Contents

1 Preface

During the week of March 04 - 08, 1996, the Seminar on New Trends in Database Languages was organized by Anthony J. Bonner (University of Toronto, Canada), Andreas Heuer (University of Rostock, Germany), and Letizia Tanca (University of Verona, Italy). Participants came from Universities or Research Centers from Belgium (4), Canada (2), Germany (15), Great Britain (1), Italy (4), Switzerland (1), The Netherlands (2), and USA (2).

Altogether 22 lectures and 2 working groups covered various aspects of new trends in database languages. Areas of particular interest were foundations of database languages and extensions of the standard languages.

The participants appreciated the outstanding local organization and the environment including all Dagstuhl facilities which enabled a successful workshop.

The seminar has its own web-site. Today (in December 1996), it is located under the URL `http://wwwdb.informatik.uni-rostock.de/dagstuhl_9610/`. There, all abstracts and links, some of the slides, and further full papers are electronically available. The link has also been added to the Dagstuhl web-sites (which can be found under the URL `http://www.dag.uni-sb.de/`).

1.1 Topics

In relational database systems, SQL has become the accepted standard as a general database language for data definition, queries, updates, and database programming (in its embedded version). More recently, several new directions for database languages have been developed

- for the specification and design of database applications,
- for more complex queries than expressible in standard SQL,
- for deeply structured and long transactions,
- for a seamless integration of database operations and application programs,
- for the use with new database models like object-oriented ones.

These new directions are especially motivated by new and more-complex applications like CAD, CAM, CASE, office automation, and scientific databases.

These trends lead to

- newer versions of SQL like SQL3 and Object SQL (OQL),
- deductive approaches to database querying like rule-based languages or database logics,
- rule-based approaches for active database applications,

- transaction languages or models for different data models,
- special-purpose specification languages,
- integrated database programming languages,
- object-oriented calculi, algebras, or programming languages with persistence.

The workshop was to bring together researchers working on foundations of database languages, especially for the relational and object-oriented database models, and researchers developing extensions of the standard languages like DATALOG and SQL, among them projects like object logic (F-Logic) or extended SQLs (SQL3, Object SQL or OQL).

The discussions at the workshop aimed to result in a deeper understanding of the forthcoming developments in the database language area and the usefulness of fundamental results in complex applications. There were special sessions on developing canonical examples for advanced applications of database languages, which captured the essence of an advanced application and reported about advanced applications showing the usefulness and problems of new database languages.

2 Abstracts

2.1 Talks

An Orthogonally Persistent Java

Malcolm Atkinson[1]
University of Glasgow, Great Britain

The language Java is enjoying a rapid rise in popularity as an application programming language. For many applications an effective provision of database facilities is required. Here we report on a particular approach to providing such facilities, called "orthogonal persistence". Persistence allows data to have lifetimes that vary from transient to (the best approximation we can achieve to) indefinite. It is orthogonal persistence if the available lifetimes are the same for all kinds of data. We aim to show that the programmer productivity gains and possible performance gains make orthogonal persistence a valuable augmentation of Java.

Verification of transactions in object-oriented databases

Herman Balsters[2]
University of Twente, The Netherlands

This abstract:
`http://wwwis.cs.utwente.nl:8080/~keulen/Diversen/DagstuhlAbstract.html`
OODM project: `http://wwwis.cs.utwente.nl:8080/oodm.html`
Group publications: `http://wwwis.cs.utwente.nl:8080/isdoc/isdoc.html`

Transactions on a database can change the state of a database in the case of an update. After invocation of such a transaction it is always the question whether the integrity constraints of the database have remained intact. What we are concerned with is how to verify (i.e. *prove*) that an update operation on a database state results in a new state that still satisfies the set of (static) integrity constraints of the database. Moreover, we want to perform this verification process irrespective of any particular input state, and we also wish to offer automatic support in the process. In doing so, we shall offer a compile-time verification of update operations written as part of the schema of an object-oriented (O-O) database. The language used for

[1] Joint work with L. Daynhs, M.J. Jordan, T. Printezis, and S. Spence.
[2] Joint work with David Spelt.

describing the O-O database schema is called TM, and has been developed at the University of Twente. TM (cf. [BBZ93]) is a strongly typed functional database language used for specification of O-O database schemas. Operations (methods) on TM database states are part of the database schema and can operate on different levels, namely on the object-, class-, and database levels. We have invoked the Isabelle interactive proof development system (cf. [Pau94]) to provide automatic support for proving correctness of TM database transactions. A large part of the research involved in a project like this is to offer a coding of some formal specification language (such as TM) into the Isabelle proof checker. Once done, we have been able to claim the following results. First of all, the proof tool can find a *minimal precondition* for any given database update method, such that if the input state satisfies this condition then invariance of the integrity constraints is guaranteed. Secondly, the proof tool can, as an alternative, generate so-called *compensating actions*; i.e. by altering the output state (after invoking the update method) according to this minimal set of compensating actions, it is again guaranteed that invariance is satisfied. A large benefit in database design gained by using such a proof tool is that, given some method M, it is determined which constraints have to be satisfied *that are as local as possible* to M such that the *global* integrity constraints are guaranteed to be satisfied after ivokement of M.

References:

[BBZ93] H. Balsters, R.A. de By, and R. Zicari. "Sets and constraints in an object-oriented data model". Proceedings Seventh European Conference on Object-Oriented Programming (ECOOP), Kaiserslautern, Germany, July 1993

[Pau94] L.C. Paulson. "Isabelle: A generic theorem prover". Lecture Notes in Computer Science #828, Springer Verlag, Berlin, 1994.

Concurrent Transaction Logic

Anthony J. Bonner[3]
University of Toronto, Canada

Related papers: `http://db.toronto.edu:8020/transaction-logic.html`
Full paper: `ftp://db.toronto.edu/pub/bonner/papers/transaction.logic/jicslp96.ps`

In an earlier work, we developed sequential Transaction Logic, which deals with state changes in logic programs and databases. It provides a framework for tasks ranging from transaction specification and execution in databases, to view updates, to triggers in active databases, to discrete system simulation, to robot planning and procedural knowledge in AI. In the present paper, we propose Concurrent Transaction Logic (abbr. CTR), which extends Transaction Logic with connectives for

[3]Joint work with Michael Kifer (SUNY at Stony Brook).

modeling the concurrent execution of complex actions. The concurrent actions execute in an interleaved fashion and can also communicate and synchronize themselves. All this is provided in a completely logical framework, including a natural model theory and a proof theory. Moreover, the framework is flexible in that it can accommodate many different semantics for updates and for databases. For instance, not only can updates insert and delete tuples, they can also insert and delete null values, rules, or arbitrary logical formulas. Likewise, not only can databases have a classical semantics, they can also have the well-founded semantics, the stable-model semantics, etc. Finally, the proof theory for CTR has an efficient SLD-style proof procedure. As in the sequential version of the logic, this proof procedure not only finds proofs, it also executes concurrent transactions, finds their execution schedules, and updates the database. A main result is that the proof theory is sound and complete for the model theory.

Towards a Formal Analysis of ODMG's Object Query Language

Stefan Conrad
Otto-von-Guericke-Universität - Magdeburg, Germany

Among the standardization efforts in the area of object-oriented database systems the proposal of the Object Database Management Group (ODMG) seems to be the most popular one. Among other topics, the ODMG-93 standardization proposal describes the query language OQL (Object Query Language). Unfortunately, the description is only informal. Therefore, we have tried to give a formal semantics to the pure query part of OQL. The formal semantics is based on a model of complex values which can be atomic values, collections (set, bag, list, array) of values as well as structured values. Object identities can be treated as values of special data sorts. This model is strongly typed. During the definition of the formal semantics several problems and questions arise. In the talk we focus on the presentation and discussion of a number of these problems.

Some of the most essential observations are as follows: The collection type "array" is not really supported by OQL. Several (useful and sometimes necessary) conversions functions are simply missing, e.g. for converting arrays. The group-by and group-by-with operations of the original ODMG-93 proposal can be considered as syntactic sugar because they can always be replaced by select-from-where queries with nested queries. The Cartesian Product over (different) collection types, especially for list types, is defined in an unintuitive way. The use (and necessity) of class indicators is not clear from the language description. The typing of query expressions is not really defined in the proposal. For instance, there are several possibilities to define the type of unions or intersections of collections over different object types.

In conclusion, there are a number of open problems and questions which have

to be answered by the ODMG people. Otherwise, the result of this standardization effort will be that all vendors claim to fulfill the ODMG standard but offer incompatible implementations.

Designing Applications with Objects and Rules: the IDEA Methodology

Piero Fraternali
Politecnico di Milano, Italy

IDEA book: `http://www.elet.polimi.it/idea`
IDEA Web Lab: `http://www.txt.it:5858`
IDEA project HomePage: `http://www.txt.it/idea`

The IDEA Methodology is a novel methodology for the the development of information systems. The IDEA Methodology addresses the analysis, design, prototyping, and implementation of database applications, with a special focus on the use of advanced features of database technology, and specifically of object orientation, deductive rules, and active rules (triggers). It takes advantage of well established approaches matured in the context of database design, but also of proposals coming from the broader context of object-oriented software engineering.

The distinguishing approach of the IDEA Methodology is the emphasis on KNOWLEDGE INDEPENDENCE; by this term, we indicate the ability of extracting semantic knowledge from applications, normally encoded in a procedural format (programs), and placing it into the database schema, encoded declaratively in the form of objects and rules; in this way, knowledge is system-enforced, shared by all applications, and can be maintained and evolve more easily.

An integrated tool environment, developed at the Politecnico di Milano, supports the various phases of the IDEA Methodology. The IDEA tools assist schema design, active rule generation, rule analysis, application prototyping, debugging, browsing, and the mapping active rule applications from the IDEA design language Chimera into Oracle V. 7.2, a popular commercial relational product supporting triggers. These tools will be demonstrated during the tutorial.

The IDEA methodology is described in the book "Designing database applications with objects and rules: The IDEA Methodology", that will be published by Addison-Wesley in the beginning of 1997. It is also described in a number of recent publications, listed below.

The IDEA methodology official Web site can be seen at the first of the above URLs. In addition, by April 1997 two additional initiatives will be completed: the IDEA Web Lab, an Internet site featuring a JAVA implementation of the IDEA design environment (now under development; URL see above); and the IDEA CD-ROM, a multimedia tutorial of the methodology.

The IDEA Methodology is one of the main results of the IDEA project, a four-year Esprit Project sponsored by the EEC with academic, research, and industrial partners from six countries (see URL above). Several case studies and experiences of use by the industrial partners have influenced the development of the methodology. The presentation has offered an extensive tour of the IDEA Methodology and its supporting environment and initiatives.

The following topics were be covered:

```
Introduction and Motivation
Overview of the IDEA Methodology
Presentation of a Case Study
The IDEA Models:
     The Object Model
     The Dynamic Model
     The CHIMERA Model and Language
The IDEA Process
     Analysis
     Design
          Schema Design
          Active & Deductive Rule Design
     Prototyping & Verification
          Rule Analysis
     Implementation
The IDEA Tools
     The Design Environment
          IADE
          ARACHNE
          ARGONAUT
          PANDORA
     The Protoytping Environment
          The CHIMERA Execution Environment
Summary and Conclusions
```

References:

P. Fraternali and S. Ceri. "Designing Applications with Objects and Rules: the IDEA Methodology". To be published by Addison-Wesley, 1997.

J. Widom and S. Ceri. "Active Database Systems: Triggers and Rules for Advanced Database Processing". Morgan-Kauffmann, San Mateo, 1996.

E. Baralis, S. Ceri, and S. Paraboschi. "Modularization Techniques for Active Rules Design". 'ACM Transactions on Database Systems', 21:1, March 1996.

P. Fraternali and L. Tanca. "A Structured Approach for the Definition of the Semantics of Active Databases". 'ACM Transactions on Database Systems'. 20:4, December 1995.

S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. "Automatic Generation of Production Rules for Integrity Maintenance". 'ACM Transactions on Database Systems'. 19:3, September 1994.

E. Baralis, S. Ceri, P. Fraternali, and S. Paraboschi. "A Support Environment for Active Rule Design". To appear on: 'Journal of Intelligent Information Systems'.

E. Baralis, S. Ceri, P. Fraternali, and S. Paraboschi. "Designing Active Rule Applications: Issues and Approaches". Proc. of Int. Conf. on Deductive and Object-Oriented Databases - DOOD'95, Singapore, dicembre 1995.

E. Baralis, S. Ceri, and S. Paraboschi. "Run-time Detection of Non-Terminating Active Rule Systems". Proc. of Int. Conf. on Deductive and Object-Oriented Databases - DOOD'95, Singapore, dec. 1995.

S. Ceri, P. Fraternali, S. Paraboschi, and G. Psaila. "The Algres Testbed of Chimera: An Active Object-Oriented Database System". Proc. of ACM SIGMOD 1995, San Jose, California, may 1995.

E. Baralis, S. Ceri, and S. Paraboschi. "Improved Rule Analysis by means of Triggering and Activation Graphs". Proc. of the Second Int. Workshop on Rules in Database Systems, Atene, Grecia, sept. 1995.

E. Baralis, S. Ceri, and S. Paraboschi. "ARACHNE: A Tool for the Analysis of Active Rules". Proc. of the Second Int. Conference on Applications of Databases - ADB'95, Santa Clara, dec. 1995.

Objects and Views in Deductive Databases

Burkhard Freitag
University of Passau, Germany

Group HomePage: `http://www.uni-passau.de/~freitag/`

We interpret a class as a logic theory, or – in more operational terms – as a module containing rules and facts that describe what is known about a collection of objects. Similarly, an object is represented as a logic theory that describes the object's state and delegates method calls to its home class theory. The representation of methods as rules, which has been proposed by several researchers and which we adopted in our approach, is not only applicable to retrieval methods but also to update methods. Within this framework, an object-oriented deductive database is modeled as a collection of logic theories that communicate with each other via contextual goals.

We propose a simple yet flexible method resolution scheme based on an appro-

priate parameterization of the module referred to in a method call. In our approach an object may be treated as a (virtual) instance of *any* virtual class provided that it satisfies the corresponding class constraint. A virtual instance changes its interface and behavior according to the particular view taken.

There is no need for global modifications when a single class is added or changed, i.e. objects and classes can be incrementally compiled. Optimization techniques of deductive databases carry over unchanged. Moreover, encapsulation and information hiding come for free provided a suitable notion of module is adopted. Our approach is extensible towards multi-agent systems in a natural way.

Tables As a Paradigm for Querying and Restructuring

Marc Gyssens[4]
University of Limburg, Belgium

Tables are one of the most natural representations of real-life data. Previous table-based data models (such as relational, nested relational, and complex objects models) capture only a limited variety of real-life tables. In this paper, we study the foundations of tabular representations of data. We propose the tabular database model for handling a broad class of natural data representations and develop `tabular algebra` as a language for querying and restructuring tabular data. We show that the tabular algebra is complete for a very general class of transformations and show that several languages designed for very different purposes can naturally be embedded into the tabular model. We also demonstrate the applicability of our model as a theoretical foundation for on-line analytical processing, an emerging technology for complementing the robust data management and transaction processing of DBMS with powerful tools for data analysis.

[4]Joint work with L.V.S. Lakshmanan and I.N. Subramanian, Computer Science Dept, Concordia University, Montreal, Quebec, Canada.

What's a Query? (What's a Program?) Simple Question?

Andreas Heuer
University of Rostock, Germany

Group HomePage: http://wwwdb.informatik.uni-rostock.de/(en)/index.html

Query languages for the relational database model mostly had an expressive power equivalent to the relational algebra or calculus. While "real" languages like SQL provided some aggregate functions and arithmetic operations besides grouping on top of this basic functionality, query languages for new database models seem to be

- either too simple (equivalent to a subset of conjunctive queries, only consisting of selections and some kind of joins), or
- too complex (if PL-SQL-programs, ObjectStore-DML-programs or SQL3- programs can be seen as "queries").

In the relational model, a query language is closed and adequate [HS91]: the result of the query consists of a new (relation) type and a new (relation) instance. In a formal OODB model, a query language is also closed and adequate, 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 ODMG standard [Cat94] or other existing OO query languages which only allow to collect sets of objects of a given class or values of some type. Object creation and derivation of classes is not possible.

Why this restriction? Is it not possible to define "real views" in these modern database models?

On the other hand, old and new query languages mostly rely on the "you-have-to-know-the-scheme" paradigm. While querying the World Wide Web, you do not have a scheme at hand but only constants. Is querying the Web a database problem? Or is it "only" an information retrieval problem because of the "content-based" style of queries (without giving meta information additionally to the constants).

In newer approaches of query languages, the following additional problems should be solved:

- Allowing mixed data-metadata-querying (F-Logic, reflective approaches, ...),
- allowing query optimization in presence of methods within a query (better interfaces for DataBlades, Database Extenders, or similar approaches),

- allowing a mobile (context-dependent) access to the database performing content-dependent query interpretation, query transformation, and query optimization.

For further information on different projects in these areas, you can visit the WWW homepage of the Rostock Database Research Group (see URL above).

References:
[Cat94] R.G.G. Cattell, editor. "The Object Database Standard: ODMG-93". Morgan-Kaufmann, San Mateo, CA, 1994.
[HS91] 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.

Query Languages of Object-Oriented Database Systems

Uwe Hohenstein
Siemens AG, München, Germany

During the last years, several commercial object-oriented DBSs (database systems) have established themselves on the database market. This new database technology mostly stress on a navigational kind of access, hangling from one object to others. Nevertheless, currently available systems incorporate associative query languages more and more. But relying on rather dissimilar approaches, notations, and paradigms, query languages can hardly be assessed with regard to their expressiveness.

This talk gives a comprehensive overview of the associative query languages of several well-known, commercial, C++ based object-oriented DBSs. Main emphasis is put on the functionality offered by the languages. Several characteristic queries are presented to this end. These queries are chosen to precisely determine functional differences between query languages. We show for each query how it is formulated, or why it cannot be expressed. Further aspects concern the embedding, i.e., how query results are represented in the host programming language, and the style of query formulation. The analysis leads to a classification of query languages, precisely differentiating and assessing the query capabilities. The comparison is based upon practical evaluations.

Deductive Object-Oriented Database Systems: From Wishful Thinking to Virtual Reality

Michael Kifer
SUNY at Stony Brook, USA

According to rumors, the early hybrids of object-oriented and deductive languages were mutants that escaped from secret Government AI labs. Whether this is true or not, the fact is that by mid-80's, database and logic programming communities began to take notice. The temptation was hard to resist: the object-oriented paradigm provides a better way of manipulating structured objects, while logic and deduction offer the power and flexibility of ad hoc querying and reasoning. Thus, hybrid languages have the potential for becoming an ideal turf for cultivating the next generation of information systems.

The approaches to integration of the two paradigms range from logic-based languages with unified declarative semantics, to message-passing prologs, to Prolog/C++ cocktails.

In the past eight years, we have been developing a unified object-based logic intended to capture most of the essentials of the object-oriented paradigm. The overall plot here is that once the fundamentals are in place, the actual hybrid programming languages can be carved out of the logic the same way as deductive data languages have been carved out of the classical logic.

There are two distinct aspects in object-orientation: structural (the statics) and procedural (the dynamics). The static aspect was somewhat easier to capture in logic and after taking a few wrong turns F-logic was developed. Taming the dynamics was harder, because there was no widely accepted solution to the problem of state changes even for traditional deductive languages. Our recent work on Transaction Logic appears to provide an adequate framework for specifying the dynamics in classical deductive languages.

This paper first reviews some core aspects of F-logic and Transaction Logic, and then shows how the two can be combined in a natural way to yield a unified foundation for future work on deductive object-oriented languages both in theory and practice. At the end, we discuss areas that still remain to be explored.

Problems with answers to queries against databases with incomplete information

Hans-Joachim Klein
Christian–Albrechts–Universität Kiel, Germany

Related publication: `http://www.informatik.uni-kiel.de/~hjk/sql.sigmod.ps`

The query language SQL handles incomplete information in such a way that answers to queries do not always represent the type of information expected by users. Three–valued logic is used in SQL for the evaluation of *search expressions*; rows qualify for the result of a *where clause* only if the corresponding search expression evaluates to true. In connection with negation and universal quantification this switch from three–valued to Boolean logic is problematic insofar as query transformations using equivalences of three–valued logic may change answers. Another problem is that answers do not represent sure information in general, i.e. there can be possible states to an incomplete database such that the answer is not valid for these states. Uncertain information may be obtained even if three–valued logic plays no role for query evaluation. This is a consequence of the no information interpretation of missing data values in SQL which can be in conflict with the closed world assumption underlying query evaluation. More sophisticated evaluation strategies are necessary to overcome this problem.

We propose different kinds of answers representing sure information to queries against databases with missing values interpreted as *value existent but at present unknown*. Furthermore, an absent value means that exactly one value is missing. With this interpretation, information represented by an answer to a query is considered to be sure if it is valid in every possible state implied by the incomplete database the query is posed against. The restriction to total tuples in answers results in a loss of information and is not suitable for specifying semantics of query languages. Partial tuples have to be allowed in answers. A partial tuple t represents sure information to a query q if for every possible state there is a tuple in the answer to q subsuming t. The set of all partial tuples with this property forms a lower semilattice which is uniquely characterized by its maximal elements. Thus, the set of maximal elements of the semilattice may be taken as a sure answer. In order to get a better approximation of possible answers by sure answers properties such as adequacy, restrictedness, and minimality can be used to define another kind of sure answer. For both kinds of answers the interpretation of missing values in tuples of an answer is different from the interpretation of missing values in the corresponding state insofar as they may represent sets of values of restricted domains. The restriction of answers to sets of tuples having a uniform interpretation of missing values

gives us a third kind of sure answer.

In case of incomplete information completeness of answers is tied up with the tautology problem. Therefore, we have to look for efficient algorithms not losing too much information. For the tuple relational calculus, interpretations using Kleene's three–valued and a new five–valued logic can be given which allow to determine subsets of different kinds of sure answers. For the evaluation of expressions of relational algebra a so–called *switch method* is presented. Using appropriate definitions of sure and maybe versions of operations the equivalence of relational calculus and relational algebra can be shown.

The methods developed for the evaluation of calculus and algebra expressions can be used to derive rules for the transformation of SQL queries such that answers are guaranteed to represent sure information.

Nested Transactions in a Logical Language for Active Rules

Georg Lausen
Universität Freiburg, Germany

We present a hierarchically structured transaction-oriented concept for a rule-based active database system. In previous work, we have proposed *Statelog* as a unified framework for active and deductive rules. Following the need for better structuring capabilities, we introduce *procedures* as a means to group semantically related rules and to encapsulate their behavior. In addition to executing elementary updates, procedures can be called, thereby defining (sub)transactions which may perform complex computations. A Statelog procedure is a set of ECA-style Datalog rules together with an import/export interface. System-immanent frame and procedure rules ensure both propagation of facts and processing of results of committed subtransactions. Thus, Statelog programs specify a nested transaction model which allows a much more structured and natural modeling of complex transactions than previous approaches. Two equivalent semantics for a Statelog program P are given: (i) a logic programming style semantics by a compilation into a logic program, and (ii) a model-theoretic Kripke-style semantics. While (ii) serves as a *conceptual model* of active rule behavior and allows to reason about properties of the specified transactions, (i) – together with the appropriate execution model – yields an operational semantics and can be used as an implementation of P.

Developing DB Languages with Active and Passive Rules: Experiences, Issues, Opinions

Rainer Manthey
University of Bonn, Germany

A number of research projects are currently involved in the investigation of the design and implementation of DB languages supporting some or all of the following forms of DB-centered specifications:

- active rules, or: triggers, defining stereotypical reaction patterns a DBMS is supposed to follow in response to the observation of particular events
- passive/deductive rules, or: views, defining datasets in terms of general laws involving other datasets
- (static) constraints, defining legal states of datasets.

There is an ongoing discussion whether both kinds of rules, active and passive ones (subsuming constraints in the following) should be supported, or if a system ought to focus on either the imperative (active) rules, or on the declarative (passive) ones. We believe that both forms should coexist and interact "freely"!

Passive rules and constraints, though being static expressions, "induce" behaviour, too, namely query- and update-driven inference. Each of the inference modes could as well be coded equivalently in terms of active rules. Thus, one could easily do without the declarative forms without loosing expressiveness. However, passive rules are extremely useful as abstract, high-level representations of pretty complex sets of active rules. Their use during application design thus drastically reduces complexity, if applicable.

Internally, passive rules can be automatically mapped into the corresponding trigger sets by suitable compilers, thus reducing inference to rule triggering and leading to a uniform architecture of a rule-based DBMS. Our group is currently involved in the development of two such languages, one object-oriented (CHIMERA, within the IDEA project), and one relational (PHOENIX).

Circumscribing Datalog: expressive power and complexity

Luigi Palopoli[5]
Università della Calabria, Italy

We study a generalization of Datalog, the language of function-free Definite clauses. It is known that standard Datalog semantics (i.e., least Herbrand model semantics) can be obtained by regarding programs as theories to be circumscribed with all predicates to be minimized. The extension we propose, called Datalog^{CIRC}, consists in considering the general form of circumscription, where some predicates are minimized, some predicates are fixed and some vary. We study the complexity and the expressive power of the language thus obtained. We show that this language (and, actually, its non-recursive fragment) is capable of expressing all the queries in DB-co-NP and, as such, to be much more powerful than standard Datalog, whose expressive power is limited to a strict subset of PTIME queries. Both data and combined complexities of answering Datalog^{CIRC} queries are studied. Data complexity is proved to be co-NP-complete. Combined complexity is shown to be in general hard for co-NEXPTIME and complete for co-NEXPTIME in the case of Herbrand bases containing k distinct constant symbols, where k is bounded. Moreover, the issue of implementing Datalog^{CIRC} queries is addressed. Finally, two PTIME fragments of the language are singled out.

SQL3-Standardisation, a Status Report

Peter Pistor
IBM Scientific CTR, Heidelberg, Germany

Starting from the current SQL Standard ISO/IEC 1995:1992 ("SQL2") a multi-part suite of standards has been evolving over the past years which are intended to cover different requirement profiles. Currently most efforts go into the following parts:

- SQL/Foundation
- SQL/Bindings
- SQL/PSM

[5]Joint work with Marco Cadoli, Dipartimento di Informatica e Sistemistica, Universitá di Roma "La Sapienza", e-mail: cadoli@dis.uniroma1.it

- SQL/Object

Their major offering are:

- Providing features that have been considered severe omissions of SQL2, such as support for recursion and triggers;
- Making SQL a computationally complete language;
- Providing support for object oriented concepts such as abstract data types (ADTs), inheritance, collections, polymorphism.

The other parts address portability (SQL/CLI) and interoperability (SQL/XA, SQL/Abstract Tables), or specific application scenarios (SQL/Temporal).

Based on these efforts, standardized ADT-libraries are being developed in parallel, addressing support for full text search, geographic information systems, and for still image databases.

The presentation will mainly address the object oriented features of SQL3.

Adding ordered data to database systems

Lawrence V. Saxton
University of Regina, Canada

Order occurs naturally in many database applications, especially in textual databases and in scientific databases. Since order is required in many application areas, it seems natural that it should be available intensionally (ie, in the database itself). Databases have been designed so that no query can enter an unreasonably long looping structure. To provide for more general use, a programmer has to write a program that would do the looping. Relational databases and object-oriented databases do not provide order intensionally because of the fear that long processing times could result. Several possible approaches to adding order information intensionally while keeping queries to limited processing time: add list structures to possible data types; add matrix structures to possible data types; add temporal data. We will emphasize lists mainly in overviewing works in progress.

While adding a data type seems reasonably useless without the corresponding query facilities, the time complexity issue above does cause concerns. In forcing extensional query facilities, we control the implicit power guaranteeing polynomial processing time. Object-oriented database systems do have primitive list and matrix types but force one to encapsulate the operators normally.

The choices of language type are variants of SQL, relational algebra, relational calculus and query-by-example. We have studied and implemented a list algebra which includes a 'find' operator to do pattern matching and various list manipulation facilities. We have shown that the list algebra can be efficiently implemented.

We are also designing a listSQL that allows basic list operations and pattern matching facilities. The structure of this extension will be introduced. We have also shown that adding lists and basic list operations doesn't significantly increase the complexity. At present, we haven't looked at approximate pattern matching representations in a language. This is of course effectively achievable, since it is well known that efficient approximate matching algorithms exist.

A fragment of temporal logic as a calculus has been shown to be powerful enough to handle order data. Finally consider a schema definition. It is known that we could store the schema as a context-free grammar. We extend this idea slightly and discuss briefly how it can be used to provide a query-by-example language.

On the Design of Object Database Languages

Marc H. Scholl
Universität Konstanz, Germany

CROQUE project HomePage:
`http://wwwdb.informatik.uni-rostock.de/Research/CROQUE.engl.html`

As a contribution to the discussion during this Dagstuhl workshop this talk emphasized two points that the author feels to be underrepresented in current research activities:

- When designing new database languages for object databases, it is important to consider *update facilities* as well as retrieval operations. In the past, that is, in the relational context, querying databases has been investigated to extreme detail, while updates have mostly been ignored and left to the implementors. It turned out, however, that there are important and challenging problems to be investigated from a principal perspective.

- The support for physical data independence (i.e., the fact that DB application programs are written against a *logical* schema/view of the database, while the actual physical storage structures are transparently managed by the DBMS) is one of the most prominent contributions of DBMSs. During the move from relational to object databases, this outstanding feature has often been sacrificed for reasons that are not really understood. The author feels that data independence goes very well with ODBMS and has been involved in DBMS implementation projects that tried to validate that claim.

In order to support the two claims for important research topics, we gave brief summaries of the work that has been done in the COCOON and CROQUE projects.

A Structured Approach for the Definition of the Semantics of Active Databases

Letizia Tanca[6]
Università di Verona, Italy

Active DBMS's couple database technology with rule-based programming to achieve the capability of reaction to database (and possibly external) stimuli, called *events*. The reactive capabilities of active databases are useful for a wide spectrum of applications, including security, view materialization, integrity checking and enforcement, or heterogeneous databases integration, which makes this technology very promising for the near future.

An active database system consists of a (passive) database and a set of *active rules*; the most popular form of active rule is the so-called *event-condition-action (ECA)* rule, which specifies an action to be executed upon the occurrence of one or more events, provided that a condition holds.

Several active database systems and prototypes have been designed and partially or completely implemented. Unfortunately, they have been designed in a totally independent way, without the support of a common theory dictating the semantics of ECA rules, and thus often show different behaviours for rules with a similar form.

In this work we consider a number of different possible options in the behaviour of an active DBMS, based on a broad analysis of some of the best known implemented systems and prototypes. We encode these options in a user-readable form, called *Extended ECA*. A rule from any existing system can be rewritten in this formalism making all the semantic choices apparent. Then, an EECA rule can be automatically translated into an internal (less readable) format, based on a logical style, which is called *core* format: the execution semantics of core rules is specified as the fixpoint of a simple transformation involving core rules. As an important premise to this research, a semantics for database updates and transactions has also been established, with respect to a notion of state that comprises both data and events.

This work has been published on ACM Transactions on Database Systems, december 1995.

[6]Joint work with Piero Fraternali.

Models and Languages for the World Wide Web

Riccardo Torlone
Universita' di Roma Tre, Roma, Italy

The World Wide Web (WWW) is an hypertext based, distributed information system that provides access to heterogeneous data located over the Internet. Because of the enormous amount of resources available, it is often difficult to retrieve specific information of interest with current browsing tools. Specifically, a support for high-level query facilities is still missing in current systems and would be very useful in this context. We discuss on these issues and report on recent researches, in the database and related areas, that can provide a contribution to the solution of this problem. They include information retrieval techniques, formalisms for extracting information from documents, graph query languages, models and languages for semi-structured data. It turns out that methods and results taken from these studies, combined with specific implementation techniques, can be at the basis of a framework that provides query language facilities for retrieving informations of interest contained in distributed hypertext based environments.

Generalized Quantifiers in Decision Support Queries

Dirk Van Gucht[7]
University of Indiana, Bloomington IN, USA

In this talk we will show how the theory of generalized quantifiers, as it was developed in linguistics for natural language formalization, can be incorporated in a logic-based query language to more naturally express complex queries that occur in decision support [BGVG95]. In addition we will discuss an implementation of this language which significantly outperforms relational query languages on such queries [RBVG96].

References:

[BGVG95] Antonio Badia, Marc Gyssens, and Dirk Van Gucht. "Query Languages with Generalized Quantifiers". In Applications of Logic Databases, R.Ramakrishnan, Ed. Kluwer Academic Publishers, 1995, pp. 235–258

[7] Joint work with Antonio Badia, Marc Gyssens, and Sudhir Rao.

[RBVG96] Sudhir Rao, Antonio Badia, and Dirk Van Gucht. "Providing Better Support for a Class of Decision Support Queries". In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada, pp. 217–227.

Describing Query Execution in Parallel Database Systems

Florian Waas[8]
Humboldt-Universität zu Berlin, Germany

Project HomePage: `http://www.dbis.informatik.hu-berlin.de/~explore/exSPECt.html`

Parallel database systems demand to extend current techniques for performing query optimization and query execution. To master the complexity of query optimization we would like to present query evaluations plans (QEPs) during the different stages of processing (ranging from non-executable specifications to executable programs) in a uniform manners. Additionally, we must describe the different aspects of query execution. We therefore propose initial ideas for designing a database query execution language (DBQEL) that allows us to describe all important features of parallel query execution. Current language approaches to capture parallelism restrict themselves to specific features of program execution rather than to provide means to describe all important aspects.

The features we would like to describe include the types of parallelism available based on existing dependencies among (sequences of) operators, the degree of parallelism, and the kind of execution mode (demand driven or data driven). The design principles of our DBQEL are based on the stream paradigm which is used by Broy (Technical University of Munich) to specify arbitrary parallel systems. We adapt and extend his approach to the requirements of parallel database systems.

Active Information Delivery in a CORBA-based Distributed Information System

Günter von Bültzingsloewen
Germany

Related publications: `http://www.fzi.de/dbs/publications/overview.html`

Many application areas require the integration of heterogeneous information sources into a coherent distributed information system. With such systems, users frequently

[8]Joint work with J.C. Freytag.

need not only be able to access information, but they also have to be notified automatically when new information that is relevant to their work becomes available. For example, in our environmental systems project, a civil servant needs to be informed when a measured air quality parameter exceeds a certain threshhold. Detection of such a critical situation requires subsequent monitoring of several information sources (e.g., a database recording measurement values and a database with limit values). We present an approach for such situation monitoring which is based on CORBA, for the technical integration of heterogeneous systems and on active DBMS-style ECA-rules. In particular, we discuss the issues in the design of an ECA-rule language supporting heterogeneous information sources.

2.2 Workgroup discussions

Working Group 1: "Descriptive Method Languages"

Chair: Gunter Saake
Otto-von-Guericke-Universität - Magdeburg, Germany

Current object-oriented database systems are using programming languages for implementing update methods, for example C++. However, there are strong reasons for using a descriptive method language:

- The concept of data independence requires database languages to be independent from implementation platforms and languages.
- Optimization of methods is desirable but feasible only for restricted languages.
- The problem of integrity checking in the presence of methods is easier to solve if the semantics of methods is known by the DBMS.

The participants of the working group generally agree on the benefits of descriptive method languages for the mentioned topics. During the working group, several approaches were discussed:

- The framework of *transaction logic* was developed as logic-based language for programming database transactions.
- Several specific restricted method languages were proposed in the context of object-oriented specification languages. These concepts can be used for implementation of methods in a descriptive style, too.
- For query methods, the an extended SQL language can serve as an implementation language. The question arises whether SQL could be extended by descriptive update features.
- ECA rules were proposed as an integrity control mechanism for object-oriented languages. This approach could be extended to updates where a method invocation basically serves as an ECA rule event.
- A database state can be interpreted as a theory. An update method has to be interpreted as a theory revision where the revision is restricted to facts.

Working Group 2: "Query Languages"

Chair: Anthony Bonner
University of Toronto, Canada

This group focussed on the capabilities and features needed in query languages for new applications. A number of speakers presented their views, and each was asked to offer a "canonical query" that captured the essence of a problem or the requirements of an application. The following is a selection of the views put forward by the participants.

Jan Van den Bussche: queries that depend on the schema, and queries stored as programs. We discussed the following two example queries:

1. *What is known about John?* Here we want all tuples $(R, A, t(A))$ where R is a relation name in the scheme, A is an attribute of R, and t is a tuple in R of which one of the fields equals John.
2. *Which of the views currently defined in the system would change if we applied update U?*. Here U could be a fixed update or could itself be stored in the database as a stored program (like view definitions).

The point is to express these queries in a way that does not depend on the database schema, although access to the data dictionary and the like is allowed. Query languages that can handle this must have reflective capabilities: they must provide the possibility of retrieving procedures, constructing new procedures (where the construction may depend on ordinary data in the database), and dynamically executing these procedures.

Burkhard Freitag: multi-media tutoring systems. Multi-media tutoring systems require a specification language that supports synchronization and error handling. One would expect, for instance, that the simultaneous display of several images is synchronized even if the retrieval of some data takes longer than anticipated. In general, if an undesired situation arises, the system should be able to go back to a previous safe state. The interesting point is that logical, *i.e.*, high level, synchronization and recovery is required rather than traditional physical recovery. Currently available commercial tutoring systems and authoring tools do not address these issues satisfactorily. Whether these questions affect query languages in the narrow sense may be debatable. However, the example shows that a "transaction programming language" is required for this class of applications.

Hans-Joachim Klein: user friendliness. The first problem is how to measure the complexity of formulating queries in a given language. The challenge is to determine when a query should be called difficult, and to find a suitable mixture of queries for the comparison of different languages. For example, let the two relation schema ORDER(C#,P#) and CAN_SUP(S#,P#) have the familiar customer-supplier-parts semantics, and consider the following query: *Retrieve the identification number of each customer such that some supplier can supply every part ordered by the customer.* It is not difficult to formulate this query in the tuple relational calculus, but it *is* difficult to formulate in relational algebra, while it becomes easy again if set inclusion is available in the language.

Another problem is the readability of query formulations. It is not always easy to feel sure about the correctness of a given formulation. How can we make query languages and query systems more flexible in order to overcome such problems?

Peter Pistor: updatable ADT views (or: How to go from the relations to objects without tears). Consider the following legacy table: EMPLOYEES(NAME, SERIAL, SALARY, DOB). If the database system supports ADT features, one could work with an EMPLOYEE ADT rather than with a "raw" tuple, and thus profit from methods attached to this type. Since it is often not feasible to physically migrate data from an old (legacy) representation to a new representation, an ADT view could be defined over a legacy table (thus leaving existing applications alone).

Herman Balsters & Maurice van Keulen: how do we type object-oriented query results? Formally, *what types can the expressions* $e_1, \ldots, e_n$ *have when they are collected into a set; i.e., the set* $\{e_1, \ldots, e_n\}$*?* When confronted with this question, database researchers usually accept the following two rules:

1. If $e_1 : \sigma_1, \ldots, e_n : \sigma_n$ and $\tau = LUB(\sigma_1, \ldots, \sigma_n)$ exists, then $\{e_1, \ldots, e_n\}$ is correctly typed and has type $P\tau$.
2. $e \in S$ whenever $e : \sigma$, $S : P\tau$ and $\sigma \leq \tau$

Here, LUB denotes the least upper bound w.r.t. the subtyping relation ($\leq$), and P denotes the power type constructor. At first sight, these rules make perfect sense. They also allow the kind of typing flexibility that people like in O-O systems. However, if we agree on these two rules, then we must also accept the following statement:

$$\langle c = 3 \rangle \ \in \ \{\langle a = 3 \rangle, \langle b = 2 \rangle\} \qquad (*)$$

The reason that (*) is type correct is that the LUB of the two record types $\langle a : int \rangle$ and $\langle b : int \rangle$ is the empty record type $\langle\langle\ \rangle\rangle$, and the type of the record expression $\langle c = 3 \rangle$ (being $\langle c : int \rangle$) is a subtype of the empty record type (according to conventional Cardelli-like rules for record subtyping). Now, most

people would agree that there is something wrong here, and that (*) should be rejected. But on what grounds? Rules 1 and 2 seem like such promising candidates for a liberal, flexible, and natural tying discipline for dealing with O-O sets. We claim that this is a serious, non-trivial problem that has to be tackled if there is going to be any sound theory underlying OQL query results.

2.3 Further Contributions

Is Deferred Faster than Immediate? - Benchmarking Schema Updates for Object Database Systems -

Roberto Zicari[9]
Johann Wolfgang Goethe-Universität Frankfurt, Germany

(talk only been announced, not given)

When the schema of an object database system is modified, the database needs to be changed in such a way that the schema and the database remain consistent with each other. Several object database systems offer mechanisms to update the database [FFMMZ95, ObjectStore,OTGen]. Mainly there are two approaches for implementing such database updates: immediate and deferred [FMZ94].

In the first case, all objects of the database are updated immediately as soon as the schema modification is performed, whereas with the deferred approach objects are updated only when they are actually used. This paper focuses on a performance evaluation of the two mentioned approaches and identifies those parameters that influenced most the results.

Schema and application designers will find answers to the following questions in the results of our study:

What difference does it make for an application if schema updates are implemented as immediate or deferred database updates?

Are there any applications that may benefit from knowing which strategy is chosen (or by default implemented) ?

Also, what are the parameters which impact such performance ? Is the size of the database important? Is the frequence of the schema updates important?

Our interest in benchmarking database updates for object database systems is rooted back when the third author was visiting GIP Altair (now O2 Technology) in 1989-1990 [HVZ90].

At that time the initial motivation was to understand how to update the O2 database system as a consequence of a schema update. Alternatives considered were immediate or deferred.

In the meanwhile database updates have been defined and implemented in the O2 commercial system. Both strategies have been implemented as a result of a joint

[9]Joint work with Fabrizio Ferrandina and Thorsten Meyer.

cooperation between O2 Technology and the database group (DBIS) at the University of Frankfurt [FFMMZ95].

References:

[HVZ90] G. Harrus, F. V'elez, and R. Zicari. "Implementing Schema Updates in an Object-Oriented Database System: a Cost Analysis". Technical Report, GIP Altair, 1990.

[ObjectStore] Object Design Inc. ObjectStore User Guide, Release 3.0, chapter 10, December 1993.

[OTGen] Barbara Staudt Lerner and A. Nico Habermann. "Beyond Schema Evolution to Database Reorganization". In Proc. of the ACM Conf. on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA) and Proc. of the European Conf. on Object-Oriented Programming (ECOOP), pages 67-76, Ottawa, Canada, October 21-25, 1990.

[FFMMZ95] Fabrizio Ferrandina, Guy Ferran, Joelle Madec, Thorsten Meyer, and Roberto Zicari. "Schema and Database Evolution in the O2 Object Database System". In Proc. of the 21th Int'l Conf. on Very Large Databases, pages 170-181, Zürich, Switzerland, September 11-15, 1995.

[FMZ94] Fabrizio Ferrandina and Thorsten Meyer and Roberto Zicari. "Implementing Lazy Database Updates for an Object Database System". In Proc. of the 20th Int'l Conf. on Very Large Databases, pages 261-272, Santiago, Chile, September 12-15, 1994.

Hans-Jörg Schek
ETH Zürich, Switzerland

(no talk given)

Group HomePage: `http://www-dbs.inf.ethz.ch/`

3 List of Participants

Addresses as of December 1996.

Malcolm **Atkinson**
University of Glasgow
Department of Computing Science
17 Lilybank Gardens
Glasgow G12 8QQ
Great Britain
mpa@dcs.gla.ac.uk
http://www.dcs.gla.ac.uk/~mpa/
tel.: +44-1-41-330-43 59

Herman **Balsters**
University of Twente
Computer Science Dept.
Postbus 217
NL-7500 AE Enschede
The Netherlands
balsters@cs.utwente.nl
http://wwwis.cs.utwente.nl:8080/
personnel/sigs/sigbalsters.html
tel.: +31-53-4893-772

Anthony J. **Bonner**
University of Toronto
Dept. of Computer Science
10 King's College of Road
Toronto Ontario M5S 3G4
Canada
bonner@db.toronto.edu
http://www.db.toronto.edu:8020/people/
bonner/bonner.html
tel.: +1-416-978-7441

Stefan **Conrad**
Otto-von-Guericke-Universität -
Magdeburg
Institut für Technische Informations-
systeme
Postfach 4120
D-39016 Magdeburg
Germany
conrad@iti.cs.uni-magdeburg.de
http://max.cs.uni-magdeburg.de/
institut/staff/conrad.html
tel.: +49-391-67-1 80 66

Piero **Fraternali**
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32
I-20133 Milano
Italy
fraterna@elet.polimi.it
http://www.elet.polimi.it/people/fraterna/
tel.: +39-2-2399 3651

Burkhard **Freitag**
Universität Passau
Fakultät für Mathematik u. Informatik
94030 Passau
Germany
freitag@fmi.uni-passau.de
http://www.uni-passau.de/fmi/lehrstuehle/
freitag/group/freitag.html
tel.: +49-851 509 3130

Johann Christoph **Freytag**
Humboldt Universität zu Berlin
Institut für Informatik
Unter den Linden 6
D-10099 Berlin
Germany
freytag@informatik.hu-berlin.de
http://poll.informatik.hu-berlin.de/~freytag/
tel.: +49-30-20181-279

Marc **Gyssens**
University of Limburg
Dept. WNI
Universitaire Campus
B-3590 Diepenbeek
Belgium
gyssens@luc.ac.be
http://www.luc.ac.be/research/groups/
theocomp/gyssens.html
tel.: +32-11-268-248

Andreas **Heuer**
Universität Rostock
Lehrstuhl Datenbank- &
Informationssysteme
D-18051 Rostock
Germany
heuer@informatik.uni-rostock.de
http://wwwdb.informatik.uni-rostock.
de/~ah
tel.: +49-381-498-34 01

Uwe **Hohenstein**
Siemens AG - Muenchen
ZFE T SE 4
Zentralbereich Forsch. u. Entwicklung
D-81730 München
Germany
uwe.hohenstein@zfe.siemens.de
tel.: +49-89-636-4 40 11

Alfons **Kemper**
Universität Passau
Fakultät für Mathematik und Informatik
Lehrstuhl Dialogorientierte Systeme
Innstr. 30
D-94030 Passau
Germany
kemper@db.fmi.uni-passau.de
http://dodgers.fmi.uni-passau.de/
~kemper
tel.: +49-851-509-3060

Michael **Kifer**
SUNY at Stony Brook
Computer Science
Stony Brook NY 11794-4400
USA
kifer@cs.sunysb.edu
http://www.cs.sunysb.edu:80/~kifer/
tel.: +1 516 632 8459

Hans-Joachim **Klein**
Universität Kiel
Inst. für Informatik und
Prakt. Mathematik
Hermann-Rodewald-Str. 3
D-24098 Kiel
Germany
hjk@informatik.uni-kiel.de
tel.: +49-431-880-44 63

Bart **Kuijpers**
University of Antwerpen
Dept. of Math. and Computer Science
Universiteitsplein 1
B-2610 Antwerpen
Belgium
bart.kuijpers@uia.ac.be
tel.: +32-3-820-24 17

Georg **Lausen**
Universität Freiburg
Institut für Informatik
Am Flughafen 17
79110 Freiburg
Germany
lausen@informatik.uni-freiburg.de
http://www.informatik.uni-freiburg.de/
~lausen/
tel.: +49-761-203-81 21

Sven-Eric **Lautemann**
Universität Frankfurt
Fachbereich Informatik (20)
Robert-Mayer-Str. 11-15
D-60054 Frankfurt
Germany
lautemann@informatik.uni-frankfurt.de
http://www.dbis.informatik.uni-frankfurt.de/
~lauteman/lautemann_e.html
tel.: +49-69-798-2 84 34

Rainer **Manthey**
Universität Bonn
Institut für Informatik III
Römerstr. 164
D- 53117 Bonn
Germany
manthey@informatik.uni-bonn.de
http://www.informatik.uni-bonn.de/~manthey/
tel.: +49-228 550 292 (Sekr.)

Luigi **Palopoli**
Universita' della Calabria
Dip. di Sistemi
I-87036 Localita' Arcavacata Rende (CS)
Italy
palopoli@si.deis.unical.it
tel.: +39-984-494-7 52

Peter **Pistor**
IBM Scientific CTR
Vangerowstr. 18
D-69115 Heidelberg
Germany
ppistor@vnet.ibm.com
tel.: +49-6221-59 42 92

Gunter **Saake**
Otto-von-Guericke-Universität - Magdeburg
Institut für Technische Informationssysteme
Postfach 4120
D-39016 Magdeburg
Germany
saake@iti.cs.uni-magdeburg.de
http://max.cs.uni-magdeburg.de/institut/staff/saake.html
tel.: +49-391-671-88 00

Lawrence V. **Saxton**
University of Regina
Dept. of Computer Science
Regina Saskatchewan S4S 0A2
Canada
saxton@cs.uregina.ca
http://www.cs.uregina.ca/~saxton/
tel.: +1-306-585-46 32

Hans-Jörg **Schek**
ETH Zürich
Institut für Informationssysteme
ETH-Zentrum
CH-8092 Zürich
Switzerland
schek@inf.ethz.ch
http://www-dbs.inf.ethz.ch/
tel.: +41 -1 632 -7240

Marc H. **Scholl**
Universität Konstanz
Fakultät für Mathematik u. Informatik
Postfach 55 60/D 188
78434 Konstanz
Germany
marc.scholl@uni-konstanz.de
http://www.informatik.uni-konstanz.de/~scholl
tel.: +49-7531-88-44 32

Marc **Smits**
University of Antwerp/UIA
Dept. of Mathematics & Computer Science
Universiteitsplein 1
B-2610 Antwerp
Belgium
msmits@uia.ac.be
tel.: +32-3-820-24 17

Letizia **Tanca**
Universita di Verona
Facolta di Scienze MM. FF. e NN.
Strada Le Grazia
I-37134 Verona
Italy
tanca@elet.polimi.it
http://www.elet.polimi.it/people/tanca/
tel.: +39-2-23-99-35 51

Riccardo **Torlone**
Universita' di Roma Tre
Dip. di Informatica e Automazione
Via della Vasca Navale, 84
I-00146 Roma
Italy
torlone@iasi.rm.cnr.it
http://www.inf.uniroma3.it/people/torlone.html
tel.: +39-6-55177053

Dirk **Van Gucht**
University of Indiana
Computer Science Department
215 Lindley Hall
Bloomington IN 47405-4101
USA
vgucht@cs.indiana.edu

Florian **Waas**
Humboldt Universität zu Berlin
Institut für Informatik
Unter den Linden 6
D-10099 Berlin
Germany
waas@dbis.informatik.hu-berlin.de
http://www.dbis.informatik.hu-berlin.de/~waas
tel.: +49-30-20181-331

Maurice **van Keulen**
University of Twente
Computer Science Dept.
INF 3034
Postbus 217
NL-7500 AE Enschede
The Netherlands
keulen@cs.utwente.nl
http://wwwis.cs.utwente.nl:8080/personnel/sigs/sigkeulen.html
tel.: +31-53-489-37 25

Jan **van den Bussche**
University of Antwerp/UIA
Dept. of Mathematics & Computer Science
Universiteitsplein 1
B-2610 Wilrijk/Antwerp
Belgium
vdbuss@uia.ac.be
http://win-www.uia.ac.be/u/vdbuss/
tel.: +32-3-820-2417

Günter **von Bültzingsloewen**
Neumarkterstr. 8a
D-79618 Rheinfelden
Germany
guenter.von_bueltzingsloewen@mhs.swissbank.com
Office:
Swiss Bank Corporation
IT Architectures and Standards
Hochstr. 16
CH-4002 Basel
Switzerland