of a query should also be extents of derived classes and types which are integrated into the existing hierarchy.

This is not possible with the OQL language of the standard which only allows to collect sets of objects of a given class or values of some type. Object creation and derivation of classes is not possible.

The ODMG-standard introduces keys as a means to identify objects not even by the (invisible) object identifier but also by values. Since keys can be defined via component classes, objects can be identified by values of component objects. There are situations where the identification of objects is not possible though there are keys defined for each class: this is caused by class-component class relationships with cycles or even more subtle structures. There are solutions for this problem in OODB theory, e.g. the usage of keys in the OSCAR OODBMS or the criteria introduced by Beeri and Thalheim (see this volume).

[1] C. Beeri. A formal approach to object-oriented databases. *Data and Knowledge Engineering*, 5(4):353–382, 1990.

[2] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.

[3] R.G.G. Cattell, editor. *The Object Database Standard: ODMG-93.* Morgan-Kaufmann, San Mateo, CA, 1994.

[4] A. Heuer and M.H. Scholl. Principles of object-oriented query languages. In *Proceedings GI-Fachtagung "Datenbanksysteme für Büro, Technik und Wissenschaft", Kaiserslautern*, pages 178–197. Springer, Informatik-Fachbericht 270, 1991.

The Advocatus Diaboli of Object-Oriented Development

Gerti Kappel
University of Linz
Linz, Austria

Object-oriented development has to overcome two currently existing obstacles, one technical by nature and the other organizational, to prove success in the large.

The *technical obstacle* has three causes. *Firstly*, some of the concepts of object-oriented analysis methods and object-oriented programming languages comprise subtle yet severe differences. A prominent example is the use of specialization inheritance during analysis and the use of specification inheritance or even code inheritance during implementation. *Secondly*, the almost exclusive concentration on the concept of an "object" forces that everything has to be expressed in terms of encapsulated objects. However, there are situations which demand further concepts, too. For example, complex operations involving several object classes should be modeled independently of a particular object class. Another example concerns the modeling of local object behavior versus the modeling of global system behavior. Whereas the former comprises the possible life cycles of objects, the latter should be stated in a more flexible manner, for example in terms of production rules. A third example concerns the information hiding aspect of objects. To be able to decrease the complexity of an object-oriented system it is necessary to resolve the coupling between objects beforehand. This could be done easily if the required interface, i.e., the operations used from other objects, would be part of the publicly visible interface of objects. *Lastly*, there is still a gap between the understanding of objects in programming languages and in database systems. For example, the notion of an active object as having an own thread of control in programming languages is different to the notion of an active object as reacting according to some predefined ECA rules in database systems.

The *organizational obstacle* is due to a lack of new project management principles. These have to take into account that object-oriented development is development by investment, that new job profiles are necessary for proliferation of a no-not-invented-here syndrome, and that we need new metrics like the lines-of-reused-code metric, to mention just a few.

Resolving these obstacles would make object-oriented development a real candidate for building quality systems.

Generic Update Operations for Objectbases

Christian Laasch[10]
University of Ulm
Ulm, Germany

In OODBMSs type-specific methods are used for manipulating objects, in order to maintain the consistency of the database. This is, however, of little help for the method implementor as far as the model-inherent constraints are concerned. We propose a set of *generic update operations*, including operations for object evolution that maintain integrity constraints that

[10]This is joint work with Marc H. Scholl, University of Ulm, Ulm, Germany

can be expressed in database schemas. On the one hand integrity constraints such as types, class memberships, subtype-, subclass-relationships, class predicates, and inverse functions are kept consistent after update operations, on the other hand the capabilities to express semantics in a schema are chosen such that such a set of update operations exists. The update operations can be used for implementing type-specific update methods or directly by applications. We present an approach to consistently define update semantics for an object model including classes, views, and variables that is based on necessary and sufficient predicates akin to defined concepts in KL-ONE style languages.

Foundations of Adaptive Software

Karl J. Lieberherr[11]
Northeastern University
Boston, USA

We present shortcomings of object-oriented programs and queries (redundancy and inflexibility) and we introduce adaptive programs to overcome the shortcomings. An adaptive program is a program which adjusts automatically to changing contexts. They may be: sequential behavior, structure, distribution, concurrency, etc.

Adaptiveness is achieved by splitting programs into loosely coupled, cooperating contexts. Changes to one context may then preserve, due to the loose coupling between contexts, the intent of the other contexts, leading to adaptiveness. One specific form of adaptiveness is achieved through succinct subgraph specifications and by using only two contexts: sequential behavior and structure. Adaptive programs of this form are expressed at a high level of abstraction, minimizing information about the implementation in the form of a specific class structure. The programs are called adaptive since they work for infinitely many class structures.

Adaptive programs apply Polya's inventor paradox to programming: Instead of developing a set of specific classes to solve a problem, we implicitly develop a family of sets of classes. A specific set of classes to solve the given problem is obtained through a customization process.

For further information, see Comm. ACM, May 94, page 94.

Incremental Specifications for Abstract Object-Oriented Modelling

Gianna Reggio[12]
Universita' di Genova
Genova, Italy

We present a novel approach showing that classical algebraic techniques can support the incremental specification of classes of objects.

The approach is in three parts:

[11]This is joint work with Cun Xiao, Jens Palsberg and other member of the Demeter Research Group.
[12]This is joint work with Egidio Astesiano, University of Genova, Italy

- object model
- basic class specifications
- incremental specifications.

Object model Commonly to many approaches (starting with CCS), an object is modelled as a labelled transition tree, whose labels correspond to method invocation and acceptance and whose nodes are the local state configurations. As it is well known, that kind of model easily supports concurrency and distribution. The encapsulation of the local state is provided by an appropriate observational semantics (bisimulation) forgetting the nodes.

Here the key new feature in our approach, supporting incremental specifications (by inheritance), is the abstract canonical form we give to the states, which in a sense is a very abstract generalization of the notion of record.

Basic class specifications A concrete class is specified as a many sorted algebra with sorts, whose elements are the states and the labels respectively, and a ternary predicate giving the labelled transitions. The specification takes the form of a conditional specification, whose axioms for the transitions follow a special pattern.

Incremental specifications An inheritance operation over specifications is defined with its semantic model. Then "incremental specifications" are defined inductively by inheritance on the basis of basic class specifications. Key property: the incremental specifications enjoy a property, persistence of the properties of the inherited symbols, which tries to capture what it is preserved through the inheritance operation.

Inheritance Conditions for Object Life Cycle Diagrams

Gunter Saake[13]
Otto-von-Guericke Universität Magdeburg
Magdeburg, Germany

Inheritance is the main principle in object-oriented design methods to support structuring and reuse of object behavior descriptions. Most proposals restrict the (formal) use of inheritance to method interfaces and method effect specifications. We propose to extend the inheritance relation to cover whole object life cycles, i.e., to long term object behavior. This inheritance relation corresponds to IS-A inheritance in object-oriented data models. An interesting question is to lift the inheritance relation from the level of object life cycles to the level of life cycle descriptions, for example graphical notations like state transition automata.

The object-oriented specification language TROLL supports behavior inheritance by including (temporal) logic behavior description of the superclass into the objects of the subclass. This corresponds to a subset relation on object traces, which in fact is a special case of an

[13]The current work on the TROLL language is joint work with T. Hartmann, P. Hartel and J. Kusch. The discussion on object life cycle diagram inheritance is result of a joint research effort with R. Feenstra, P. Hartel, R. Jungclaus and R. Wieringa.

embedding graph morphism. The talk discusses to lift the idea of having graph morphisms as inheritance condition up to the level of *behavior descriptions*. It is shown that this criterion gives an intuitively acceptable condition to check two given state transition automata whether one of them inherits the other description. After sketching the basic idea of inheriting object life cycles, we give inheritance conditions and inheritance-preserving construction operators for the graphical notation used in the OMT dynamic model for specifying life cycles.

This work was partially supported by CEC under ESPRIT WG 6071 IS-CORE II (Information Systems – COrrectness and REusability), and under ESPRIT WG 8319 ModelAge (A Common Formal Model of Cooperating Intelligent Agents).

Group-Oriented Architecture Development

Wilhelm Schäfer[14]
Universität Dortmund
Dortmund, Germany

The talk presents an architecture definition language (Module Interconnection Language). The main new feature of this language are particular language constructs which are exploited to support cooperative architecture development. A corresponding software process model based on those features is presented. The suitability of an object-oriented database system as the underlying implementation platform is discussed. Finally, the talk extends the idea of "product-based" process modeling. It sketches the solution of the Merlin project which is based on a dedicated, rule-based process model definition language and a set of predefined so-called cooperation patterns. Similar to transactions, those patterns have a well-defined synchronization policy which guarantees consistency of a software project but allow flexibility in terms of defining sharing policies of software documents in a (large) software project.

Inheritance of Object Behavior: Extension and Refinement of Object Life Cycles

Michael Schrefl[15]
Universität Linz
Linz, Austria

We discuss inheritance of object-life cycles in the realm of Behavior Diagrams[KS91], which are based on Petri Nets. A behavior diagram of an object type represents the possible life cycles of its instances by activities, states, and arcs corresponding to transitions, places, and arcs of Petri Nets. Opposed to Petri Nets, an activity must be explicitly invoked on a specific object resident in all pre-states of the activity; and, as activities are not instantaneous, an object resides in an activity state for every activity which is currently performed on the object.

[14]This is joint work with Wolfgang Emmerich and Stefan Wolf
[15]This is joint work with Gerti Kappel, University of Linz

Object life cycles can be specialized in two ways: (1) by extension, which means adding activities, states and arcs, and (2) by refinement, which means expanding activities and states in subdiagrams.

A life cycle occurrence (LFO) of an object is a sequence of life cycle states (LFS), each being a set of states and activities, such that the transition from one LFS to the next one reflects the start or completion of an activity.

The behavior diagram B' of a subtype O' of O consists at least of the activities, states and arcs of the behavior diagram B of O (inheritance axiom).

B' is a consistent extension of B, if every LCO of B' from which all activities and states added in B' have been deleted is also a LCO of B.

Sufficient and necessary conditions for checking whether B' is a consistent extension of B exist, if only activities alive in the initial LFS are considered and the inheritance axiom is obeyed: B' is a consistent extension of B, iff (1) no arc is added in B' between two elements which both belong to B and (2) an activity added in B' does not have a state belonging to B as pre- or post-state.

Behavior refinement is used in two ways: (1) for developing the behavior diagram of one object type by stepwise refinement [Sch90], and (2) for refining activities and states, not yet refined at a supertype, at a subtype.

Structured refinement primitives for states and activities together with sufficient rules for consistently embedding refinements are presented.

[Sch90] Schrefl, M.: Behavior Modeling by Stepwise Refining Behavior Diagrams. Int. Conf. on Entity-Relationship Approach, 1990.

[KS91] Kappel, G. and M. Schrefl: Object Behavior Diagrams. IEEE Int. Conf. on Data Engineering, 1991.

Fundamental Concepts of Object-oriented Databases

Bernhard Thalheim
Cottbus Technical University
Cottbus, Germany

It is often claimed that object oriented databases overcome many of the limitations of the relational model. However, the formal foundation of OODB concepts is still an open problem. Even worse, for relational databases a commonly accepted data model existed very early on whereas for OODBs the unification of concepts is still missing. During our talk we report our first investigations on a formally founded object oriented databases.

A clear distinction between objects and values turns out to be essential in the OODM. Types and classes are used to structure values and objects respectively. One important concept of object oriented databases is object identity. The immutable identity of an object can be encoded by the concept of abstract object identifiers. The identification problem can be solved for classes with extents that are completely representable by values (value-representable). Uniqueness constraints express equality on identifiers as a consequence of the equality of some values or references.

The success of the relational data model is due certainly to the existence of simple query and update languages. Preserving this goal in OODB is a serious goal. This property of existence of generic update operations can be carried over to object oriented data models if classes are value- representable.

One of the primary benefits that database systems offer is automatic enforcement of database integrity. One type of integrity is maintained through automatic concurrency control and recovery mechanisms; another one is the automatic enforcement of user-specified integrity constraints. The maintenance problem is the problem how to ensure that the database satisfies its constraints after certain actions. There are at present two approaches to this maintenance problem. The first one, more classical is the modification of methods in accordance to the specified constraints. The second approach uses generation mechanisms for specified events. Upon occurrence of certain database events like update operations the management component is activated for integrity maintenance. The first direction did not succeed because of limitations within the approach. The second one is at present one of the most active database research areas.

Especially for active database systems rule triggering systems are used for integrity maintenance. However, this approach cannot succeed even for very simple schemata. During the talk we demonstrate that the first approach can be extended to object oriented databases using stronger mathematical fundamentals. We outline an operational approach based on the computation of greatest consistent specializations.

This research shows that identification is one of the main properties for object oriented databases. For the user the abstract identifier of an object has no meaning. Therefore, a different access to the identification problem is required. The unique identification of an object in a class leads to the notion of weak value-identifiability. It turns out that this notion is equivalent to the notion of identifiability. A database instance is identifiable if the orbit of each object in the database is trivial. This notion is weaker than the notion of value-representability which is required for the unique definition of generic update operations.

References:
C. Beeri, B. Thalheim, Can I See Your Identification Please. Manuscript, Jerusalem-Cottbus, 1994.
K.-D. Schewe, B. Thalheim, Fundamentals Concepts of Object-oriented Databases. Acta Cybernetica, 11, 1-2, 1993, 49 - 83.
K.-D. Schewe, B. Thalheim, J.W. Schmidt, I. Wetzel, Integrity Enforcement in Object-oriented Databases. In 'Foundations of Models and Languages for Data and Objects' Modelling Database Dynamics, Springer, 1993
K.-D. Schewe, J.W. Schmidt, D. Stemple, B. Thalheim, I. Wetzel, A Reflective Approach to Method Generation in Object-oriented Databases. Rostocker Informatik-Berichte, 14, 1992.

Database Applications of Reflective Programming

Gottfried Vossen[16]
Universität Münster
Münster, Germany

We survey the use of reflection in various areas of computer science, with an emphasis on the area of databases. Besides its exploitation in implementing object-oriented languages and data models, little use of reflection has been made so far from a purely theoretical point of view; indeed, the idea vastly appears in two approaches only: the reflective relational machine of Abiteboul, Papadimitriou, and Vianu, and the reflective relational algebra of Van den Bussche, Van Gucht, and Vossen (which subsumes a third approach by Ross on an algebra). We give an overview of ideas, techniques, and results used and obtained for the latter, and discuss various applications in which the reflective algebra appears to be useful.

Exploring the Foundations of Interactive Computing

Peter Wegner
Brown University
Providence, USA

A computation is "interactive" if new information can be acquired and used during the process of computation and "algorithmic" if it yields a unique result (postcondition) for all input arguments (preconditions). The greater richness of interactive over algorithmic behavior is demonstrated by showing both that the observable behavior of objects cannot be expressed by algorithms and that life cycle requirements for programming in the large are nonfunctional. Objects can exhibit "passive" non-algorithmic behavior by "echoing" non-algorithmic input sequences or "active" non-algorithmic behavior that cannot be expressed by algorithmic transformations of input messages to outputs. Software systems are not simply large algorithms; they are interactive, open systems with non-algorithmic requirements and non-algorithmic behavior.

Algorithms and objects both specify infinite observable behavior by finite imperative code specifications. But the observable behavior of algorithms can, according to Church's Thesis, be specified by computable functions, while the observable behavior of objects cannot. The non-formalizability of observable object behavior is a strength rather than a weakness, allowing imperative object specifications to express richer behavior than formalizable algorithmic specifications. Interactive systems may have formalizable behavior for specific tasks or clients, but complete (fully abstract) observable behavior is too rich to be formalizable by functions or first-order logic.

Objects are open systems that incrementally acquire and use new information, while automata are closed systems with predefined input tapes. The delayed (unspecified) binding time of inputs is crucial in causing objects to exhibit non-algorithmic behavior while predefined

[16]Major portions of this talk are based on joint work with Jan Van den Bussche (University of Antwerp, Belgium) and Dirk Van Gucht (Indiana University, USA).

inputs constrain automata to behave algorithmically. The "proof" that objects exhibit non-computable behavior when they "echo" non-computable input sequences is trivial. Examples of "echo" systems include Simon's "ant on a beach", the Eliza program that simulates a passive psychiatrist, and the agent who wins half the games in a simultaneous chess tournament by echoing the moves of one player on the board of another. The "proof" that objects can exhibit non-algorithmic transformational behavior is also straightforward. Milner gives a version of this argument in his Turing lecture (CACM January 1993) and it has been part of the folklore of concurrent programming for much longer. However, the implications of the greater richness of interactive over algorithmic computation have not, to the author's knowledge, been previously explored.

The inherently non-algorithmic nature of interactive problem solving has radical consequences for the theory of computation. Turing machines can no longer be considered a universal model of computational behavior, while Church's thesis loses its force because functions are not rich enough to capture interactive computing. However, interactive problem solving is shown to be the the computational counterpart of empirical modeling in the physical sciences and therefore as robust and respectable in its scholarly pedigree as the rationalist algorithmic problem solving paradigm of logic and mathematics.

The correspondence of interactive models with open systems, programming in the large, and empirical models indicates that interaction is as robust in its invariance for a variety of definitions as algorithmic computability, and that canonical interaction machines playing the role of Turing machines for interactive systems probably exist. The Pi calculus, which is claimed by Milner to provide universal primitives for interaction, is a candidate universal interaction machine. The behavior analysis used to show that interaction is richer than algorithmic computation can be adapted to show that empiricism is richer than rationalism, thereby settling a 2000-year-old fuzzy philosophical question by a precise computational argument.

A Flexible Framework for Formal and Informal Conceptual Modeling

Roel Wieringa
Free University
Amsterdam, The Netherlands

The research reported here has three goals: to show that there is progress in the field of system development method research; to integrate formal and informal methods; and to define a flexible method that can accommodate components of structured and object-oriented methods. To this end, a framework is for system development methods is presented and a general structure for conceptual models of external system behavior is given. MCM (Method for Conceptual Modeling) is presented as an example method that fills in this framework and satisfies the stated goals. It models external system behavior by means of various techniques taken from structured and object-oriented methods: a function decomposition tree, context diagrams, a class diagram, life cycle diagrams, a system transaction decomposition table and a class dictionary. In addition to these techniques, MCM contains a variety of induction and evaluation methods from structured and object-oriented methods, from formal specification theory and from a philosophical analysis of the concepts involved. If required by the application, a formal specification of external system behavior can be produced in LCM (Language for Conceptual Modeling), a specification language based on order-sorted dynamic logic and process algebra, which allows a precise and formal specification that corresponds to the informal specification given by means of diagrams and tables. The talk ends with a discussion of the role of formal specification in method building and model building.

4 Summary of Working Groups

Working Group 1: Inheritance - Covariance vs. Contravariance

Chair: Roel Wieringa, The Netherlands

Participants: Herman Balsters, Sjaak Brinkkemper, Jan van den Bussche, Jürgen Ebert, Giovanni Guerrini, Marc Gyssens, Andreas Heuer, Gerti Kappel, Christian Laasch, Diana Sidarkeviciute

This brief note summarizes the results of a discussion group on covariance and contravariance in inheritance relations. Three questions were discussed, as summarized in the following three sections. Only the outcome of the discussion is given, not the road leading to this outcome.

1. What is the meaning of "inheritance"?

- **Code inheritance**; also known as implementation inheritance. It consists of reuse of code, e.g. code import. It is a relation between specifications (at the textual level). It can be shown that a queue specification can code inherit from a stack specification or vice versa. Covariance and contravariance have no meaning in this kind of inheritance.

- **Subtyping**; also known as specification inheritance. To give a precise definition, one must choose a formal framework (e.g. Cardelli-Wegner type theory or algebraic data type specification). To avoid religious disputes about what the best type theory is, some general statements were put forward on which all participants agreed.

  - Subtyping is a partial ordering on types.
  - If $\tau \leq \sigma$ then all terms of type τ can be used in any context in which terms of type σ can be used.
  - If $\tau \leq \sigma$, then instances of type τ have (at least) all properties that instances of type σ have. In order-sorted logic we have that the theory of the supertype is contained in the theory of the subtype, i.e. $\tau \leq \sigma \Leftrightarrow Th(\sigma) \subseteq Th(\tau)$. Subtyping therefore implies a subset relationship between intentions.

  We have that $SQUARE \leq RECTANGLE$ but $RECTANGLE \not\leq SQUARE$.

- **Specialization**; also known as *is_a* inheritance. This too is a partial ordering on classes. $\tau \leq \sigma$ iff each instance of τ is identical to one instance of σ. For instance, $CAR \leq VEHICLE$ because each car is (identical to) a vehicle. This is a subset relationship between extensions of classes.

There are two questions regarding the relationship between specialization and subtyping:

1. *Does specialization (is_a) require subtyping?* A consensus emerged that if you define a specialization, e.g. $PROF \;\; is\_a \;\; PERSON$, then you define a subset and therefore you at least inherit the structure (i.e. type) of the superclass. So the type of the subclass is a subtype of the type of the superclass. In other words, specialization requires subtyping.

2. *Does subtyping require specialization?* A consensus emerged that you can have a class C_1 of objects of type τ, and a class C_2 of type σ, such that $\tau \leq \sigma$ but C_1 is not a subclass of C_2. This issue is related to the fact that in programming languages, you do not want to conclude from structure equivalence of types that the types are identical (have the same instances).

Covariance and contravariance are meaningful for the subtype relationship (and therefore for the subclass relationship).

2. What is the meaning of covariance and contravariance?

Suppose $CAR \leq VEHICLE$ and $INTEGER \leq RATIONAL$, and declare the function *speed* as

$$speed : VEHICLE \rightarrow INTEGER.$$

Then speed also accepts arguments of type CAR and the results it delivers are also of type $RATIONAL$. So *speed* is also a function of arity

$$CAR \rightarrow RATIONAL.$$

If we denote the type of functions from τ tom σ by $\tau \rightarrow \sigma$, then we have

$$VEHICLE \rightarrow INTEGER \quad \leq \quad CAR \rightarrow RATIONAL.$$

So if we generalize a function type, we specialize its domain type. This phenomenon was noted by Cardelli in 1984 and is called **contravariance**. It was remarked by Herman Balsters in the discussion that when a function with argument type $VEHICLE$ is applied to an argument $c : CAR$, then the argument is coerced to a supertype before the function is applied to it.

When we inherit a method

$$speed : VEHICLE \rightarrow RATIONAL$$

from the class $VEHICLE$ to the class CAR, then we may want to restrict the codomain, so that we get for example a method

$$speed : CAR \rightarrow INTEGER$$

in the CAR class. This is **covariance**, for when we specialize, we specialize the domain as well as the codomain of the method. It was remarked by Herman Balsters that in covariance, when we apply a method to an object, we push the method down into the class hierarchy so that its domain is the type of the argument.

3. When do we need covariance and contravariance?

It was argued that for function inheritance, we need contravariance whereas for method inheritance, we need covariance. Contravariance is the natural thing we get when comparing function types. In method inheritance, by contrast, we do not compare function types, but we want to specialize a method for a particular subclass. In contrast to Cardelli's approach in his 1984 paper, this requires us to declare methods not as fields in a record, but as functions applicable to record types.

Suppose we have types *ColoredPoint* with attributes x, y, c and *Point* with attributes x and y. Then $ColoredPoint \leq Point$. Using an example given by Herman Balsters, Now define a method *eq* as

$$\lambda p_1 : Point.\lambda p_2 : Point.(p_1.x = p_2.x \wedge p_1.y = p_2.y).$$

In *ColoredPoint*, we override (or really extend) *eq* by a method which additionally tests the equality of the colors:

$$\lambda p_1 : ColoredPoint.\lambda p_2 : ColoredPoint.(p_1.x = p_2.x \wedge p_1.y = p_2.y \wedge p_1.c = p_2.c).$$

The typing rule for the equals sign (=) says that the left- and right hand side must have the same smallest type. Applying *eq* to two points or two colored points does not given any problem. At the same time, specialization uses here the covariance rule.

The typing of the *eq* method can be made explicit using Cardelli and Wegner's bounded quantification:

$$eq = \Lambda\alpha \leq Point.\lambda p_1 : \alpha.\lambda p_2 : \alpha.(p_1.x = p_2.x \wedge p_1.y = p_2.y).$$

Working Group 2: Observing Behavior of Objects

Chair: Gunter Saake, Germany

Participants: Vytautas Čyras, Grit Denker, Hans-Dieter Ehrich, Gregor Engels, Karl Lieberherr, Gianna Reggio, Wilhelm Schäfer, Michael Schrefl, Silke Seehusen, Bernhard Thalheim, Peter Wegner, Andreas Zamperoni

1. Starting Point

The working group was motivated by some problems and open points collected on the first day of the Dagstuhl workshop:

- Trace models for objects: What is the role of traces in dynamic models and object design?
- Behavioral semantics of objects: How should the complete behavior of objects be described?
- Observing object behavior: Should the rules for observing objects be explicitly modeled in object semantics?
- Local object behavior vs. global system behavior.

At the beginning of the meeting, the participants added two more points to this list:

- Alternatives to the trace model for objects, for example Labelled Transition Systems (related to the contribution of Gianna Reggio)
- *Concurrency*: The role of different models for concurrency for object behavior.

  The participants agreed that this point is of importance for the topic of the working group.

After the introductory discussion, we realized that the title of our working group gave rise to some very fundamental questions:

- What is the meaning of *Observation* ?
- What is an *Object* ?
- What is *Behavior* ?
- And what is the role of *Concurrency* for these concepts?

But it was suggested that discussing these concepts individually might be too general and that discussing the questions "What is the observable behavior of concurrent objects?" and "How should it be specified?" might be more specific.

One answer to this question was to specify the behavior of an object by its interaction history over its lifetime (also called its life cycle behavior). This behavior can in special cases be specified using *traces*. But traces are not adequate to specify time stamping requirements, fairness requirements, or concurrency requirements. The general question of how the life cycle behavior of objects should be specified has no easy solution and appears to be intractable in general.

2. What is an object?

The first oft the fundamental questions discussed in some detail was of course *"What is an Object?"*. The participants discussed some different definitions but finally only agreed (more or less) on a very general definition based on the presentation of Peter Wegner from Monday morning.

> *The state of an object (or equivalently the reaction on stimuli or the next local decision of an object) depends* **only** *on the local history (including the initial state).*

This definition is very general because it leaves room for defining the concepts of

- *locality*, and
- *history*.

The history may consist of a trace of external stimuli, an event structure modelling some concurrency in the past, or even more general models. As a discussion question the following problem arises:

What is a sufficient set of (past) observations to reproduce object behavior?

In fact, this definition only excludes a global state accessible by an object (locality property) and object behavior depending on the future. So the next discussion point was:

Is it possible to be more concrete about suitable models for object behavior? Does it make sense to restrict this very general idea to simpler models?

3. Models for Object Behavior

After some different statements what are appropriate models for object behavior the participants agreed to distinguish at least two kinds of models depending on the problem domains:

- General Models for (real world) Objects
- Tractable Models for (artificial) Objects

3.1. General Models for (real world) Objects

For modelling "real world" objects, ie, objects that may appear in the analysis of arbitrary application areas, we can give no real restrictions:

- For arbitrary problem domains we have to handle *full concurrency.*
- As a result, object histories are *more general than traces.*
- For these general models, observational equivalence is in general undecidable and in practice not tractable.
- **but:** *Objects are inherently concurrent!*

The participants agreed that such general models are the price we have to pay if we want to model real applications appropriately.

3.2. Tractable Models for (artificial) Objects

For artificial objects (ie, software artifacts), it seems to be a good idea to restrict the behavior models to more specific models.

- The proposal of Hans-Dieter Ehrich in his talk from Monday morning is a good candidate for such a model.
- *Local sequentiality* seems to be a good compromise for software objects.

  Local sequentiality means, that basic objects are sequential processes and concurrency appears only between objects. Hans-Dieter Ehrich presented a model based on event structures in his talk which has this property. This restrictions seems to be reasonable at the current stage of software technology.

- Such restriction make the design process and implementation easier, and enable the use of existing formal methods for sequential processes.

- **but:** *The restriction to sequential base objects seems to be too restrictive for analysis and in special application areas!*

4. Spectrum for Behavior Modelling

As a result of the discussion on the need of several model classes for object behavior depending on the application area, we discussed the spectrum we have in fixing a suitable model for object behavior:

- synchronous versus asynchronous communication

- degree of concurrency

- degree of (internal) nondeterminism

- inter-object or intra-object concurrency

The participants agreed that there is no "best choice" for the general problem.

The choice of a model with respect to this spectrum depends on the application area as well as the phase on the software construction process!

As a summary we come to the following observation:

General models are important....

[because objects are inherently concurrent!]

but simplification towards tractable models is necessary in software design.

[for example, local sequentiality for base objects.]

5. Discussion on Behavior Modelling

Besides the fundamental discussions on appropriate models for object behavior, the discussion touched a lot of topics concerned with behavior modelling. The following list is a summary of the areas discussed there:

- **Communication patterns.** We need abstraction mechanisms for patterns of cooperation in analogy to the mechanisms we have for structuring. Such patterns tend to be specific for application areas and are candidates for reuse.

- **Communication architecture.** Arbitrary interaction between objects makes large systems unmanageable. A restriction to manageable connection networks seems to be a solution to these problems.

Other areas which are touched in the discussion are *activity* (active versus re-acting objects) and *behavior evolution* (objects may persist longer than the specification of the application software is valid).

Working Group 3: Object Evolution

Chair: Roel Wieringa, The Netherlands

Participants: Herman Balsters, Grit Denker, Giovanna Guerrini, Radu Grosu, Andreas Heuer, Christian Laasch, Boris Magnusson, Jan Paredaens, Gianna Reggio, Pierre-Yves Schobbens, Roel Wieringa

This brief note summarizes the results of a discussion group on object evolution. Only the outcome of the discussion is given, not the road leading to this outcome.

1. We did not discuss class evolution. **Class evolution** is the phenomenon that a class changes its definition, i.e. it is a schema update. **Object evolution** is the phenomenon that an object changes its class without the definition of the class changing. This is distinct from **object updates**, in which an attribute changes its value.

2. The starting remark was that each attribute value defines a class, and that therefore each attribute update defines a migration through classes. Conversely, each partition of a class into subclasses defines an attribute with a enumeration range. A class migration causes an update of this attribute. In the extreme case, therefore, object evolution and object updates are equivalent.

The Martin/Odell development method takes the above view of attributes and class migration. Such a representation is however inconvenient and in practical data models, not all object evolutions will be object updates and not all updates will be evolutions.

3. If we want to allow class migration, we want to allow an object to have multiple roles, which it may play simultaneously. *Roles* have been given various other names in the literature, such as *aspects* and *dynamic subclasses*.

Wieringa proposed to define the concept of dynamic subclass. Each class can be partitioned in several ways into dynamic subclass partitions; each dynamic subclass partition is a partition of the state space of the object, and the object can, while moving through its state space, go from one dynamic subclass to another (see the ECOOP'94 paper by Wieringa, De Jonge and Spruit). Class migration methods can be allocated to the appropriate subclasses. For example, $PERSON$ could be partitioned into $STUDENT$ and $NON\_STUDENT$ with methods $finish$ and *become_student*, respectively.

Requiring a dynamic subclass to be an element of a partition was thought too artificial by the majority of the discussion group. Furthermore, most participants, agreed that *become_student* should be modeled as a database method (rather than being allocated to the database).

As a rebuttal against these arguments, it was argued that partitioning actually makes the model conceptually cleaner (cf. the abstract superclass rule) and that we need *NON_-STUDENT* exactly because it is an event in the life of a person (in the state of being a non-student). Nevertheless, we proceeded on the assumption that *become_student* would be a database method and that we would not require dynamic subclasses to be exhaustive partitions.

4. In most object-oriented programming languages, an object can be an instance of only one class. To allow class migration, we need to allow an object to be an instance of several classes at the same time (i.e. to be an element of the extent of several classes). We then need only two methods, each of which comes in two variants:

- *insert* an object into the extent of a class.
  - *create*: this creates a fresh object identifier (oid) and inserts it into an extent.
  - *start*: this takes an existing oid (already in at least one extent) and adds it to an extent.
- *remove* an object from the extent of a class.
  - *delete*: this removes the object from all extents it is currently in.
  - *finish*: this removes the object from some, but not all extents it is currently in.

For some cases of class migration, some of these methods must be executed in one atomic transaction (eg. moving the object from one extent to another). The insert methods would be database methods (or possible class methods).

The *finish* method should take care that all dangling pointers to an object in a particular role are removed.

5. The taxonomic structure of the class model defines some obvious constraints on class migration. For example, if we put an object into the extent of C, it must also be in the extent of all superclasses of C. A recent VLDB paper discusses some of these class migration constraints.

6. It was noted by Andreas Heuer that there is a distinction between dynamic subclasses for which an explicit start/finish pair of events is defined, and dynamic subclasses whose membership is determined by a predicate (*predicate classes*) or query (*derived classes*).

Boris Magnusson remarked that derived classes often offer a more accurate modeling tool than other kinds of subclasses. For example one can model $SQUARE$ as a subclass of $RECTANGLE$ on the grounds that the in $SQUARE$, $length$ and $breadth$ attributes are subject to an extra constraint $length = breadth$. On the other hand, we may say that $SQUARE$ is a *super*class of $RECTANGLE$, because it has only one attribute $length$, whereas $RECTANGLE$ has an additional attribute $breadth$. In both cases, the problem is that a rectangle may be updated so that $length$ happens to be equal to $breadth$. In both cases, this is represented incorrectly or at least clumsily. A better way would be to associate the condition $length = breath$ to the derived class $SQUARE$, which would thereby become a dynamic subclass of $RECTANGLE$.

7. Boris Magnusson remarked that instead of representing roles in the Simula way as nested subclasses. A rough example (in a non-existing programming language) would be:

```
class PERSON
  age: NATURAL
  r: ROLES
  become_student
    add new STUDENT to r

  role STUDENT
    ... body of STUDENT ...

  role EMPLOYEE
    ... body of EMPLOYEE ...
```

$STUDENT$ is also accessible from within $PERSON$. One can also easily implement inheritance in this way.

8. A different variant of class migration is obtained if we allow an object to play several roles *of the same class* simultaneously. (These may be called Roel's roles, as opposed to roles as discussed up till now. See also the ECOOP'94 by Wieringa et al.) For example, one may define a role class EMP with a many-one relationship (i.e. an object-valued attribute) $player : EMP \rightarrow PERSON$. EMP is almost the same as an object class, except for one thing: suppose the age function is defined for $PERSON$ objects but that it is applied to an EMP object e. Then $age(e)$ is a type error and the compiler should replace this by $age(player(e))$. This is a kind of compile-time delegation that is performed especially for roles but not for objects.

There were some doubts about the utility of this. For example, it was remarked that roles can be represented by set-valued attributes. A rebuttal to this is that this is equivalent to the above representation, because the intended set-valued attribute is equivalent to the *player* function.

There are ways in which we can avoid multiple role playing in the model. For example, perhaps a person is two employees at the same time because he or she is a secretary and engineer at the same time. Then we can define EMP subclasses $SECRETARY$ and $ENGINEER$, define the intersection class, and put the object in this intersection class (with a single identifier). It is then member of the $SECRETARY$ and $ENGINEER$ extents and can therefore be said to play these two roles. No conclusion was reached about the best representation.

Working Group 4: Class Organization

Chair: Karl Lieberherr, USA

Participants: Vytautas Cyras, Gregor Engels, Hans-Dieter Ehrich, Gerti Kappel, Gunter Saake, Michael Schrefl, Silke Seehusen, Diana Sidarkeviciute, Bernhard Thalheim, Gottfried Vossen, Peter Wegner, Andreas Zamperoni

The main topics discussed were

1. The general issues of class organization
2. Structure-shy class organizations
3. Roles

1. The general issues of class organization

Peter Wegner asked: What is a good class organization?

Gerti Kappel and Michael Schrefl have studied this issue from the point of view of coupling and cohesion. They studied, for example, three kinds of class coupling: component coupling, interaction coupling and inheritance coupling. Whereas component coupling deals with the structural and interaction relationships between object classes interaction coupling investigates the kind of interaction which takes place between object classes. They noticed tradeoffs between those coupling kinds, for example, when following the Law of Demeter, the component coupling improves in the sense that an object class does only talk to a restricted set of object classes. But the interaction coupling gets worse in the sense that it increases interaction of a class with its part classes.

Previous work on coupling was done by Kemerer/Chidamber, Wirfs-Brock, Berard, and Coad/Yourdon.

The Law of Demeter (LoD) was described as a style rule (among others) which reduces the dependency of individual member functions on the class structure. The LoD says that a member function M of class C should only use the functionality of "closely related" classes: the (stored and computed) part classes of C, the argument classes of M and the classes whose objects are created in M. In other words, you should only talk to friends, not to strangers.

Gerti Kappel was pointing out that there are situations where the LoD is beneficial and others where it makes sense to violate it. These situations can be analyzed from a semantic data model point of view. If the component structure of some class C comprises dependent components, which only exist in the context of C, or if the component structure captures mainly the implementation of C the Law of Demeter should be considered. If the components describe conceptual objects of their own, and if they are not considered as implementation details of some inspected class C but rather as different classes related to C it is favorable to reveal the component hierarchy thus violating the LoD.

Michael Schrefl was pointing out a further style rule which limits the set of classes who can update other classes. His style rule prevents communication among objects of the same level.

Peter Wegner was summarizing the pattern work. It describes and classifies generally useful class organizations. A specific pattern was discussed: The Null Object Pattern.

Beck and Johnson define patterns to have preconditions that define when a pattern is applicable, a problem specification of the problem that it solves, constraints that specify the principal sub-cases, and a solution structure that specifies the class organizations. Patterns relate class organization to the semantics of particular problems being solved.

2. Structure-shy class organizations

The title is somewhat controversial as pointed out by Andreas Zamperoni and Gottfried Vossen: How can a class organization be structure shy? Classes are thought to be specified in two parts: the structure part and the behavior part. When we write the behavior part with minimal dependence on the structure part, we say we have a structure-shy class organization. The trick to achieve this is to apply Polya's inventor paradox: instead of writing one class organization, we write an entire family of class organizations from which we select the desired one later. The paradox consists in the fact that, although we solve a more general problem, the solution becomes easier. The resulting solution is more generic than "usual" solutions.

Adaptive programs have been introduced at IFIP '92 and CASE '92 and a recent description is in Comm. ACM May 94, page 94. They are implemented in the Demeter System/C++ which is used at several universities.

Peter Wegner was pointing out that adaptive programs describe behavior minimizing information about the implementation in the form of a specific class structure. This is also the goal of his work on mega-programming with Gio Wiederhold. It is also the goal of Jacobson's use cases to describe behavior without being side-tracked by a detailed object-structure.

Gottfried Vossen was presenting the point of view of the data base community towards "schema-shy" queries. There are a number of papers on this topic in the literature:

- Catriel Beeri and Hank Korth, 1. ACM PODS '82
- E.J. Neuhold and Michael Schrefl, VLDB 1988
- Ed Sciore, TOIS '91, ER '91
- Chang and Ed Sciore, IEEE TKDE '90
- Jan van den Bussche, Gottfried Vossen, DOOD '93
- Ioannidis et. al, SIGMOD '94

The data base community has been working with abbreviated path expressions to reduce the amount of information on the schema needed for querying the database. The focus is on finding "minimal" paths according to some metric.

Gunter Saake pointed out the relationship between adaptive programs and views. The wrappers are working on the views. To use a collection of adaptive programs is related to using a collection of views with the same conceptual schema.

Gottfried Vossen was pointing out that work in the relational database field on view integration is related to coming up with a type theory for adaptive programs. In particular, views are used in bottom-up database design to model external schemas; the goal is to integrate them into a global conceptual schema. In the oo context, path expressions/adaptive programs could play a similar role. However, as known from relational database theory, such an approach can run into problems which are not computable. Specifically he was pointing to the work of Bernhard Convent "Unsolvable Problems Related To The View Integration Approach", International Conference on Database Theory, ICDT '86, Rome Italy, September 1986, Springer Verlag LNCS 243.

The situation with adaptive software is somewhat different than with view integration: When an adaptive program is developed, it is done in the context of a conceptual model which serves to test the adaptive program. This way it is guaranteed a priori that the integration can be done. However, when libraries of adaptive programs are used and programs from different libraries are combined, the issue of view integration comes up.

Hans-Dieter Ehrich pointed out the concept of logical data independence of the ANSI SPARC community (late seventies). Three kinds of schemas are used: external, conceptual and internal schemas for the implementation. It is possible to change the conceptual schema as long as it is still consistent with all the views. A similar situation exists in adaptive programs: The conceptual schema can be changed as long as it is consistent with all adaptive programs which use it.

Adaptive programs are more abstract than views since they are defined in terms of succinct subgraph specifications.

Gunter Saake pointed out the following references:

Dennis Tsichritzis and A. Klug, "The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Data Base Management Systems, Information Systems, Vol. 3, No. 3, 1978, pp 173-191.

DAFTG, Database Architecture Framework Task Group of the ANSI/X3/SPARC Database System Study Group: Reference Model for DBMS Standardization, ACM SIGMOD Records, Vol. 15, No. 1, 1986, pp 19-58.

Comparison of the database work and the work on adaptive software

- Common:
  - Motivation to be structure-shy.
  - Focus on paths in conceptual data model.
- Database work:
  - Focus on non-computationally complete query notations.
- Adaptive software work:

- Focus on a computationally complete programming language which extends C++, Smalltalk or Eiffel.

Gregor Engels pointed out that a collection of adaptive programs can be considered as a module which has an additional, new interface not present in ordinary modules: This interface contains the succinct subgraph specifications of all adaptive programs contained in the module. In the simplest form, the succinct subgraph specifications satisfy the grammar S::= [A,B] — S+S — S*S.

The issue of how to combine adaptiveness with concurrency was discussed briefly. Gerti Kappel suggested that the concept of transactions be added. Adaptiveness and concurrency have been combined in an ECOOP '92 paper.

Silke Seehusen viewed adaptive software as a useful tool to specify the behavior of frameworks. Indeed, an adaptive program can describe an entire family of frameworks.

3. Roles

Bernhard Thalheim presented the following approach to roles:

Objects are described by their values and references and belong to classes. They can be involved in different roles during their life cycle, sometimes in parallel and several times. In this case the 1-1 allocation of objects to one class is not sufficient. Roles can be presented directly in the SAMT language which is is developed in accordance to C. Beeri's Kyoto proposal for oo languages. Since each object has an identifier, it is possible to define class and type hierarchies which are not in a sub-hierarchy relationship. This possibility can be used for modeling roles and aspects of objects in higher granularity. In this case role or aspect classes are subclasses. The modular design-by-units approach of our system uses roles and aspects for class organization and behavioral views.

Michael Schrefl was presenting the concept of roles treated in several recent papers to cope with inflexibilities of class hierarchies.

Gottlob, Kappel and Schrefl (East-West Database Workshop) studied the semantics of role inheritance and type inheritance using evolving algebras. Albano et al. (VLDB '93) showed how roles can be provided in a new data base programming language called Fibonacci. Wieringa (ECOOP '94) stressed the importance of distinguishing traditional classes, roles with role-specific identifiers, and dynamic classes to which instances of classes are associated dynamically. Gottlob, Schrefl, Rueck (ACM TOIS, in print) showed how class-based languages such as Smalltalk can be extended with roles.

Working Group 5: Database Models

Chair: Gottfried Vossen

Participants: Herman Balsters, Giovanna Guerrini, Marc Gyssens, Christian Laasch, Karl Lieberherr, Jan Paredaens, Gunter Saake, Michael Schrefl, Diana Sidarkeviciute, Jan Van den Bussche

The focus of our discussions was not on data models and databases in general, but mostly on object-oriented data models. Specifically, two broad aspects were discussed, first the practical point of view, then the theoretical point of view.

Although most of the participants are not practitioners, we were able to make several observations on the practical use of object-oriented (OO) databases:

1. It is easy to see that OOA, OOD, and OOP are in wide use in practice, but when people using or applying OO technology need a database, they still switch to SQL, i.e., a relational one. So it seems that people are not attracted to OO databases. Although their concepts are powerful, systems incorporating these concepts are not mature enough.

2. Many people feel the need for OO, but not for OO *databases*. Indeed, most database applications are still happy with the tabular form of data representation offered by the relational model, possibly augmented with OO features. So it is not surprising that many people feel the next generation of database systems will be OR systems, which combine object and relational aspects.

3. Personally, one of my favorite arguments is that CAx technologies, which were among the triggers for the development of OO database systems some 10 years ago, are still not using what is commercially available. And worse, present-day publications do not even refer to these applications any more when discussing or illustrating features of a model or a system. It should be mentioned, however, that there are a few exceptions to this (e.g., CAD systems using an OO database).

4. Relational databases have required lots of implementation efforts and even considerable financial investments, so many companies hesitate to throw that away. A prominent example is IBM, who recently "bought" an OO database system for incorporation into company projects, instead to develop one from scratch.

5. In the world of OO databases, there are no people who do for this area what Codd and Date did for the relational field.

A final remark in this context, not directly practical, is that it can even be observed that, at least at the moment, OO papers seem to vanish from the major database conferences.

From a more theoretical perspective, the most interesting feature of this working group was that we had people among the participants representing six different groups which have be developing, or still develop, an OO data model. I now list these models (in random order) and indicate what their most distinguished features are, according to their inventors:

1. GOOD (Antwerp): the emphasis is on graph-structured databases and on the idea of patterns: if access to an information system is desired, the user typically knows some properties he or she is looking for, and searches for items in the systems following that pattern.

2. Troll (Braunschweig): this is actually not a data model in the strict sense; emphasized is the view that objects are communicating processes. Troll combines this view with OO concepts (e.g., inheritance).

3. COCOON (Ulm): here the focus is on operations; the model takes advantage of (nested) relational algebra and relations, but provides operations with more semantics. Also emphasized is that the data model part and the operational part should not be separated.

4. CHIMERA (Milano): here the focus is on rules and on the combination of deductive and OO features; furthermore, it is on activeness in databases. Rules are used both for expressing semantics in a data model, and for modeling active parts of a database.

5. TM (Twente): here the motivating observation was that some form of mathematics is missing in existing models. So the emphasis here is on a mathematically sound type theory; in addition, the TM model knows various kinds of constraints, and treats methods as part of a database schema.

6. Demeter (Boston): the major aspects is that binding of methods to classes is delayed, through the use of succinct (sub-)graph specifications.

Other than the distinctive features of each model listed above, all these (and even other models not mentioned here) have a number of commonalities, which indicates that a convergence in this area is in sight. However, it seems important that researchers in the area do not get blinded by OO stuff, thereby forgetting about core database features:

- Contrary to programming languages, database systems are able to do *optimization*; in particular, they can optimize access times to objects, which is irrelevant in programming languages.
- Databases allow to *share* things between different users or applications. This, in turn, requires high-level languages, which, as said, must then be optimizable.

As a consequence, the current discussion about how to deal with methods in OO databases can benefit from remembering these basic aspects; indeed, methods in an OO database might even use a restricted language, as along as they are optimizable. A database method language needs not be computationally complete; that can be captured in the outside application.

I mention that we also briefly discussed views in OO databases, which are not simple extensions of relational views since methods are present. An agreement seemed to exist on that a view restricts database access: for methods, this can mean forbidding the use of some methods, or the derivation of new methods (in a view) in terms of given ones.

A general agreement was that the foundations of OO databases are still in their infancy. What will really happen in the future and survive from the developments and proposals so far will, as usual, heavily depend on the commercial world. However, that should not keep academia away from producing new ideas and experimenting with them.

Working Group 6: Software Engineering Models

Chair: Sjaak Brinkkemper

Participants: Vytautas Cyras, Grit Denker, Juergen Ebert, Gregor Engels, Gerti Kappel, Boris Magnusson, Gianna Reggio, Wilhelm Schaefer, Silke Seehusen, Diana Sidarkevicuite, Bernhard Thalheim, Peter Wegner, Roel Wieringa and Andreas Zamperoni

Optimal set of OO specification techniques

First, the motivations for splitting the complete system specification into several partial specifications were discussed. This splitting, also called factorization, can be according to various stages in the systems life-cycle: the horizontal factorization, or according to the system perspectives: the vertical factorization. There are several motivations for factorizations:

- Reduction of working complexity.
- Visual (graphical) restrictions in modelling.
- Distinct aspects to be modelled.
- Communication medium.
- Support for the abstraction and information gathering processes.

For one system it turns out that there exist different suitable factorizations of the specification. The choice for the most appropriate one is made by development team influenced by the application domain characteristics and the functionality of the design tools. The phenomenon of factorization is also present in other engineering areas, such as mechanical engineering, civil engineering, and architecture. The different specialties in the design team (e.g. database designers and user-interface designers) need their own formalisms.

The optimal set of specification types gave four different viewpoints:

1. Conceptual viewpoint (Roel Wieringa): structure, dynamics, and communication
2. Design process viewpoint (Peter Wegner): structure, dynamics and functional
3. Systems development viewpoint (Sjaak Brinkkemper): instance, class structure, life cycle, subsystems, class communication, event triggering, functionality, implementation system
4. Teaching viewpoint (Juergen Ebert): the paradigms of: entity relationship, data flow, control flow, state transition, and logic

The core of specification was discussed in depth. The suitability of functional model by means of Data Flow Diagrams was questioned. The necessity of tools for the simulation and animation of dynamic specifications was generally accepted. Furthermore, experience shows that an intermediate specification level above the level of object classes is needed to obtain an overview of the system structure.

Tools support to show the multiple views on screen and to record accordingly in repository. Procedural support for the assistance of designers during the specification process can be incorporated in the tool as well. From the viewpoint of teaching the toolkit approach was advocated. This toolkit approach implies that designers are able to work with a set of high quality specification formalisms that can be employed according to circumstances and to the application domain.

OO metrics

There is little experience with quality metrics for object oriented designs. Booch mentions: coupling, cohesion, sufficiency, completeness and primitive, but gives no measurement function. Furthermore, all but coupling are qualitative metrics.

Chidamber and Kemerer [1] have proposed a set of OO metrics. They have not collected actual numbers for the metrics yet, so these have not been validated. The proposed metrics are among others: depth of inheritance tree, number of children, coupling between objects, response for class, weighted methods per class, lack of cohesion in methods, object-to-root depth, object-to-leaf depth, fan-in/fan-out, and used-by/uses relationships of objects.

Metrics on complexity of a specification technique and for the aggregate level of a complete method are under development. In this case concrete figures have to be evaluated in order to establish insight.

We ended with some wise words on the dangers of metrics. Metrics should not substitute thinking. Moreover, it makes no sense to apply metrics that are based on poor knowledge.

[1] Shyam R. Chidamber, Chris F. Kemerer: A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, vol. 20, no. 6, June 1994, pp. 476-493

Acknowledgement

The organizers are grateful to Andreas Zamperoni who helped a lot in organizing the workshop and editing this report.

Dagstuhl-Seminar 9434 List of Participants

Herman **Balsters**
University of Twente
Computer Science Department
P.O. Box 217
7500 AE Enschede
The Netherlands
balsters@cs.utwente.nl
tel: +31-53-893-772

Sjaak **Brinkkemper**
University of Twente
Center of Telematics and Information Technology
P.O.Box 217
7500 AE Enschede
The Netherlands
sjbr@cs.utwente.nl

Jan Van den **Bussche**
University of Antwerp (UIA)
Dept. of Mathematics and Computer Science
Universiteitsplein 1
2610 Antwerp
Belgium
vdbuss@wins.uia.ac.be
tel: +32-3-8202-417
fax: +32-3-8202-421

Vytautas **Cyras**
Vilnius-University
Faculty of Mathematics
Naugarduko 24
2600 Vilnius
Lithuania
vytautas.cyras@maf.vu.lt
tel: +370-2-636-035/028

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

Jürgen **Ebert**
Universität Koblenz-Landau
Institut für Softwaretechnik
Rheinau 1
56075 Koblenz
Germany
ebert@informatik.uni-koblenz.de
tel: +49-261-9119-412
fax: +49-261-9119-499

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

Gregor **Engels**
Leiden University
Dept. of Computer Science
P.O. Box 9512
NL-2300 RA Leiden
engels@wi.leidenuniv.nl
tel: +31-71-27 7063
fax: +31-71-27 6985

Radu **Grosu**
Technische Universität München
Fakultät für Informatik
Arcisstr. 21
80290 München
Germany
grosu@informatik.tu-muenchen.de
tel: +49-89-2105-2398

Giovanna **Guerrini**
Università di Genova
Dip. di Informatica e Scienze dell'Informazione
Viale Benedetto XV, 3
16132 Genova
Italy
guerrini@disi.unige.it
tel: +39-10-3538783
fax: +39-10-3538028

Marc **Gyssens**
University of Limburgs
Dept. WNI
3590 Diepenbeek
Belgium

Hele-Mai **Haav**
Estonian Academy of Sciences
Institute of Cybernetics
Department of Software
Akadeemia tee 21
EE-0026 Tallinn
Estonia
helemai@cs.ioc.ee
tel: +3722-2-527-314

Andreas **Heuer**
Universität Rostock
Lehrstuhl Datenbank & Informationssysteme
18051 Rostock
Germany
heuer@informatik.uni-rostock.de

Gerti **Kappel**
University of Linz
Institute of Computer Science
Altenbergerstraße 69
4040 Linz
Austria
gerti@ifs.uni-linz.ac.at
tel: +43-732-2468-879

Christian **Laasch**
University of Ulm
Facultät für Informatik
Abt. Datenbanken & Informationssysteme
89081 Ulm
Germany
laasch@informatik.uni-ulm.de
tel: +49-731-502-4135
fax: +49-731-502-4134

Karl **Lieberherr**
Northeastern University
Center for Software Sciences
College of Computer Science
125 Cullinane Hall
Boston, MA 02115-9959
USA
lieber@ccs.neu.edu
tel: +1 (617) 373 2077
fax: +1 (617) 373 5121
ftp: ftp.ccs.neu.edu
pub/people/lieber
pub/research/demeter

Boris **Magnusson**
Lund University
Dept. of Computer Science
P.O. Box 118
22100 Lund
Sweden
boris@dna.lth.se
tel: +46-46-108044

Jan **Paredaens**
University of Antwerpen
Dept. of Math. and Computer Science
Universiteitsplein 1
2610 Antwerpen
Belgium
pareda@wins.uia.ac.be
tel: +32-3-820-2401

Gianna **Reggio**
Universita' di Genova
Dip. di Informatica e Scienze dell'Informazione
Viale Benedetto XV,3
16132 Genova
Italy
email: reggio @ disi.unige.it
tel: +39-10-3538032
fax: +39-10-3538028

Gunter **Saake**
Otto-von-Guericke Universität Magdeburg
Fakultät für Informati
Inst. für Technische Informationssysteme
P.O. Box 4120
39016 Magdeburg
Germany
saake@iti.cs.tu-magdeburg.de
tel: +49-391-5592-3800

Wilhelm **Schäfer**
Universität Paderborn
FB 17 - Informatik
33098 Paderborn
wilhelm@uni-paderborn.de
tel: +49-5251-60 2428

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

Silke **Seehusen**
Fachhochschule Lübeck
Fachbereich Elektotechnik
Stephensenstr. 3
silke@acm.org
tel: +49-451-500-5219

Diana **Sidarkeviciute**
Vilnius-University
Faculty of Mathematics
Department of Informatics
Naugarduko 24
2600 Vilnius
Lithuania
diana.sidarkeviciute@maf.vu.lt
tel: +370-2-636-035/028

Bernhard **Thalheim**
Cottbus Technical University
Computer Science Institute
P.O.Box 101344
03013 Cottbus
Germany
thalheim@informatik.tu-cottbus.de

Gottfried **Vossen**
Universität Münster
Institut für Wirtschaftsinformatik
Grevenerstr. 91
48159 Münster
vossen@uni-muenster.de
tel: +49-251-9275-103
fax: +49-251-839754

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

Roel **Wieringa**
Vrije Universiteit Amsterdam
Faculty of Mathematics and Computer Science
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
roelw@cs.vu.nl
tel: +31-20-4447771

Andreas **Zamperoni**
Leiden University
Dept. of Computer Science
P.O. Box 9512
2300 RA Leiden
The Netherlands
zamper@wi.leidenuniv.nl
tel: +31-71-27-7103
fax: +31-71-27-6985