more complex structures and their behaviour.

A Design Environment for Migrating Relational to Object Oriented Database Systems

by Jens Jahnke, Wilhelm Schäfer, and Albert Zündorf, University Paderborn

Abstract:
Object-oriented technology has become mature enough to satisfy many new requirements coming from areas like computer-aided design (CAD), computer-integrated manufacturing (CIM), or software engineering (SE). However, a competetive information management infrastructure often demands to merge data from CAD-, CIM-, or SE-systems with business data stored in a relational system. In addition, complex dependencies between those data stored in the different systems might exist and should be maintained. One approach for seamless integration of object-oriented and relational systems is to migrate the data (and the corresponding schema) from a relational to an object-oriented system. In this paper we describe an integrated design environment that supports the migration process and overcomes major drawbacks of comparable approaches.

digms. Furthermore, fundamental concepts such as revisions, variants, configurations, and changes are defined and related to each other. In particular, we focus on intensional versioning, i.e., construction of versions based on version rules. Finally, we provide an overview of systems which have had significant impact on the development of the SCM discipline, and classify them according to a detailed taxonomy.

Polylingual Persistent Object Systems

by Jack C. Wileden, University of Massachusetts, Amherst

Abstract:
Our goal is seamlessness in software systems. Extending a system, or constructing a system from diverse components, tends to introduce seams. The distinct pointer types for persistent and transient objects in the C++ binding of the ODMG persistence model, or the distinct type models for shared (IDL) and local (native language) objects in CORBA-style interoperability, are notable examples of seams. Our work on supporting seamlessness is currently focused on two projects. The JavaSPIN prototype of a persistence extension for Java provides a seamless integration of persistence with the Java programming language. The PolySPIN approach, and the-PolySPINner tool implementing it, supports seamless interoperability among C++, CLOS and Java software modules. Using PolySPIN, an application in one of these languages can transparently access persistent objects defined in any or all of the languages. From the application's perspective, all the objects appear to be defined in the application's language. Seams found in other approaches, such as non-native (e.g., IDL) data models, explicit cross-language function calls or wrappers are not found in PolySPIN. Neither existing applications nor existing data objects need be modified when PolySPIN is used for interoperating them. Hence, PolySPIN offers a basis for polylingual persistent object systems, where we use the term "polylingual" to suggest that language differences can be ignored, analogously to the way in which "polymorphic" indicates that type differences can be ignored.

Generic Definitions of Relationships in Object-Oriented Databases

by Wolfgang Wilkes, University of Hagen, Germany

Abstract:
Relationships between entities are a very important part of each conceptual model. The participation of an object in a relationship influences the behaviour of that object: It needs the ability to interact with related objects. In the classic object oriented models and languages, relationships and their semantics are hidden behind attributes and specific object methods. In our model, an explicit relationship concept is provided which comprises relevant methods for objects participating in the relationship. The basis of our model is an object model which allows objects to change their types during their lifetimes. By defining a relationship, the participating entity types get new subtypes which contain the methods for interacting with related objects. An object is related to another object by acquiring such a subtype. Based on this relationship concept, generic and composite relationships are introduced. By specifying a generic relationship, the operational semantics of a class of relationships is described. The generic relationship can be instantiated as a relationship between specific entities. By instantiating relationships in the context of another generic relationship definition, relationship patterns are built which represent

Broker/Services: Reactive components for process-oriented environments

by Dimitrios Tombros, University of Zurich,

Abstract:
Cooperative process-oriented environments (CPE) such as workflow management systems (WFMS) or process-oriented software engineering environments are highly complex distributed systems consisting of heterogeneous participating software components and human users. The participants cooperate under the restrictions and rules described in predefined processes. In our talk, we described a software architecture model of structural, behavioral and functional aspects of CPE. The BROKER/SERVICES MODEL allows the description of the various aspects of heterogeneous participating components in a uniform way. It provides a service-abstraction above the concrete functionality provided by individual components in the CPE. Furthermore, it is amenable to a formal description of processes executed in a distributed environment, which can be used for the definition of the exact process semantics and thus for proof of correct process execution. The components described in the BROKER/SERVICES MODEL can be integrated and can execute cooperative processes through the use of the event-based execution platform EvE we have developed at the University of Zurich.

Relational structures in generic software development environments

by Jim Welsh, University of Queensland, Australia

Abstract:
Relational structures play a variety of roles in software development. Syntactic structure, as exhibited by hierarchic sequential languages, is well understood and well-supported by generic tool-building technology, using multiple interaction paradigms. Other forms of relational structure are discernible, in process modelling and enactment, in formal methods support, in diagrammatic modelling, in documentation and traceability, and in software comprehension. Generic support for these relational structures is less well-developed, and often specific to one of these areas. We review the requirements for generic support in each of these areas, and outline UQ*, a prototype environment that aims to meet a range of these requirements within a flexible interaction paradigm.

Version Models for Software Configuration Management

by Bernhard Westfechtel, RWTH Aachen,

Abstract:
After more than 20 years of research and practice in software configuration management (SCM), constructing consistent configurations of versioned software products still remains a challenge. In [1], we focus on the version models underlying both commercial systems and research prototypes. We provide an overview and classification of different versioning para-

(CIM), or software engineering (SE). However, a competitive information management infrastructure often demands to merge data from CAD-, CIM-, or SE-systems with business data stored in a relational system. One approach for seamless integration of object-oriented and relational systems is to migrate from a relational to an object-oriented system. The first step in this migration process is reverse engineering of the legacy database. In this talk we propose a new graphical and executable language called Generic Fuzzy Reasoning Nets for modelling and applying reverse engineering knowledge. In particular, this language enables to define and analyse fuzzy knowledge which is usually all what is available when an existing database schema has to be reverse engineered into an object-oriented one. The analysis process is based on executing a fuzzy petri net which is parameterized with the fuzzy knowledge about a concrete database application.

An Assessment of Non-Standard DBMSs for CASE Environments

by Udo Kelter, Universität - GH Siegen,

Abstract:

Many new non-standard database management systems (NDBMSs) and data models have been proposed with the promise to facilitate the construction of better engineering environments and tools and to solve integration problems in environments. However, there is hardly any evidence or experience to what extent these goals are actually met. This paper summarizes experience gained in several major experiments in which different classes of tools in CASE environments (graphical editors, consistency checkers, transformators) have been built using several design approaches and architectures. It turns out that, regrettably, most NDBMSs proposed so far have quite substantial weaknesses and that their overall value for tool designers is fairly modest. The paper first shows that advantages are only possible if the tool architecture is ``redundancy-free" and if the tools operate directly on the database. Assuming this ``DB-oriented" architecture, we examine typical features of an NDBMS (schema management, integrity controls, transactions etc.) with respect to their usefulness in tool implementation. We discuss the resulting requirements on the performance and on the design of the API and the runtime kernel of the NDBMS. We also point out useful new features which do not seem to exist in any NDBMS so far.

Analysing and Integrating Specialization Hierarchies --

Applying Techniques of Formal Concept Analysis for Database Design

by Ingo Schmitt,University Magdeburg,

Abstract:

Integrating inheritance hierarchies is an essential task of schema integration as part of the design of a federated database. Based on the formalization of inheritance hierarchies as concept lattices an efficient algorithm for deriving an integrated inheritance hierarchy was presented as result of merging two inheritance hierarchies with overlapping extensions and types. Several methods can be applied to modify the integrated hierarchy to optimize the resulting object-oriented schema with respect to certain quality criteria(disjoint specialization, number of classes, support of null values, etc). These methods are necessary to adjust the integrated schema to specific requirements and for realizing application-specific views.

two phase methodology. The first one, called Data Structure Extraction, is devoted to recovering the technical description of the data structures, i.e. their logical schema. It proposes a set of extensible techniques to elicit hidden constructs and constraints from various information sources, including program source code. The second phase, Data Structure Conceptualization, tries to make the underlying semantics explicit by discarding technical constructs and restructuring optimization oriented structures. It yields a tentative conceptual schema that still has to be normalized. The approach is supported by the DB-MAIN CASE environment. Based on a programmable toolbox architecture, it offers developers powefull analytical tools, sophisticated evaluators and transformation operators. It also includes a collection of powerful assistants which make available many techniques and heuristics dedicated to data structure engineering, and more specifically reverse engineering. Moreover, its meta level allows its users to develop new concepts and new functions according to the specific needs of each reverse enginering project.

<u>**Selected Patterns for Software Configuration Management**</u>

by James Hunt, University Karlsruhe

Abstract:

The discipline of Software Configuration Management (SCM) has advanced tremendously over the past two decades. There have been many informative books and articles dedicated to the subject and a thriving industry has sprouted forth. Yet, much of the knowledge required to build good software managment systems has not been adequately documented. Though the commercialization of SCM may contribute to this deficiency, the lack of well defined methodologies for describing software techniques has been more decisive. Though algorithmic knowledge is relatively easy to describe, most of the expertise in SCM tools is more structure and relationship oriented. Until recently, ad hoc descriptions of structural and relationship techniques existed. The Concept of Design Patterns has changed this. Design Patterns provide a means of documenting and cataloguing software techniques beyond the algorithmic level. There are many aspects of SCM tools that could be formulated as patterns. Some notions can be formulated upon existing patterns. For instance, the concept of a project can be modeled using the Compositepattern. Others have not demonstrated their general applicability. One would like to focus first on the patterns that have reoccured most often in SCM systems and tools. To this end, five patterns have been selected for description: Revision History, Directed Delta, Interleaved Delta, And-Or and Selection Graph, Three Way Merge, and Product Derivation. The patterns are described after the fashion of Messrs Gamma, Helm, Johnson, and Vlissies.

<u>**Generic Fuzzy Reasoning Nets as a Basis for Reverse Engineering Relational Database Applications**</u>

by Jens-H. Jahnke, University Paderborn,

Abstract:

Object-oriented technology has become mature enough to satisfy many new requirements coming from areas like computer-aided design (CAD), computer-integrated manufacturing

phProject and GENESIS, generators for graphical and textual tools. The framework was validated through a trial constructing a customised PSEE for British Airways. Most of the database enhancements developed by GOODSTEP have already been included in the O2 product. The GOODSTEP approach to PSEE construction will be exploited for the improvement of DOORS, a commercial requirements engineering environment.

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

by Gregor Engels, Leiden University
joint work with
Hartmut Ehrig, Reiko Heckel, Gabi Taentzer, Technical University of Berlin

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

Database Reverse Engineering

by Jean-Luc Hainaut, Jean Henrard, University of Namur

Abstract:
Reverse Engineering is the process through which one tries to recover the specifications of a software component. Database Reverse Engineering addresses the problem of rebuilding the logical and conceptual schemas of a set of files or of a database. The problem usually proves much more complex than expected, mainly due to the weaknesses of the technical data models and to programming practices that tend to bury structures and integrity constraints into the application programs. Beyond the mere analysis of the declarative statements that define the data structures, it is necessary to analyse complementary information sources such as the procedural code, the data, screens and reports, etc. In addition, the data seldom are purely semantical constructs. Most often they include technical and optimization aspects that make their meaning more or less obscure. The DB-MAIN approach to database reverse engineering is based on a

-for each measure, a list of transformation rules resulting in improvements of the measure is given.
During our work with MeTHOOD we have made the following observations:
It is possible to make scattered object-oriented design knowledge tangible for designers in an effective way. It seems to be impossible to describe schema quality as a whole (this observation is also supported by others [15-17]). There are two possible ways to describe a non-trivial aspect of schema quality. With non-trivial we mean that the set of "high quality schemas" are not restricted to a single possible structure or a hierarchy of structures like in algorithmic based relational database design [18,19]. One way is to measure internal properties of schema elements (e.g. the ratio between public and private attributes) assessing aspects of internal product quality. Another way is to measure conflicts to a set of heuristics (heuristic quality).It is possible to recognise all locations in all schemas that can be changed to improve heuristic quality for a specific set of heuristics. It is possible to recognise some locations in some schema that can be changed to improve the internal product quality. Some heuristics are related to some measures: if conflicts to these heuristics are removed measured values are improved. For some detected locations it is possible to generate alternative improved schemas. Location recognition and generation of alternative schemas can be automated. Some characteristics of applications have an influence on the relevance of heuristics and measurements. E.g., the lifetime of some applications in the area of security trading are very short. Heuristics and measurements considering aspects of maintenance are unimportant.

GRAL: Z-like description of integrity constraints

by Juergen Ebert,University Koblenz-Landau

Abstract:
TGraphs, i.e. typed attributed and ordered directed graphs are used as the internal data structure in a variety of software engineering tools (e.g. in the metacase tool KOGGE and the program understanding tool GUPRO).Classes of TGraphs can be described by extended entity-relationship (EER) diagrams together with additional GRAL conditions. GRAL is a subset of Z extended by regular path descriptions which are used as operators and/or predicates. GRAL predicates can be checked in polynomial time on a given graph. An EER/GRAL specification can be translated to a complete Z specification of a class of TGraph thus giving a formalbasis to the description.

The GOODSTEP Project

by Wolfgang Emmerich, City University London

Abstract:
The goal of the GOODSTEP project was to enhance and improve the functionality of an object database management system (ODBMS) to yield a platform suited to the construction of process-centred software engineering environments (PSEEs). The baseline of the project was the O2 ODBMS. The project enhanced O2 in order to make it a real software engineering database. These enhancements were exploited and validated by the construction of the GOODSTEP framework for PSEE construction, which includes the SPADE software process toolset and Gra-

*************************Technical Contributions****************************

Active Database Management Systems

by Andreas Geppert, University of Zürich

Abstract:
Typical database management systems (DBMSs) are passive in the sense that they perform actions only as a response to explicit requests. In contrast, active DBMSs (ADBMSs) are in addition to the usual capabilities of a DBMS able to react to predefined situations occurring in the database or its environment. This kind of reactive behaviour is typically specified in the form of event-condition-action rules (ECA-rules), with the following semantics:
- when the event occurs and
- the condition holds
- then execute the action.

Object-Oriented Design Heuristics

by Thomas Grotehen, Klaus R. Dittrich,University of Zürich

Abstract:
The MeTHOOD (1) approach makes design knowledge tangible that will help to improve conceptual object-oriented schemas. Although a large amount of this important knowledge is available in current literature, for example [1-4], it is nearly unusable for designers. The objective of the MeTHOOD approach therefore is to enable a design process allowing designers to inspect and improve a conceptual class schema continuously. It is not necessary to reinvent a complete design methodology (including object model, high level process and notation) to meet this objective successfully. Purely based on well-known object modelling languages [5,6], MeTHOOD provides an extension to existing methodologies. The core of MeTHOOD is a catalogue of "design knowledge" consisting of existing and new object-oriented design heuristics, transformation rules and measures. A design heuristic [7], e.g. "Keep related data and behaviour in one place.", is a guideline or rule of thumb representing design experience and best practice. It is an advice for making design decisions. A design heuristic is used to describe a family of potential design flaws (e.g. locations in a schema where "related data and behaviour are separated") and help designers to detect them. A measure [8] is a mapping between an entity and a number and to characterise a feature of the entity (e.g. the number of its dependencies to other entities). MeTHOOD measures are product measures. They are used to give designers continuous instant feedback on design decisions. Measured values of schema properties make improvements resulting from schema changes more transparent for the designer. A transformation rule is an advice on how to eliminate occurrences of a design flaw. Design heuristics, measures, and transformation rules are more or less known concepts. The core idea of the MeTHOOD approach is a formalisation and an integration of heuristics, measures, and transformation rules. The integration makes implicit relationships between these concepts visible and benefits from them. The integration is described in the MeTHOOD integration schema and instanciated in the MeTHOOD catalogue. This catalogue contains the description of 8 measures, 21 heuristics and 40 transformation rules. Furthermore, -for each measure, a list of heuristics detecting locations where the measure can be improved is given,
-for each heuristic, a list of transformation rules describing ways how a violation of the heuristic can be removed is given, and

nagement. This tool kit enables a database implementor or applications designer to assemble application-specific transaction managers. Each such transaction manager is meant to provide a number of individualised, application-specific transaction types which, hopefully, satisfy the application specific demands in a rather adequate way.

Versions in CAD/CAM Databases

by Wolfgang Wilkes, University of Hagen

Abstract:
One of the main problems in versioning is to describe which sets of versions are consistent. This is addressed differently in various areas. Temporal databases, for instance, use time to define the context for valid version sets, other models propose to use the results of transactions as consistent version sets. In CAD/CAM databases, this question is closely related to the various relationships in which an object is involved: revisions, views orrepresentations, composition hierarchies, status, and variants [1,2]. In particular, by combining these different relationships, various choices have to be done to adopt the versioning strategy to the needs of the specific field of application. In this "mini-tutorial", several examples of combinations have been presented to illustrate how these relationships or "versioning dimensions"influence each other.

An Event Engine (EvE) for Event-Driven Workflow Enactment

by Andreas Geppert, University of Zürich

Abstract:
In the mini tutorial, we introduced the basic features of ECA-rule models (kinds of event types, forms of condition specification, and operations allowed in actions). We also presented aspects of execution models, which define the semantics of event detection and rule execution. Possible applications of ECA-rules, some of which might be relevant for software engineering environments, have also been discussed: consistency constraint enforcement, notification, management of materialized views /update propagation, access control, and certain aspects of workflow management. In a second talk, we discussed the use of ECA-rules for event-driven workflow management as supported by the Event Engine (EvE). EvE supports the control of reactive components in distributed environments (see the +talk by D. Tombros). EvE and the reactive components interact with each other by generating and reacting to events. Thus, a workflow is executed in that reactive components generate events (e.g., in order to request the execution of a certain activity, or to signal the result of an activity execution) and other components reacting to this event according to the workflow specification. EvE has a multi-server architecture, whereby each server is responsible for event detection and event history management, as well as for interaction with its local reactive components and other EvE-servers in the system.

the goal of preventing the chaos that quickly arises when software engineers carry out multiple, simultaneous, and uncoordinated changes in a project. SCM seeks to keep an evolving system in a defined state and regulates and tracks changes. At the base of SCM systems is typically a version management component that stores multiple versions of atomic software objects and configurations. For conserving space, versions are usually encoded as deltas relative to a reference version. The checkin-checkout model is a widely used access protocol. Checkout retrieves a copy of a version for modification, while checkin freezes the copy after modifications have been completed. Three different models are in use for dealing with configurations: the composition model, the change set model, and the long transaction model. In the composition model, configurations are hierarchically organized parts lists. Configurations may contain elements that refer to one or more groups of versions rather than individual objects. In this case, a configuration is called "generic" (because it represents a set of configurations) or "unbound". When a non-generic or "bound" configurations is desired (e.g. for inspection, compiling, execution, or delivery), a set of selection rules are applied to reduce the generic configuration elements to singleton software objects (which are either atomic or subconfigurations). The change set model is the dual to the composition model, showing the updates that lead from a reference configuration forward. Rather than forcing the user to explicitly compose a desired configuration, the change set model derives the configuration by applying selected updates to the baseline. The long transaction model is the checkin-checkout model applied toentire configurations. All changes to a configuration are carried out in a transaction; subtransactions are used for exploratory development. This model provides an easy mechanism for undoing changes, but the complete isolation of change activities caused by transactions causes difficulties in practice. While the basic techniques for version management and handling of configurations are well understood, further research is needed in the following areas: distributed SCM (especially via web technology), incremental software distribution (via networks or otherwise), dynamic loading and reconfiguration, automatic composition of configurations, smart merging of competing changes, and process support. SCM is a well-established discipline within software engineering, providing recognized benefits for the practicioner. From beginnings in the 1970s it has grown into a respectable industry with a number of commercial SCM systems available today.

<u>Advanced Transaction Management: Motivation, Concepts and Models</u>

by Rainer Unland, University of Essen,

Abstract:
Advanced database applications, such as CAD/CAM, CASE, large AI applications or image and voice processing, place demands on transaction management which differ substantially from those in traditional database applications. In particular, there is a need to support enriched data models (which include, for example, complex objects or version and configuration management), synergistic cooperative work, and application- or user-supported consistency. Unfortunately, the demands are not only sophisticated but also diversified, which means that different application areas might even place contradictory demands on transaction management. In my mini-tutorial I first introduced the basic terminology of conventional transaction management and then discussed why conventional transaction management fails to adequately support so called non-standard applications. Therefore, more appropriate concepts have to be developed and applied. However, due to the broad spectrum of demands it seems to be unrealistic to assume that a transaction manager can be developed that is able to satisfy all these demands in an appropriate way. Therefore, we introduced a flexible and adaptable tool kit for transaction ma-

**************************Mini-Tutorials***

Transaction Models for Software Engineering Database

by Reidar Conradi, Jens-Otto Larsen, Nguyen Ngoc Minh, Alf Inge Wang, Norwegian University of Science and Technology (NTNU), Trondheim, Norway and Chunnian Liu, Beijing Polytechnic University, P.R. China

Abstract:
Design work, such as software engineering, poses more advanced needs for transaction support than commercial applications.That is, ACID transactions are no long sufficient. Many never transaction models for long, nested and cooperating transactions have been presented, but few implemented and tried out. The "ideal" transaction model is first presented, giving basic support for flexible data sharing. A more pragmatic transaction model in the EPOS PSEE is then presented, being implemented in the versioned EPOSDB. Process support can then be implemented on top of the transaction support.Some experiences and open questions are finally reported.

Object-Oriented Modelling

by Gregor Engels, Leiden University

Abstract:
Object-oriented modelling approaches follow the idea that structure and behaviour of a system should be modelled in a closely coupled, integrated way. This idea became very popular in the beginning of the '90s when a lot of (commercial) object-oriented methods were published. Prominent examples are Object-Oriented Analysis (OOA) by Grady Booch, Object Modeling Technique (OMT) by Jim Rumbaugh, Object-Oriented Software Engineering (OOSE, Objectory) by Ivar Jabobson, or Object-Oriented Design (OOD) by Coad-Yourdon. In 1994, a joint effort at Rationale started to design a new language which unifies existing languages into one. The team, headed by Grady Booch, Jim Rumbaugh, and Ivar Jacobson published in 1996 the Unified Modelling Language (UML). A language description, published in January 1997, has been submitted to OMG (Object Management Group) to be accepted as standard for object-oriented modelling. The mini-tutorial gives an overview on this development and sketches briefly the components of UML. These are several types of diagrammatic notations to model structure and behaviour of a system. The talk closes with a critical discussion on the semantics of UML. In particular, the extensibility of the meta model description by so-called stereotypes is presented and discussed. More information on UML can be found at http://www.rational.com.

Software Configuration Management

by Walter F. Tichy, University of Karlsruhe,

Abstract:
Software Configuration Management (SCM) controls the evolution of software systems with

Software Engineering and Database Technology

Naser S. Barghouti, Bear Steams & Co., New York
Klaus Dittrich, University of Zürich, Zürich
David Maier, Oregon Graduate Institute,Oregon
Wilhelm Schäfer, University of Paderborn, Paderborn

Although software engineering and database technology are two distinct areas of computer science, they nevertheless intersect in several respects, and each of them has "interfaces" to the other. Combined efforts of both areas thus have a considerable potential for synergy and are expected to be rewarding for both communities. Database researchers and software engineers can benefit from each other in several ways. Both are "clients" or users of the other technology: When considering software engineering environments, software engineers need data stores for managing potentially large repositories of software artifacts, they need appropriate transaction models for controlling the cooperation of designers, etc. Database researchers, on the other hand, construct software systems such as DBMSs and DBMS-based application systems, and would like to develop them using an engineering approach and with powerful tools. Both database researchers and software engineers are concerned with the software architecture of large, complex systems, of which database systems are both an example themselves and often components in larger systems. Furthermore, both communities have recently started to investigate sets of concepts which - even if named differently in the two areas - are strongly related. An example of such related areas are object-oriented methodologies for specification, design and implementation on the software engineering side and object-oriented data models and database systems on the data management side.

This Dagstuhl workshop brought together researchers from both fields who not only presented their own work but also challenged "the other side" to relate its requirements to the other field.

In order to get those discussions off to a quick start, we used a slightly different format than other workshops of this kind. The first 1 1/2 days were devoted to tutorial-like introductions on particular topics. Those tutorials given by key researchers were particularly targeted towards the "other" community to update them on recent developments in the researcher's field. Truly in the nature of the workshop, those tutorials ended in lively discussions especially when presentations on similar topics came from different camps, such as versions, configuration management and nonstandard transactions. The workshop was complemented by a number of demonstrations. The week continued with short presentations of particular research projects, working group meetings targeted at bringing participants with related interests from the two camps together, and open discussion sections with all the attendees. Briefly, some key issues raised during the week were: (1) Where is the borderline between a database system and the application, i.e., whatshould be "presupplied" and what has to be custom-made? (2) Will code-writing disappear, i.e., might application development become purely a matter of parameterising a database system? (3) How far can we go with existing re-engineering tools (though some people were impressed by the sophistication of current tools)? (4) Is the UML (Unified Modeling Language) a suitable database design language? Many other more particular issues arose as well.

Finally, the question was raised whether success stories about fruitful cooperation between the two camps exist. Although the total number might not be great, a few were reported, including the development of (new) object-oriented design languages (and data models), the introduction of versioning into the O2 database by an ESPRIT-funded project, and the development of ObjectStore as a CAD-database system