DECEMBER 1990, VOLUME 13, NO. 4

a quarterly bulletin of the
IEEE Computer Society
technical committee on

Data Engineering

CONTENTS

IEEE

SPECIAL ISSUE ON DIRECTIONS FOR FUTURE DBMS RESEARCH AND DEVELOPMENT

DECEMBER 1990, VOLUME 13, NO. 4

Editor–in–Chief, Data Engineering
Dr. Won Kim
UNISQL, Inc.
9390 Research Boulevard
Austin, TX 78759
(512) 343–7297

Associate Editors
Dr. Rakesh Agrawal
IBM Almaden Research Center
650 Harry Road
San Jose, Calif. 95120
(408) 927–1734

Prof. Ahmed Elmagarmid
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907
(317) 494–1998

Prof. Yannis Ioannidis
Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin 53706
(608) 263–7764

Dr. Kyu–Young Whang
Department of Computer Science
KAIST
P. O. Box 150
Chung–Ryang, Seoul, Korea and
IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

Chairperson, TC
Prof. John Carlis
Dept. of Computer Science
University of Mennesota
Minneapolis, MN 55455

Past Chairperson, TC
Prof Larry Kerschberg
Dept. of Information Systems and Systems Engineering
George Mason University
4400 University Drive
Fairfax, VA 22030
(703) 764–6192

Distribution
IEEE Computer Society
1730 Massachusetts Ave.
Washington, D.C. 20036–1903
(202) 371–1012

Data Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Data Engineering. Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Data Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in the Data Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non–member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both full members and participating members of the TC are entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Letter from the Issue Editor

This is a special joint issue of *ACM SIGMOD RECORD* and *IEEE Data Engineering Bulletin* on **Directions for Future Database Research and Development**. In the past some senior researchers, singly and collectively, attempted to provide directions for future database research in the form of so–called "manifestos" and reports of workshops sponsored by a few organizations. Further, each issue of the *IEEE Data Engineering Bulletin* during the past decade has focused on a single topic of "current" database research topics, thereby providing directions for near–term research. My objective in organizing this special joint issue is to bring these recent disparate efforts to a grand culmination with a compendium of the thoughts of some of the leading authorities in each of a dozen or so areas of database research and development that many believe to be the most important.

I have held the belief for some time that, as the report of the recent Lagunita workshop concludes also, the primary challenges facing database professionals today are to bring about the transition from the current relational database systems to the next generation of database systems in order to dramatically expand the scope of applicability of database technology to beyond the conventional transaction–oriented business data processing; to allow interoperability of a wide variety of data sources managed by existing (and emerging) database systems and file systems; and to evolve the architectures of database systems in keeping with the rapidly unfolding revolutions in hardware technology. The first challenge requires significant additional research in various subdisciplines within databases, including object–oriented databases, extensible databases, spatial databases, deductive databases, temporal databases, imprecise databases, real–time databases, scientific databases, database security, data dredging, database programming languages, and nonprogrammer's application development environments. The second area of challenge is global or multidatabase systems. The third area of challenge is parallel database systems and distributed database systems.

I selected the above areas of research, and then for each area invited a few of the leading authorities to provide a report on the current status of the area and directions for further research and development. For six of these areas, I invited one expert from academia and one expert from industry to jointly author the report to achieve a balanced view of the area. For database programming languages, I elected to include a condensed report of a recent workshop on the subject co–sponsored by the National Science Foundation (NSF) and INRIA. For heterogeneous databases and scientific databases, I invited and edited the reports of workshops on the subjects that NSF sponsored during the past year. I also invited the report of the recent Lagunita workshop, also sponsored by NSF, as the extended introduction to the joint issue. I failed to find authors willing to do a report on friendly user interfaces. (I should add that Maria Zemankova should receive at least a figurative bouquet of flowers and a case of champagne from the database community for her efforts in securing NSF funds to sponsor these three timely workshops.)

I elected to include some of the reports in both *SIGMOD RECORD* and *Data Engineering*, but others in only one of the two publications. This is to accommodate the 64–page space limit in *Data Engineering*, and to avoid repeating in the same publication any topic which was the subject of a special issue within the past few years. Further, ACM SIGMOD and IEEE TC on Data Engineering both charge membership fees to partially offset the cost of publishing and distributing their respective newsletters, and I believe that members of these professional societies would not care to see a completely overlapping joint issue. I included in both publications reports on the topics that in my view have potentially the most impact on the future of database technology, even if they may have been subjects of special issues in the recent past: the reports on imprecise databases and database security are cases in point. The following reports are included in both publications: heterogeneous databases workshop report, object–oriented databases, spatial databases, database security, and distributed databases. The following are included only in the *SIGMOD RECORD*: Lagunita workshop report, scientific databases workshop report, database programming languages workshop report, extensible databases, deductive databases, temporal databases, and parallel databases. The following reports are included only in *Data Engineering*: real–time databases, data dredging, and on the future DBMS architecture.

The contributors to this issue are all leading authorities in the areas they report on (I hope my including myself as a co–author of one of the reports will not disturb anyone's sensibilities), making this joint issue a true "all–star" issue. It certainly was a privilege and pleasure to have worked with these outstanding professionals; all in earnest efforts and considerable time on their reports, and (more or less) met all my deadlines. I hope that this special issue will have a significant and lasting influence on the course of our field.

I request that those interested in obtaining copies of this joint issue to contact directly ACM Headquarters in New York City and IEEE Computer Societyin Washington, D.C.: the demands of my job makes it impossible for me to be responsible for distributing complimentary copies of this issue to anyone who calls.

Won Kim

UniSQL, Inc.
Austin, Texas
October 25, 1990

Report on the Workshop on Heterogenous Database Systems held at Northwestern University Evanston, Illinois, December 11-13, 1989 Sponsored by NSF

General Chair: Peter Scheuermann

Program Chair: Clement Yu

Program Committee

Ahmed Elmagarmid, Hector Garcia-Molina, Frank Manola, Dennis McLeod, Arnon Rosenthal, Marjorie Templeton

Executive Summary

Advances in networking and database technology during the past decade have changed dramatically the information processing requirements of organizations and individuals. An organization may have heterogenous database systems which differ in their capabilities and structure and which may be dispersed over a number of sites. In addition to these characteristics of heterogeneity and distribution, the ever larger number of databases available in the public and private domain makes it imperative to allow shared access to these databases in such a way that individual systems maintain their autonomy. Thus it becomes necessary to develop new techniques and provide new functionality to support the interoperability of autonomous database systems without requiring their global integration. Furthermore, the demands for interoperability extend beyond database systems to include office information systems, information retrieval systems and other software systems.

Research into the interoperability of heterogenous database systems plays an important role in the development of high level open systems. This important fact has been recognized not only in the United States but also in Japan and in Europe, with Japan having allocated around $120M over five years for research and development in this area.

The objective of this workshop was to explore current approaches to interoperability of autonomous information systems and to identify the most important research directions to be pursued in this area. This report summarizes our discussions and broadly classifies the issues into the following categories: [1]

1. Semantic Heterogeneity and Schema Integration
2. The Role of the Object-Oriented Approach
3. Transaction Processing
4. Query Optimization
5. Standardization Efforts

[1] Any opinions, findings, conclusions, or recommendations expressed in this report are those of the panel and do not necessarily reflect the views of the National Science Foundation.

Introduction

We are currently at a crossroads in the development of heterogenous distributed database systems. Advances in the networking and database technology have added two new dimensions to the problems to be solved, namely size and autonomy. The proliferation of public and private databases implies that for effective sharing of information we must provide tools to help users locate the information sources and learn something about their contents. In addition, organizations must maintain a certain degree of autonomy over their data in order to allow access to their information. We can distinguish different types (degrees) of autonomy: design autonomy, communication autonomy and execution autonomy. Design autonomy refers to the capability of a database system to choose its own data model and implementation procedures. Communication autonomy means the capability of a system to decide what other systems it will communicate with and what information it will exchange with them. Execution autonomy refers to the ability of a system to decide how and when to execute requests received from another system. Design autonomy usually has been assumed in distributed database systems, and this assumption brought with it the issue of heterogeneity. Here we can distinguish between data heterogeneity and system heterogeneity. Examples of system heterogeneity are differences in data model, data manipulation language, concurrency control mechanism, etc.

The workshop examined the impact of heterogeneity, autonomy and size on the development of federated database systems, in particular with respect to schema integration, transaction processing and query processing. We use the term federated database system or multidatabase system to refer to a collection of predefined database systems, called local database systems, that cooperate to achieve various degrees of integration while preserving the autonomy of each local database system. We explored the techniques and functionalities required to support the interoperability of federated database systems as well as the interoperability of database systems with other software systems.

This report summarizes the invited talks and position papers presented as well as the open discussion held with all the participants on the last day of the workshop.

Semantic Heterogeneity and Schema Integration

Each local database system in a federated architecture has its own conceptual schema, which describes the structural and dynamic properties of its information. Structural properties refer to the specification of object types and their relationships and constraints at various levels of abstraction. Structural description included both data and meta-data specifications. Dynamic properties describe how constraints are to be enforced and give the rules for update propagation in response to various operations.

We can observe a spectrum of database coupling that has been proposed (at the schema level) to support the interoperability of pre-existing database systems in a federation. At one end of the spectrum we find advocates of total integration, one global federated schema constructed under the responsibility of a global administrator. At the other end, we find systems that use partial integration, with the users themselves specifying which subsets of

the conceptual schemas should be grouped into partially federated schemas. While total integration may be feasible for a small number of databases, it appears that the partial integration approach is more desirable for an environment with many databases, some of which may appear and disappear on a daily basis. However, the problems of enforcing constraints and view update propagation across a number of partially federated schemas remain to be solved.

The schema integration process, whether it results in one or multiple federated schemas, presents a number of problems caused by various aspects of semantic heterogeneity and design autonomy. Schema integration includes the resolution of naming conflicts (e.g. homonyms and synonyms), scale differences, structural differences and missing data. Some tools for schema integration are being proposed to aid in identifying various object relationships such as equality, overlap, containment, etc. An issue that has not been addressed is that of determining relationships among objects that also exhibit behavioral abstractions as is the case in object-oriented systems. More importantly, it remains to be seen to what extent these tools can be automated and to verify their validity on real life systems.

If the federated database schema(s) uses a different data model from the local conceptual schemas, a schema translation module(s) must be provided for in the federated architecture. Efforts have been reported towards the development of specification languages for mapping among different data models. Although the problem of translation among data models received much attention in the early 1980's, the usual approach requires that a new mapper be implemented any time a new DBMS is added in the federation. The advantage of a specification language is that it would enable automatic generation of model mappers. While this seems a promising approach to the schema translation issue, it is not always possible to map from one data model to another and the extent to which this process can be automated for arbitrary models also is not known.

The approach to federated integration discussed so far assumes that the users or the database administrators have complete knowledge of the local conceptual schemas. However, in an environment consisting of hundreds of databases, the first step prior to any possible integration is to learn what information sources exist, where they are located, and what their contents are. A number of concepts have been proposed, but it appears necessary to couple these with AI techniques in order to help incorporate semantic knowledge into this learning process.

The Role of the Object-Oriented Approach

Object-oriented approaches are being considered as potential solutions to problems that exist in a number of software environments. Object-oriented DBMSs (OODBMSs) are being developed to allow databases to include "unconventional" data types such as text, voice, graphics, CAD data, etc. In a more general context, several papers at the workshop addressed the problems of using object-oriented approaches to provide interoperability of heterogenous computing resources, including both data and software components, and of integrating object-oriented databases with conventional relational database systems. Technology supporting these types of data and component integration will be crucial to the development of

the National Collaboratory, an infrastructure proposed by NSF to foster remote interaction between multi-disciplinary teams of scientists. The use of object-oriented approaches both complicates and eases the integration of heterogenous databases with other components.

First, increasing use of object-oriented systems will increase the complexity of the problem. The enhanced capabilities of object-oriented systems create the potential for increased heterogeneity of systems, since a richer collection of data types, as well as software components, will be included. Moreover, distributed object-oriented systems create the potential for users and programs to access vast areas of resources. This implies the need for increased assistance in simply selecting, let alone using, appropriate resources within the network. This problem was also mentioned in papers at the workshop.

Second, object-oriented approaches provide a natural framework for use in integrating heterogenous components. As a design approach, thinking of the components in a federation as objects or collections of objects allows a common design methodology to be applied to objects at all levels of granularity. As an implementation approach, an object-oriented approach provides a rich data model for use in problems of semantic heterogeneity. Behavioral modeling provides a framework for procedures required in inter-object communication, such as data conversion procedures, to be incorporated directly in the objects. Inheritance facilities found in most object systems provide a means of organizing similar data found in heterogenous components.

However, the papers at the workshop suggest that object-oriented approaches, at least so far, have had relatively little impact on some key aspects of problems in heterogenous systems. The architectures currently proposed for object-oriented heterogenous systems seem reasonably straightforward extension of those found in many existing heterogenous database systems. Although the use of objects provides a natural framework for including additional metadata such as units, time information, etc. that may be useful in attribute integration it is not clear how the complexity of this problem is simplified in object-oriented systems. Similarly, considerably more research is required in such problems as query optimization and concurrency control in the context of both object-oriented and heterogenous database systems. At the same time, since OODBMSs are, in a sense, inherently heterogenous (since they may include data of widely varying structure), work on OODBMSs and heterogenous databases will be mutually supportive in these key areas.

In addressing these problems, it will be necessary to make use of work in related technologies. Since object-oriented approaches deal with objects that include both procedures and data, the required technology overlaps such areas as database, programming languages, and operating systems technologies. Research in these areas is already becoming interrelated, as illustrated by emerging work in areas such as persistent programming languages and object-oriented distributed operating systems. It will be necessary to determine how these related technologies interact with the specific problems of heterogenous DBMSs. For example, as already mentioned, the development of advanced modeling facilities in object-oriented systems may well help in heterogenous database system development. On the other hand, work on data storage mechanisms in OODBMSs may have less effect, since the storage requirements in heterogenous systems will be handled primarily by the underlying local DMBSs.

It also will be necessary for researchers to gain more experience with real systems that include many large databases of realistic complexity. It is only this way that both the scope of the problems, and those aspects of real systems that sometimes allow for simplifying

assumptions, will become evident.

Transaction Processing

One of the key issues in federated database systems is transaction management. The problem is to make a collection of different database systems, running on different computers, cooperate and execute a transaction.

In conventional systems, a transaction is a collection of database actions that must be executed with three properties:

1. Atomicity: The entire transaction must be completed, or none of its actions should be executed.

2. Serializability: The execution of a transaction must be isolated from other concurrent transactions.

3. Durability: The values written by a transaction must persist in the database after the transaction completes.

Guaranteeing these properties in a federated system is difficult mainly for three reasons:

1. The actions of a transaction may be executed in different systems, each of which has different mechanisms for ensuring the properties of transactions. For instance, one system may use locks to guarantee the serializability property, while another may employ timestamps.

2. Guaranteeing the properties of transactions may restrict node autonomy, which may be undesirable in a federated system. For example, to guarantee the atomicity property, the participating systems must execute some type of a commit protocol. During this protocol, some systems must become subordinate to other systems, and the subordinates may not unilaterally release the resources, thereby compromising autonomy.

3. The local database systems may not provide the necessary "hooks" (functionality) to implement the required global coordination protocols. Again referring to the commit protocol, it is usually necessary for local systems to become "prepared," guaranteeing that the local actions of a transaction can be completed. Existing systems may not allow a transaction to enter this state (without committing all changes of the transaction to the local database), and providing this functionality may violate design autonomy.

Current efforts in this area may be classified into four (not necessarily mutually exclusive) approaches:

1. Develop strategies for meshing together existing but different transaction processing mechanisms. For example, some researchers have looked into mixed concurrency control algorithms (e.g., locking, timestamps, optimistic) and mixed commit protocols (e.g., centralized, distributed). Each local database system continues to use its native strategy, although there may have to be modifications so that the global mechanism can work.

2. Coordinate existing systems without any modifications. It is usually assumed that the systems share some basic concurrency control and recovery strategies, but that they must be globally coordinated. Each local system receives transactions either locally or from a global execution component; it treats all transactions in the same fashion. The global execution component must assure to it that the transaction properties are guaranteed for non-local transactions.

3. Weaken the properties that are guaranteed for transactions. In other words, new concepts are defined that encompass some, but not all, of the desirable properties of transactions. These new models make it easier to execute "transactions" in a heterogeneous environment.

4. Restrict the types of transactions and/or when they can run. If we can limit transactions a-priori, then it may be easier to guarantee the desired properties. As a very simple example, suppose that node a contains object x, while node b contains y. Local transactions at a and b can read and write the local objects. Suppose there is a single type of global transaction, which only reads x and writes y. In this case no global coordination is required. Each site can use its local concurrency control mechanism, and the resulting schedule will be serializable.

We note that among these four approaches, the first violates design autonomy, the second may violate execution autonomy, and the last two approaches aim to preserve autonomy at all levels.

Research in this area is at an early stage. A number of solutions to the heterogeneous transaction management problems have been suggested, but the solution space has not been fully explored. The exploration of weaker transaction models is at a particularly early stage. It is clear that the payoff in this direction can be significant, but a critical problem is finding a new transaction model that is useful in practice while at the same time allows efficient execution in a heterogenous environment. An almost completely open problem is the comparison of proposed solutions. A first step would be the definition of meaningful metrics for this environment. A second step would be a comparison of the options and tradeoffs.

Query Processing and Optimization

Query optimizers are optimizing compilers with the one notable difference that many query optimizers use quantitative estimates of operation costs to choose among alternative execution plans. Query optimization in homogenous distributed database systems has been an area that has received considerable study. The optimization techniques developed include, at one end, heuristic solutions that cannot guarantee optimality and, at the other end, exhaustive enumeration techniques that potentially may be very slow. In the middle of this spectrum we find techniques based on semijoin algorithms that obtain exact solutions in polynomial time for restricted types of query classes but are also of heuristic nature for arbitrary queries.

When dealing with a single real world object that comes from multiple local databases, the global query processor must solve the problem of data integration, e.g., how to assign ap-

propriate values to attributes in a global request when some of the underlying local databases overlap and disagree on the information or when some of it is missing. The outerjoin operation has been introduced to model an extended join operation in which null values are assigned to missing data. While it is easy to provide a reasonable implementation of the outerjoin, optimization of queries involving outerjoins is an unsolved problem. In particular, expressions that involve several outerjoins, joins and selections may need a new parenthesization (i.e. reassociation) to avoid computing large intermediate results, and the necessary theory is progressing slowly. Some papers at the workshop proposed more sophisticated interpretation of information combination by which some information that is not explicitly represented in the outerjoin can be deduced, rather than simply set to null. But these schemes will see little use if they require a complete overhaul of the query evaluator of the database system - they need to be implemented as extensions rather than replacements for the current algebras that underlie query processing.

One of the major problems in query optimization in a federated architecture is to develop an appropriate cost model, because many local systems have no facilities for estimating or communicating cost information. One option is to emphasize global optimization techniques that rely only on qualitative information such as semantic query optimization (perhaps using information from a generalization hierarchy). Another option is to implement a global optimizer based on a set of parameterized, modifiable cost equations. But customizing this optimizer to a particular local database system is difficult and must be repeated for every new database system release. In either case, cost information in a federated database system is likely to be inaccurate so it may be desirable to incorporate into the optimizer techniques for learning from past experiences.

Early federated database systems used a fixed high-level language, such as DAPLEX, as the multidatabase query language. A query in the multidatabase language must be translated into subqueries of the local database systems. Different translations can yield executions with widely different costs since some older DBMSs have little optimizing capability. Thus, it is important to provide translators that yield optimal or near-optimal subqueries in the local database systems.

Another important issue to be addressed in query optimization and processing is extensibility. It had been assumed that the multidatabase query language had more power than any of the local query languages. But new local systems may support additional operators on attributes that correspond to new data types or on entire relations (e.g., outerjoin, recusion) that are not available in the multidatabase query language. Researchers in extensible optimizers are examining ways in which one may declare the properties of new operators so that they can be exploited by the global optimizer. Thus, a new operator may require extensions to the multidatabase query language, to the optimizer's model of the local system's capabilities, to its cost model and its ability to exploit the new operator's properties. It is imperative to minimize the amount of effort it takes to modify the optimizer in response to the addition of a new local system or a new operator.

Standard optimization algorithms may also need changes to take into account hitherto unconsidered operations. Current optimizers assume that predicate evaluation should be done as early as possible because attribute comparison is cheap; this assumption may fail

for spatial and other user-defined data types.

Standardization Efforts

Although the goal of standardizing all hardware and software interfaces cannot be achieved, without some basic standards federated database systems are not feasible.

Standardization of network interfaces, file transfer formats, and mail formats has allowed extensive interconnection of computers for loosely coupled applications such as electronic mail. The next step is to move toward closer coupling at the data and program level, ultimately allowing programs to process data from any location and to pass partial results from an application on one computer to a second computer. This objective requires well defined standards for locating data, for understanding the semantics of the data, for understanding the capabilities of the data source, for connecting to the data source, for specifying data to be retrieved or updated, for transferring data, for controlling data access, for accounting for resource usage, and for dealing with errors.

Current efforts towards standardization are well underway in some important areas:

- RDA (Remote Data Access) defines a generic interface to a DBMS including database open/close, command execution, and transaction start/end.
- SQL defines a standard data manipulation language.
- TP (Transaction Processing) defines the behaviour and language for transaction processing.
- RPC (Remote Procedure Call) defines interprocess communication at the program call level.
- API (Application Program Interface) is being defined by a group of vendors.

The challenge is to extend standards to meet new requirements without making the systems that implement the standards obsolete. The SQL standard is a good example of a standard that is difficult to extend. SQL is based on the relational model and does not address DBMSs with extended functionality such as OODBMSs or older DBMSs and file systems which will continue to held data for many years.

Concurrent processing is not well supported by the present standards. RPC is a blocking protocol designed for client-to-server communication. RDA is also a peer-to-peer protocol. These standards need to evolve to support a model of multiple autonomous cooperating processes.

The TP protocol needs to support multiple models of transaction processing that can be negotiated between the client and server(s). Traditional protocols use serializability as the criterion for correctness. This may be too slow and too restrictive for some applications. CAD/CAM applications require very long transactions where availability for concurrent updates may be more important than serializability. Real-time applications may have hard time constraints that are more important than data consistency.

In addition to the extensions required in the current standards, new standards are needed. Tension between users who want easy access to all resources and owners of resources who

want to protect their resources. Before we can expect resource owners to open their systems to remote users, access control standards must be devised. Current data dictionaries provide insufficient semantic information and new standards for data definition are needed to include such information as level of abstraction, granularity and so on. Finally, we need standards for describing the capabilities and behaviour of systems, as well as standards for systems to advertize their data and services and for negotiating shared access.

Acknowledgement

Dr. Maria Zemankova, NSF Program Director of the Database and Expert Systems Program, recognized the substantial potential of this area and actively supported our efforts.

Research Directions for Distributed Databases

Hector Garcia-Molina
Department of Computer Science
Princeton University
Princeton, NJ 08544

Bruce Lindsay
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

1. Introduction

Communication networks make it feasible to access remote data or databases, allowing the sharing of data among a potentially large community of users. There is also a potential for increased reliability: when one computer fails, data at other sites is still accessible. Critical data may be replicated at different sites, making it available with higher probability. Multiple processors also open the door to improved performance. For instance, a query can be executed in parallel at several sites.

We have so far avoided the term *distributed database*. For some people, this term implies a particular type of distributed data management system where users are given transparent, integrated access to a collection of databases. In other words, a user is given the illusion of a single database with one global schema. Queries on this database are automatically translated into queries on the underlying databases. In the early days (before 1980) this was thought to be the ultimate goal for all distributed data management systems, and hence the term distributed database became associated with transparency and integration. Nowadays most researchers agree that transparency and integration may be incompatible with requirements for autonomy and diversity of implementations. They are using the term "distributed database" in a more general sense to mean a collection of possibly independent or federated database systems. Each system has some set of facilities for exchanging data and services with other members. In this paper we take the broader meaning of distributed databases in order to cover a wider spectrum of the challenging problems facing researchers.

This paper forms part of a collection of articles on current and future research issues in the database area. Since it is impossible to cleanly partition research areas, it is natural to expect overlap between the articles. In our case the overlap is more significant because two of the most important distributed database issues are being discussed in separate articles: heterogeneous and parallel databases. Many of the topics covered by other articles also have strong connections to distributed databases: security is especially critical in a distributed environment, scientific databases are often distributed, future DBMS architectures must have distribution in mind, etc.

In an attempt to reduce overlap, in this paper we will focus on distributed database issues that are not central to the other papers in this collection. Thus, we stress that our coverage here will be incomplete. Of course, even for the remaining issues, our discussion must be viewed as illustrative, not comprehensive. We are simply trying to point out some research areas that in the author's opinion have potential. For this, we have grouped our ideas into four broad areas and covered each in one of the following sections.

Before starting we would also like to clarify that due to space limitations this will not be a survey of relevant papers and work. We will actually avoid making references, for as soon as one reference is made, for fairness others must follow. Interested readers may refer to a Special Issue of the IEEE Proceedings (May 1987) on Distributed Databases. It contains many valuable references, as well as a discussion of state of the art distributed database processing. Current database textbooks are also a good source of references.

2. Distributed Data Architecture

Consider a user local to a database management system. Consider also a second remote database that the user wishes to access. How should the local system present the remote data? As discussed in the introduction, under a transparent, fully integrated architecture, the remote database is made to appear as part of the local one. Operations on the remote data, e.g., joining a remote table with a local one, can be done (at least from the point of view of the user) as easily as fully local operations. At the other end of the spectrum, the remote site may simply offer a set of services that may be invoked by explicit calls. For example, if the remote computer handles an airline database, it may let a user request the schedule for a given flight or reserve a seat on a flight.

Transparency is not an all or nothing issue. It can be provided at various levels, and each level requires a particular type of agreement between the participants. In the fully transparent case, the sites must agree on the data model, the schema interpretation, the data representation, the available functionality, and where the data is located. In the service (non-transparent) model, there is only agreement on the data exchange format and on the functions that are provided by each site.

The tradeoffs involved with providing or not transparency revolve around simplicity of access and ability to integrate data from diverse sources versus issues of site autonomy and specialized functions. Clearly, from the point of view of a user desiring remote access, a transparent architecture is desirable. All the data at the remote site is accessible, just as if it were local. However, from the point of view of the administrator of the remote site, transparency provides access that is difficult to control. The remote site could only make visible certain views on its data, but view mechanisms in many systems are not powerful enough to provide the desired protection. For instance, at a bank site funds transfer may only be allowed if the account balance is positive and the customer has a good credit rating. A simple view mechanism cannot express this.

It is much easier to provide these checks within a procedure that is called remotely. Although the data may be freely accessible to local users, remote users see the data encapsulated by a set of procedures, much like in an object oriented

programming environment. This type of remote service or federated architecture is simpler to implement than full transparency. Less agreement is needed between the participants, and complex algorithms such as a multi-site join need not be implemented. Sites have greater autonomy to change the services they provide or how they provide them.

The research challenge in this area is to fully understand the spectrum of alternatives. While we have sketched the two extreme solutions (full transparency and a service model), the intermediate models are not well defined. The fundamental issue is the *level* at which remote requests and responses are exchanged. Great care is needed to avoid weakening remote access functionality to the lowest common denominator while, at the same time, avoiding a proliferation of service and implementation specific protocols. Fruitful research directions include extending data access and manipulation protocols to support database procedures (to encapsulate services and policies), authentication standards, and relaxed serializability levels (with special authorization required for full serializability). In addition, further research is needed on technologies for exporting type definitions and *behavior* to allow remote users to exploit the semantic content of retrieved information (i.e., object distribution).

3. Problems of Scale

Current trends indicate that the number of databases is rapidly growing, while at the same time their size is also increasing. Some database and distributed data algorithms do not scale up nicely as the number of components in the system grows. In a conventional database, for example, one may need to place data off-line to make a backup or for reorganization. Typically, this is done during the night. As the database grows in size, the backup or reorganization time grows, and a night is no longer long enough.

In a distributed database, one is faced with such problems of scale as individual databases grow, and also as the number of databases and the scope of the system grows. For instance, in a world-wide distributed system, there is no night time to do reorganizations or backups. Key system algorithms may break down in larger systems. For example, in a small system, it may be feasible to search for a particular file of interest by broadcasting a request to all nodes. In a very large system, this becomes impractical. Having a central directory of all resources is also a bad idea, not just because of its large size, but because it is prone to failures and because not all sites may want to advertise their resources to everyone. Thus, the problem of resource finding in a very large distributed data system is quite challenging. When one starts a search, one not only does not know where the resource is, but one does not know what directories are available for this type of resource.

As an illustrative example, consider a scientist searching for a database of ozone readings over antarctica for the year 1980. (For one thing, "antarctica" and "ozone" are denoted differently in Russian databases.) Different organizations have directories of their own databases, but there is no reliable directory of organizations. Heterogeneity is an added complication here: it is not clear how to make our query understandable to different organizations, and if a relevant database is found, how to know what it really contains, expressed in our terms. While some progress has been made with yellow and white pages servers, the mechanisms for describing data resources in human

or machine readable form are quite crude. One only needs to try to use today's bibliographic search systems to realize that capturing and encoding the semantic or technical essence of data collections is not well advanced.

In addition to the resource finding protocols, there are other algorithms that may not scale up. The challenge is to identify them and to find alternatives. For instance, what deadlock detection, query processing, or transaction processing algorithm will work best in a very large system, or in a system with very complex queries, or with many participants in a transaction?

Administration of large distributed databases is another problematic area. As the number of components, users, and transactions rapidly grows, it becomes harder and harder to manage the system effectively. The volume of accounting, billing, and user authentication information grows. The number of choices for improving or speeding up a system grows. The size and number of database schemas grows. It becomes harder to evaluate the performance of the system and to predict its behavior under changes. Upgrading or installing new software also is harder, as there are more sites that are affected and it is impractical to halt the entire system to do an operating or database system upgrade. A related problem is the management of the underlying computer communication network. The problems are analogous: handling growing information on links, protocols, and performance. Also, key network algorithms do not scale up, e.g., those for initializing a network after its failure.

4. Information Management

Distributed systems are increasingly used for retrieving information on a wide variety of topics, ranging from stock prices, to cars for sale; from news items to jokes. In some cases the information is stored in commercial database services and a fee is paid. In other cases, computers are interconnected in ad-hoc networks and information exchanged between end-users, as in "net-news" or in bulletin boards. In still other cases, the communications companies themselves are providing information services. This is the case of MiniTel in France. Traditionally, these information networks have not been considered a distributed database. However, there is no reason why the mechanisms developed for formal distributed databases could not be extended to informal and/or loosely coupled information repositories.

In simple terms, the problem of information management is one of matchmaking. On one hand we have a user desiring some type of information (e.g., who has found this bug in this program?). On the other hand we have suppliers of information that may fully or partially match those needs. The goal is to make a connection between the supplier and the consumer. The problem is related to that of resource finding described in Section 3, but there are added complications.

Sometimes information requests are not for a single item, but are "standing orders", e.g., send me all future news items on this topic. This means the system must not only find the appropriate sources, but it must also set up the data paths for ongoing communication. Batching data transmissions is also important. For example, if two users at neighboring computers request the same information, it may be more effective to route the information to one computer, and to then forward it to the other. Existing systems such as net-news provide batching, but they make many restrictions as to what users may read and when they can read.

A provider of information often wishes not only to track who has received it, but also may need to be able to control how it is to be used. This will require facilities for access tracking and for "contracting" with the recipient. Important social, legal, political, and scientific issues must be addressed before open information distribution systems can be used for anything other than the exchange of "trivial" information.

Existing information systems usually do not provide reliability. In particular, a user may miss information if his machine is down. Thus, one challenge is to incorporate exiting distributed database crash recovery technology into information networks.

Distributed data and information can also be used in non-traditional ways, i.e., more than simply submitting "queries" and getting replies. For instance, electronic newsletters with user contributions are one such interaction. Users submit articles or news items to an editor or a group of editors. The editors check the articles, trimming them or eliminating uninteresting ones. The accepted articles are then distributed on the network to "subscribers" or are made available for queries. The National Science Foundation has recently suggested a "colaboratory," a distributed electronic laboratory where researchers can share information and conduct their research. Clearly, shared, distributed data must play a critical role here. The challenge is to expand the models and mechanisms of distributed database to encompass these new applications.

5. Transaction Models

In a federated database architecture, computers provide services to other sites. The glue that ties together the system is the transaction management system. It coordinates a sequence of interactions with different nodes, providing data consistency and atomicity.

Conventional transaction models based on locking and two phase commit may be inadequate. One reason is that they force participants to use the same model, and to possibly give up their autonomy. For instance, a participant in a two phase commit must become a subordinate of the coordinator(s), not releasing resources held by a transaction (locks) until instructed to do so. Another problem is that a large transaction may last for a long time, holding up critical resources. In a sense, this is a problem of scale: as the number of participants in the protocol grows, or that amount of work each participant must do grows, the time that resources are tied up also grows. This is unacceptable if these are critical, often accessed resources.

The need for relaxed concurrency control protocols in the local case has been recognized in some products and standards proposals. For distributed systems, there have already been numerous proposals for weaker transaction models that give participants more autonomy and improve performance, while still guaranteeing some basic correctness properties. For example, a sequence of steps at various sites can be considered a saga and not a full fledged transaction. After each step completes, a local transaction commits. If the saga needs to be aborted later on, a compensating step is run at nodes where transactions committed. This eliminates the need for two phase commit. However, sagas can now see intermediate results of other sagas, so programming such applications may be trickier. Also, the need for compensating steps creates more work for the application programmers. Tools for developing applications in a saga environment would be a very valuable contribution.

Without global transactions (as is the case with sagas and similar approaches), only consistency constraints local to a single site are maintained. Global constraints, e.g., object X at site A must be a copy of object Y at site B, are not necessarily guaranteed. If the inter-site constraints are known, it is possible to maintain them in an "approximate" way, e.g., making X approximately equal to Y or X equal to a relatively recent value of Y. Such approximate constraints may be adequate for some applications, e.g., an ATM machine may not need to know precisely how much money a customer has in his account; a rough estimate may be enough to make the decision if funds can be withdrawn. Approximate constraints may make it possible to operate without two phase commit for many steps or sub-transactions, improving autonomy and performance.

In general terms, there is a need for more options for transaction management in a distributed database. For each option it is necessary to precisely define the type of correctness it provides, and to evaluate the performance and autonomy it yields.

6. Conclusions

In order to extend data distribution to large scale environments, the requirements and characteristics of different DBMS implementations and information collecting organizations must be addressed. The challenge is to support integration of information from diverse sources without retreating to a low-function common protocol (e.g. NFS) while, at the same time, avoiding a proliferation of high-level, service-specific interfaces. Future research should consider carefully the role of remotely invoked procedures (which might have interfaces similar to other, general data accessing interfaces) as a mechanism to respond to organizational autonomy and control issues. Such interfaces could encapsulate local control procedures but execute in the context of a larger information interchange environment (e.g., authenticated user, transactions, accounting, data exchange formats, etc.).

We also believe that any success in providing distributed information processing facilities in a heterogeneous environment will probably rely, ultimately, on standard protocols for authentication, accounting / billing / contracting, data access specifications, schema exchange, typed object transport, and information resource characterizations. Accommodating multiple data models will increase the difficulty of developing mechanisms for information exchange and integration. The alternative is to provide the user with an array of information accessing tools, each of which can extract or manipulate data in a single type of DBMS using a unique user interface and communication protocol. Information integration from different sources in such an environment would be the user's problem.

Finally, we believe that one fruitful direction for data distribution technology is the extension of data semantics. This means that mechanisms for defining type specific behavior (procedures, functions, or methods) should, somehow, be extended to allow data objects to retain their semantic qualities as they are transported from one location to another. In some sense, this challenge echoes the earlier efforts to develop seamless functional distribution for the data model and its operations. Without techniques for transmitting objects without loss of their semantic qualities, information is pinned to the environment in which it is created and cannot interact (be integrated) with information (objects) from other cultures (environments).

ARCHITECTURE OF FUTURE DATA BASE SYSTEMS

Michael Stonebraker
EECS Dept.
University of California, Berkeley

Abstract

This paper explores the architecture of future DBMSs by posing a collection of questions that affect the construction of DBMSs and associated application development systems. These questions are grouped into ones which impact basic DBMS function, ones that impact operating systems, ones that deal with distributed data bases, and ones that consider user interface issues.

1. INTRODUCTION

This paper is motivated by numerous lengthy discussions that I had had with my colleagues over the last couple of years. Over and over again, we are drawn to examining the same questions. Moreover, on many of the questions there is fundamental disagreement over the desired (or correct) answer. Hence, I have taken the liberty to write them down, along with my opinion where appropriate, in the hope that the discussion will be illuminating. There are a total of 15 questions which are grouped into 4 categories:

Basic DBMS function
Operating system issues
Distributed data base issues
User interface issues

and we discuss them in the following 4 sections in turn.

2. BASIC FUNCTION

Our first question is motivated by the ever falling price of CPUs, main memory and disks.

B1: Will performance of DBMSs become unimportant.

If one wants to perform 100 transactions per second on a 100 megabyte data base, then it is probably cost effective to buy 100 megabytes of main memory connected to a high speed RISC processor. With such a configuration, it is straightforward to perform 100 TPS on the resulting main memory data base. Consequently, meeting the performance requirements of many current transaction processing applications can be cost-effectively accomplished by applying the appropriate amount of hardware to the application. In such a world, the performance of DBMS software becomes increasingly unimportant, resulting in the irrelevance of much of the current DBMS research.

On th eother hand, I firmly believe that performance will remain significant for the forseeable future for two reasons. First, most data bases are getting very large very quickly. Not only is average data base size going up at 35% per year, but also users are beginning to put very large objects into data bases such as text, image, and signals. As a result, I expect many data bases will continue to be large compared to available memory technology. Second, the complexity of transactions is also increasing rapidly. For example, many clients of airline reservation systems wish the lowest fare from point A to point B, no matter how

This research was sponsored by the Defense Advanced Research Projects Agency under contract N00039-84-C-0089, the Army Research Office under contract DAAL03-87-G-0041, and the National Science Foundation under contract MIP-8715235.

many intermediate stops are required. This query requires the transitive closure of the route map to be calculated, a much more complex task than that specified in standard benchmarks such as TP/1.

Because data base size and transaction complexity are increasing rapidly, I expect performance will continue to be very important. However, the most difficult data base problems will probably shift from transaction processing applications to decision support applications. As noted in [SILB90], the largest and most complex data bases occur in this arena. Consequently, there should be a corresponding shift in the research agenda to decision support issues. For example, traditional concurrency control and crash recovery will probably become less important and query optimization for large multi-way joins more important.

The second question is motivated by the desire to provide a better interface between DBMSs and general purpose programming languages.

B2: How will seamlessness and multi-lingualness co-exist?

Clearly, persistent X, for any programming language X, can be built. However, persistent X will satisfy very few users because it has no capability to specify queries, except by laboriously constructing algorithms for them based on low level primitives. Therefore, I expect many programming languages will be extended to have query language support, and such support may well be language specific. An example of such an approach is that of [AGRA89]. Because it is prohibitively expensive to implement a query optimizer and execution engine for each programming language, language designers will translate queries expressed in X into whatever query language the DBMS actually supports.

In addition, future DBMSs will clearly support functions that operate on data stored in the data base. Such functions are available in most current relational DBMSs and are called **data base procedures.** Such functions are coded in whatever programming language has been chosen by the DBMS vendor. However, the user would clearly like to use a given function to operate on transient data in his address space. This is the ultimate definition of a **seamless** DBMS interface. Unfortunately, this requires that the DBMS and X have exactly the same side effects and execution model. However, there are at least a dozen popular programming languages, and it is hopeless for the DBMS to seamlessly support them all. I have no idea how to resolve this apparent contradiction between multi-lingualness and seamlessness.

The third question is motivated by discussions with application designers.

B3: Where is major leverage in supporting application design?

Application designers universally state that they are required to model the "real world", i.e. the client's problem as he describes it, and they perform this modelling exercise using some abstract model. To be effective, this abstract model should have constructs that are semantically close to those of the real world. In business applications, this abstract model is often most usefully expressed as **processing steps** taking input from other processing steps and generating output to other processing steps. There are many **rules** which control the operation of the processing steps as well as the next processing step to perform.

The application designer must now translate this abstract model into a data base design and collection of application programs. Performing this translation is difficult because the data model supported by current DBMSs is semantically distant from the abstract model. Moreover, the current trend toward making the data model richer will not make a significant difference because the extended models have no notion of **processing steps.** For example, if a purchase order is filled out wrong, then it should be sent back to the originator for correction. Current and proposed data models do not contain constructs that could support such office procedures. In order to provide leverage to application developers, DBMS tools will have to include a process model as well as a data model, and work in this area should be encouraged.

The fourth question is motivated by a comment made to me by an MIS manager in at large application shop.

B4: What can be done to assist users with migration paths out from under their "sins of the past"?

Basically, he lamented that the research community is very good at dreaming up new functionality for

DBMSs, which is generally useful. however it does not address his crucial problem, i.e. he has a collection of 20 years worth of applications that he must maintain. These make use of previous generation DBMSs or no DBMS at all, and he can only afford to rewrite them over many years. Therefore, he needs migration tools, and the research community should focus on making a contribution in this area.

3. OPERATING SYSTEM ISSUES

The first question concerns the delivery of transaction services.

O1: Who will provide transactions?

Traditionally, DBMSs have contained significant code to provide transaction management services (concurrency control and crash recovery). However, such services are only available to DBMS users and not to general clients of the operating system such as mail programs and text editors. There has been considerable interest in providing transactions as an operating system service, and several OSs have done exactly that. Moreover, if a user wishes to have a cross system transaction, i.e one involving both a DBMS and another subsystem, then it is easily supported with an OS transaction manager but is nearly impossible with a DBMS-supplied one. Hence, the basic issue is whether DBMSs will continue to provide transaction support with DBMS specific code and algorithms or whether they will use the service provided by the operating system.

I am fairly optimistic that during the 1990s, transaction support will migrate from DBMSs into the operating system. I base this prediction on the following observations. First, the performance of the transaction system may become less important as users buy massive amounts of main memory to solve their performance problems as noted earlier. Second, it is plausible that transaction support may move into the disk controller. If so, the operating system will assuredly control such hardware or firmware. It is difficult to imagine that transaction software inside the DBMS will be competitive with an OS solution in the disk controller. Lastly, it is plausible that disk arrays [PATT88] will become the dominant style of I/O system. If so, there is a large incentive to perform sequential writes, since they are nearly free compared to random writes. This may cause file systems to change to log-structured ones. Such file systems support transactions much more efficiently than today's file systems [SELT90].

For these reasons, I believe that OS transaction support will ultimately prevail, and DBMS developers will not be required to implement transaction management. If so, it is important that there be a high bandwidth dialog between DBMS and OS developers to ensure that OS transaction systems are usable by the DBMS.

The second issue deals with the provision of buffer pool management services.

O2: Will virtual memory file systems prevail?

Some operating systems are moving toward providing file system services through virtual memory. In this architecture, when a file is opened, it is bound into a virtual memory segment and then read by referencing a page in virtual memory, which is then loaded by the operating system. Similarly, a write causes a virtual memory page to become dirty and be eventually written to disk.

If virtual memory file systems are successful, then buffer management code can be removed from DBMSs, resulting in one less problem to worry about. On the other hand, a virtual memory file system requires the efficient management of a very large (say 1 terrabyte) address space. Hence, I am unclear how this question will ultimately be resolved.

The third issue arises in DBMSs that are extended with user supplied functions.

O3: Will protection domains appear?

Extendible data base systems are clearly desirable, and there has been considerable research on how to provide them. Researchers have typically assumed that user written functions (methods) simply run in the same address space as the DBMS. Fundamentally no security is provided by such a system, as a malicious user can arbitrarily change DBMS data whether or not he is so authorized.

To do better, some help from the hardware and the operating system is required, namely the concept of protection domains or rings [ORGA72]. This would allow a DBMS to execute a protected procedure call to a user-written function with reasonable performance. Unfortunately, I see no reason to be optimistic that this capability will be forthcoming from the major hardware vendors. Therefore, I see secure, extendible DBMSs as a significant problem.

The last issue concerns the operating system notion of processes.

O4: When are OS processes going to get fixed?

A DBMS would prefer to fork an operating system process for each application program that uses its services. This so-called "process-per-user" paradigm is the easiest one to implement. However, it suffers from three fundamental flaws. First, there is no reasonable way to control the multiprogramming level in the resulting system. Second, a process-per-user DBMS tends to use excessive amounts of main memory. Lastly, every DBMS file must be opened and closed by each process, resulting in high overhead.

An improvement is to run the DBMS as a so-called **server** process. Hence, all users run in the same OS process and are multi-threaded by the DBMS code. Unfortunately, a server is a failure on a multiprocessor computer system, such as the IBM 3090-XXX or the DEC 58XX, because it can utilize only one physical processor. In order to run effectively on a multiprocessor, one must move to a **multi-server** (or server class) architecture. Here, there are a collection of N DBMS server processes, where 1 <= N <= #users, and each user submits his queries to one server.

Essentially all commercial DBMSs have implemented servers or multi-servers for these reasons. Therefore, they have paid the extra development effort required to write a task management system and associated scheduler. In effect, they have implemented an operating system running within a single OS process. All this pain and suffering is required because the operating system notion of a process is flawed. I would urge OS developers to proceed aggressively in this area. It is silly for the DBMS to be implementing an operating system on top of an operating system.

4. DISTRIBUTED DATA BASES

It seems clear that distributed data bases will be a dominant theme in the 1990's. Distributed DBMS issues have been well investigated in the past decade, and strategies for distributed query processing, concurrency control, crash recovery and multiple copies have been worked out [CERI84]. Moreover, many of the issues surrounding **heterogeneous** distributed DBMSs have also been considered. However, there are several questions in this area that may have significant ramifications. The first one concerns so-called block servers.

D1: Will distributed data bases prevail over block servers as a distributed processing model?

In a distributed DBMS, query execution will naturally occur at the site where the data resides. Put differently, queries are sent to the data they operate on, rather than bringing the data to the query.

However, consider a collection of data servers and diskless client workstations. Suppose a client repeatedly references a single record, e.g:

```
retrieve (EMP.all)
where EMP.name = "Bill"
```

Using distributed DBMS technology, the query will be sent from the client to one of the servers and a result returned in a second message.

On the other hand, consider a second architecture in which the DBMS is run on each of the clients and accesses data blocks as necessary from the servers. Consequently, query execution moves from the servers to the clients, leaving the servers only with the job of satisfying disk block requests. In this case, repeated reference to Bill's record can be satisfied from the local buffer in the DBMS at the client's site, and no messages need be exchanged. A benchmark showing the superiority of this block server architecture is presented in [CATT90].

On the other hand, the block server architecture is unlikely to be viable in a wide area network environment and has the same difficult synchronization problems present in any "shared disk" architecture. The most likely scenario is that both architectures will have a place, and commercial systems will be required to operate in either mode.

A second question concerning distributed architectures is the role of a transaction monitor such as IMS/DC, CICS or Tuxedo in a distributed application. Transaction monitors expect to manage the interactions between a client program and a collection of local or remote data bases managed by (perhaps) several DBMSs. Each is expecting to coordinate a two-phase commit and take appropriate recovery actions when crashes occur. Hence, a transaction monitor performs part of the function of a distributed DBMS. Hence the second question arises:

D2: Will distributed DBMSs prevail over transaction monitors?

It should be obvious that transaction monitors provide less function than a distributed DBMS. With a transaction monitor it is difficult to perform a distributed join and multiple copies of data objects are troublesome. As a result, distributed DBMSs have a considerable advantage. On the other hand, transaction monitors are in place today and present a natural evolutionary path forward into the future.

The last distributed data base question concerns heterogeneous distributed DBMSs. Clients want to have local data bases managed by a variety of general purpose DBMSs and to have such local data bases participate as members of a heterogeneous distributed data base. Therefore, most vendors are writing distributed DBMSs and are constructing gateways that map from their version of SQL to those of their competitors.

However, all major vendors are hard at work on extensions to SQL. None of the significant extensions (procedures, rules, type extension, inheritance) are covered by SQL 2 or SQL 3. Hence, each vendor has (or will soon have) an incompatible superset of SQL. Moreover, in each vendor's version of SQL there are functions that are essentially impossible to map through a gateway. Therefore, each vendor of a distributed DBMS will choose to support a subset of his extended version of SQL in each gateway.

Consequently, a user of a heterogeneous distributed DBMS has an inherent problem. He can choose to code his application in the superset of SQL supported by one particular distributed DBMS vendor. However, this locks him in to the set of gateways supported by the vendor of choice as well as makes his application non-portable to another vendor's distributed DBMS. On the other hand, he can remain within the subset of SQL that is standard across vendors, i.e the lowest common denominator. If so, he can easily move from one vendor of distributed DBMSs to another but must give up valuable function to do so. Hence, users are presented with a perplexing question:

D3: Should one bind oneself to the distributed DBMS from a particular vendor or use the lowest common denominator approach?

5. USER INTERFACES

There has been a great deal of interest in GUIs (Graphical User Interfaces) for data base applications. It is evident that end users will get powerful workstations on their desks with multiple windows, a mouse, display of icons, etc. In such an environment, all the traditional user interfaces, based on a glass terminal paradigm, are instantly obsolete, and new presentation services must be implemented. To support the construction of GUIs, a new collection of report writers, form systems, fourth generation languages, graphics packages, etc. must come into existence, and I fully expect these new toolkits to completely replace the current set. The only question is the amount of time that this transformation from glass terminals interfaces to GUIs will take.

U1: How long will it take for glass terminal interfaces to disappear?

A second question concerns the absorption of mips on the desktop. My Berkeley students have always complained about the performance of their current workstation and yearned for the next faster model. However, this viewpoint is changing rapidly. At the moment, they have 10-15 mips on their desktop, and view the move to 25 mips as nice but not remarkable. The problem is that their computing

environment is becoming both I/O bound and network bound, and a faster processor doesn't help much. Moving to a faster network requires extensive rewiring and considerable money and may take some time to occur. Hence, it is plausible that users will stop upgrading the workstation on their desk every couple of years. If this situation is widespread, then workstations may begin to have a much longer lifetime, and growth of the workstation market will slow markedly. Hence we are led to the next question:

U2: Is there a limit to absorption of mips on the desktop?

The next question is even more compelling. Speech recognition hardware and software will certainly mature in the 1990s. At the very least, it will be possible to economically process continuous speech from a single speaker who has taken the time to train the software to understand him. Vocabulary is now in the 1000-3000 word range with low error rates (< 1%), and this situation will only improve as the decade continues.

Therefore, it seems very likely that I will abandon text editing in favor of dictation as the means of interacting with my document processing and mail systems. This is a dramatic shift of paradigm which will require such systems to be rethought completely. Hence we are led to the question:

U3: What impact will speech processing have on user interfaces to DBMSs?

Finally, there is the perennial problem, namely data base design. At present, users have a lot of trouble performing this function for relational systems. Data base design in the future will clearly be more complex, because DBMSs will have a richer data model and execution environment. Distributed data bases will arrive with the resulting required optimization of location and replication of data. Hence, we are led to ask the last question:

U4: Will data base administration become impossible in the 1990s?

I am mildly uneasy that DBMS function will overrun the ability of users to manage it. Moreover, there is very little (if any) research on how to help users with this problem.

REFERENCES

[AGRA89] Agrawal, R. and Gehani, N., "ODE: The Language and the Data Model," Proc. 1989 ACM-SIGMOD Conference on Management of Data, Portland, Ore., May 1989.

[CATT90] Cattell, R. and Skeen, J., "Engineering Database Benchmark," Technical Report, Sun Microsystems, Mountain View, Ca., April 1990.

[CERI84] Ceri, S. and Pelagatti, G., "Distributed Databases: Principles and Systems," McGraw Hill, New York, N.Y., 1984.

[ORGA72] Organick, E., "The Multics System," MIT Press, Cambridge, Mass., 1972.

[PATT88] Patterson, D. et. al., "A Case for Redundant Arrays of Inexpensive Disks," Proc. 1988 ACM-SIGMOD Conference on Management of Data, Chicago, Il., June 1988.

[SELT90] Seltzer, M. and Stonebraker, M., "Transaction Support in Read Optimized and Write Optimized File Systems," Proc. 1990 VLDB Conference, Brisbane, Australia, Sept. 1990.

[SILB90] Silbershatz, A. et. al., "Database Systems: Achievements and Opportunities," University of Texas, Dept. of Computer Science, Technical Report TR-90-22, February 1990.

Object–Oriented Database Systems: In Transition

François Bancilhon

Altaïr
B.P. 105
78153 Le Chesnay Cedex, France

Won Kim

UniSQL, Inc.
9390 Research Blvd.
Austin, Texas 78759

1. Introduction

An object–oriented database management system (OODB) is a database system which supports an object–oriented data model. Just as any traditional database system, it must provide disk management, data sharing, data integrity, security, and a query language. Further, in support of an object–oriented data model, it must manage complex objects with object identity, support objects that encapsulate data and behavior, structure objects in classes, and organize classes in a hierarchy.

Starting around 1983, the field of object–oriented databases has rapidly turned into a major area of research and become reasonably mature. It has attracted significant attention from the research community, the business community, and the user community. During the past 7 years much has been accomplished:

- *Existing database and programming language technologies have been reused and adapted*: Concurrency control, recovery techniques, storage techniques, distributed data management, query optimization and processing, type theory, and compilation techniques are examples of such technologies.
- *New technologies have been developed for OODBs*: Data models, query languages, indexing techniques, query optimization, schema modification, user interfaces, authorization mechanisms, performance metrics, client/server architecture, and in–memory object managers are examples of such new developments.
- *Considerable experimentation has been performed*: Complete and partial prototypes have been built, with IRIS [Fishman *et al.* 87], ORION [Kim *et al.* 90], and O2 [Deux *et al.* 90] being the major prototypes. These prototypes have been used as vehicles of experimentation with most of the design and implementation issues for OODBs; they have also been used to evaluate the performance and functionalities of the OODB technology.
- *A first consensus on the definition and core concepts concerning OODBs has been established* [Dittrich 86], [Kim 90] and [Atkinson *et al.* 89]; and standards committees are currently attempting to forge standards for various aspects of the OODB technology.
- *Some products have been introduced in the market*: Gemstone [Maier *et al.* 86], Vbase and its successor Ontos [Andrews 87], and the commercial version of ORION are examples of currently available products. Further, a few additional products are being readied for release in 1990 and 1991.

This high level of productivity in research and development is mainly due to the fact that OODB researchers and designers have reused and adapted the technology and experiences accumulated from the development of relational systems in the 1970s. We believe that the exploratory and experimentation phase for the OODB technology is rapidly winding down, and the technology is poised to shift gear into the next phase to take root in production environments to fulfill the needs that traditional database systems have failed to satisfy. Further, we believe that the transition from the current phase to the next will take at least a few years, requiring intensive further research and consolidation. The research must be focused on bridging and negotiating the gap between the OODB technology and traditional database technology; and the best results of the past and current research and development efforts, not only in the field of OODBs but also extensible databases, distributed databases, and user interfaces, must be consolidated in building the next generation of OODBs for production environments. On the basis of these observations, we outline in this paper five topics we regard as the most important and promising, in terms of relevance to OODB developers and/or technical challenges to OODB researchers.

- query model and query optimization
- user interfaces
- design methodologies and tools
- view mechanism
- performance benchmarks

We note that there are other, albeit less pressing in our opinion, topics of relevant research, including the following.

- type theory and its integration into database languages
- architectural issues in implementing object managers for both client/server and distributed machine configurations
- specification and enforcement of integrity constraints
- design, usage and maintenance of libraries of reusable objects

2. Query Model and Query Optimization

2.1 query model

Although some promising research has been done on queries (query model, query languages, and query optimization), various problems remain. The major one is the definition of a query model that would account for encapsulated objects:

- Unlike traditional query languages which only access data, query languages for OODBs must access data encapsulated in objects by invoking methods. A method may be system–defined or user–supplied; in either case, a method defines a far richer set of operators on data than those supported in traditional database systems.

- It may be desirable to extend the *closure property* of the relational query languages to OODBs. A query language with the closure property takes as input a schema (and a database) and generates as output another schema (and another database). An OODB schema usually consists of a set of classes, classes consist of objects which have a structure and a set of methods. The query languages proposed for OODBs thus far are in general able to map structured objects into other structured objects. However, the objects returned do not necessarily belong to any of the existing classes; that is, to save the result of a query seems to require the creation of some artificial new classes somewhere on the hierarchy of classes. Further, query languages may take methods as "input" but do not generate objects with new methods.

If we are to define a formal foundation for OODBs as an extension of the relational framework, it is highly desirable to define a query model with the closure property. The notion of a complete query language defined for relational query languages will be helpful to formalize the power of the query languages for OODBs. A complete query language with the closure property will also serve as the basis for a view mechanism for OODBs, as we will see in Section 5.

2.2 query optimization

Query optimization for OODBs, just as for relational databases, involves enumeration of logically correct scan orderings for the classes, generation of query–execution plans, cost estimation for each of the plans, and application of algorithms and heuristics to reduce the search space for query–execution plans. However, it is more complex than in relational systems because

1. data has a more complex structure,

2. queries involve method invocation and thus optimizing a query can mean optimizing an entire program, and

3. there is no object–oriented equivalent of the simple and concise relational algebra (and it may be impossible to define such an algebra).

The complex structure of data in OODBs simply adds to the computational overhead in generating query–execution plans, but does not appear to introduce fundamentally new and significant problems that designers of relational systems have not addressed. However, queries involving methods have no relational equivalent, and pose some serious problems. If a method is used as an operator in a query, conventional access methods, such as a B–tree index, are no longer useful in optimizing the query; as such, new access methods are necessary to expedite the evaluation of a query involving such an operator. Further, the selectivity of such a general operator must be defined to augment the selectivities defined for conventional operators in the query optimizer of conventional relational database systems.

3. User Interfaces

Despite the longstanding consensus on the importance of user interfaces for non–programming users of database systems, the area of user interfaces for database systems remains largely barren. Most of the work has been focused on the design of a specific user interface for a specific system, rather than on the development of a *generic user–interface technology*.

It is tempting to leave the problem to researchers in the user interface discipline: after all, the database community does not develop operating systems to support database systems, so why should it develop user–interface management systems (UIMS) specifically for database systems and users? We believe that the problems that OODBs pose are particular and challenging, and they will be best addressed by the database community.

First, a specific data model, a specific schema structure, and a specific query language all require a specific user interface. At the very least, one has to design such an interface using available user–interface technology and tools. User interfaces for OODBs may very well become *applications* for a UIMS.

Second, even if one is to design a specific user interface for a specific OODB, the nature of the OODB introduces some new challenges, beyond those addressed for traditional databases. In particular, user interfaces for OODBs must deal with

- complex objects and their representations on the screen, either by graphs, by embedded graphical structures or by indented text structures (object sharing also requires an adequate visual representation on the screen),
- multimedia objects,
- display and manipulation of very large objects on the screen, and
- active objects.

Interfaces to traditional databases only had to deal with "passive data" which simply needs to be displayed and edited. In an OODB, objects have associated methods which may be activated directly on the screen, objects may reference other objects, and objects may be cut and pasted on other objects.

Thus, we believe that one must use existing user interface technology and eventually develop a new one to build systems which display complex interactive objects and connect them to an application. These systems should display interactively very large objects on the screen, display multimedia objects, provide a clean separation between the application and the interface and allow users to design interactively the display they want for objects.

4. Design Methodology and Tools

There is currently a need for a design methodology for OODB applications. It is one of the consistent feedbacks users have been giving to OODB experimenters. The standard questions are "where and how do we start" and "where do we go next?"

We must also address some cultural problems. There is a set of available object–oriented software design methodologies; however, they take little or no account of databases. Further, database application program-

mers have used existing methodologies for the past decade, based on some formal theories, and sometimes with associated tools. It is not reasonable to simply expect that the programmers will discard all this technology and culture. We must therefore devise new design methodologies for OODB application by either generalizing the existing ones or by providing a migration path from the old to the new methodologies.

5. View Mechanism

Views, as provided by relational systems, form a simple and powerful mechanism for data representation and structuring. The three–level ANSI/SPARC architecture which uses the view mechanism is a useful architecture to design complex applications involving several designers. Current OODBs do not offer a general view mechanism. A few of them offer some degraded form of views, either through exports of schemas or through encapsulation, but no complete and simple mechanism is yet available.

One of the reasons was mentioned in Section 2: current query languages are neither closed nor complete.

Another reason is the added complexity the view–derivation relationship introduces to the schema; that is, the view–derivation relationship results in a directed graph of classes and views along which some aspects of the class or view definition may be inherited. This "view hierarchy" is in addition to the class hierarchy and nested attributes of classes.

Object identity also makes view support in OODBs more difficult than in traditional systems. Typically, view definitions are stored in databases; but this seems to be difficult to do in OODBs because of the need for the generation of new identifiers. For example, let us consider a database of people, with some of them being married. Assume the schema describes this by objects of the class PERSON with an attribute spouse of class PERSON. Suppose now we need to extract a view of this database with objects of the class COUPLE with attributes HUSBAND and WIFE of the class PERSON. Then, this view must contain new object identifiers for the COUPLE objects. One interesting question is whether these identifiers should be persistent or transient/ virtual.

6. Performance Benchmarks

The question of the performance of systems is raised every time a new system or a new generation of systems is proposed. One of the important contributions the research community should make is to answer the question "what is a fast system?". This is usually done through benchmarks. The two traditional database benchmarks are the Wisconsin Benchmark which measures relational systems; and the ET1/TP1 benchmark which measures transaction systems.

A few benchmarks have already been proposed for OODBs: the SUN benchmark [Cattell 90], the Hypermodel benchmark [Anderson *et al.* 90], and the ACOB benchmark [DeWitt *et al.* 90]. All these benchmarks have been designed for engineering applications. Further, they only exercise complex object manipulation in OODBs.

We believe that a more general benchmark is necessary for OODBs. Such a benchmark should be designed for a much broader spectrum of application areas than just engineering applications, since applicability of OODBs is not confined to engineering applications. Further, the benchmark should be designed to exercise the large set of new database functions that OODB designers have been incorporating into the systems; examples are message passing, schema modification, version management, multimedia data management, etc. Finally, the benchmark should subsume relational benchmarks to allow a comparison across different OODB systems as well as a comparison between OODB systems and relational systems; this is an objective since there is a large degree of overlap in functions provided by OODB and relational systems, and an OODB data model can subsume the relational model of data.

7. Concluding Remarks

OODB research has been very active and productive during the past seven years, and the first batch of systems, incomplete as they may be, have reached the market in response to a growing demand for better solutions to data management problems than what the traditional systems offer. We believe that, although the OODB technology has significantly matured, there is much room for further research and experimentation

which should result in significantly more satisfactory next generation of systems. In this paper, we focused on five topics of research in OODBs which we regard as the most important and relevant in responding to the demands of the applications and which hold promise for exciting technical challenges.

References

[Andrews 87] T. Andrews and C. Harris. "Combining Language and Database Advances in an Object–Oriented Development Environment," " *Proceedings of the ACM Conf. on Object–Oriented Programming Systems, Languages, and Applications*, Orlando, Florida, Oct. 1987.

[Atkinson *et al.* 89] M. Atkinson, *et al.* "The Object–Oriented Database System Manifesto", *Proceedings of the 1st Intl. Conf. on Deductive and Object–Oriented Databases*, Kyoto, Japan, December 1989.

[Anderson *et al.* 90] L. Anderson, *et al.* "The HyperModel Benchmark," Proceedings of Conf. on Extending Data Base Technology, Venice, Italy, March 1990.

[Cattell 90] R. Cattell and J. Skeen. "Engineering Database Benchmark," SUN Microsystems Technical Report, Mountain View, Calif., April 1990.

[O. Deux *et al.* 87] O. Deux, *et al.* "The Story of O_2," *IEEE Trans. on Knowledge and Data Engineering,* March 1990.

[Carey *et al.* 88] M. Carey, D. DeWitt , and S. Vandenberg. "A Data Model and Query Language for EXODUS", *Proceedings of the 1988 ACM SIGMOD Conference,* Chicago, June 1988.

[DeWitt *et al.* 90] D. DeWitt, Ph. Futtersack, D. Maier, and F. Velez. "A Study of Three Alternative Workstation Server Architectures for Object–Oriented Database Systems," *Proceedings of the 16th Intl. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990.

[Dittrich 1986] K. R. Dittrich. "Object–Oriented Database System : The Notions and the Issues", in *Proceedings of the 1986 International Workshop on Object–Oriented Database Systems*, K. Dittrich and U. Dayal (eds.), IEEE Computer Society Press, 1986.

[Fishman *et al.* 87] D. Fishman *et al.* "Iris: An Object–Oriented Database Management System", *ACM Trans. on Office Information Systems,* vol. 5, no.1 January 87.

[Kim 90] W. Kim. "Object–Oriented Databases: Definition and Research Directions," *IEEE Trans. on Knowledge and Data Engineering,* March 1990.

[Kim *et al.* 90] W. Kim, J. Garza, N. Ballou, and D. Woelk. "Architecture of the ORION Next–Generation Database System," *IEEE Transaction on Knowledge and Data Engineering,* March 1990.

[Maier *et al.* 86] D. Maier, J. Stein, A. Otis, and A. Purdy. "Development of an Object–Oriented DBMS," *Proceedings of the ACM Conf. on Object–Oriented Programming Systems, Languages, and Applications*, Portland, Oregon, Oct. 1986.

Accommodating Imprecision in Database Systems: Issues and Solutions

Amihai Motro

Information Systems and Systems Engineering Department
George Mason University
Fairfax, VA 22030-4444

Abstract

Most database systems are designed under assumptions of precision of both the data stored in their databases, and the requests to retrieve data. In reality, however, these assumptions are often invalid, and in recent years considerable attention has been given to issues of imprecision in database systems. In this paper we review the major solutions for accommodating imprecision, and we describe issues that have yet to addressed, offering possible research directions.

1 Introduction

Information stored in a database is *precise* if it is assured to be identical to the "real world" information which it represents. When precise information is unavailable, it is often the case that some relevant information is nonetheless available. In these cases, it may be advantageous to design methods by which this information, termed *imprecise* information, can be stored, manipulated and retrieved. Imprecision may also be present in requests to retrieve data, when users, either intentionally or by necessity, formulate their queries in imprecise terms.

Depending on the data model used, the information stored in a database may take different forms, and imprecision could affect each and every one of them. For example, in the entity-relationship model, imprecision could occur in both entities and relationships. However, as virtually all recent research work in the area of database imprecision has been in the context of the relational data model, our discussion here is limited to this model. The structures of the relational model admit two different kinds of imprecision. The first kind involves imprecision at the level of data values; for example, the values of SALARY in the relation EARN (EMPLOYEE, SALARY) may be imprecise. The other kind involves imprecision at the level of the tuple; for example, the values of each of the attributes of the relation ASSIGN (EMPLOYEE, PROJECT) may be precise, but there may be uncertainty as to the precise assignment of employees to projects.

The relevant information that is available in the absence of precise data may take different forms. First, consider these examples of imprecision at the level of data values. It may be known that the true data value belongs to a specific set of values. Such imprecise data is often referred

This work was supported in part by the AT&T Affiliates Research Program.

to as *disjunctive* data. If the set to which the value belongs is simply the entire domain, then the imprecise data is referred to as *unavailable* (or *unknown*, or *missing*). If each of the candidate values is accompanied by a number describing the probability that it is indeed the true value (and the sum of these numbers for the entire set is 1), then the imprecise data is *probabilistic*. Note that probabilistic data subsumes all the types of imprecise data described so far, as well as precise data. Occasionally, the information available in the absence of precise data is a *descriptive* term. Such imprecise data is referred to as *fuzzy*. Imprecision is also possible at the level of tuples. For example, it may be known that the true tuple belongs to a set of tuples (disjunctive data), or a particular number may be available that describes the possibility that a given tuple belongs to the relation (fuzzy data).

It is useful to distinguish between the ability to accommodate *imprecise data* and the ability to handle *imprecise queries*, as each capability could be provided separately. It is possible to develop a model that accommodates imprecise data, and yet retain the standard query language, with its semantics extended to define when imprecise data match queries. On the other hand, it is possible to allow only standard databases, but extend the query language and the query processing methods to permit imprecise queries. By accommodating imprecise data, a database system provides for databases (and, therefore, answers) that are better approximations of the real world (the alternative being to ignore altogether information which is "imperfect"). The ability to handle imprecise queries is very helpful for requests which are intrinsically vague. With a database system that can only evaluate standard (specific) queries, the user must emulate such requests with standard queries. Usually, this means that the user is forced to retry a particular query repeatedly with alternative values, until it matches data that are satisfactory (if the user is not aware of any close alternatives, then even this solution is infeasible).

2 Solutions

Many of the approaches to the modeling of imprecision in databases are based on the theory of fuzzy sets [11, 12]. The approach described here is derived from [9, 14, 10].

The basic concept of fuzzy set theory is the *fuzzy set*. A fuzzy set F is a set of elements, where each element has an associated value in the interval [0,1] that denotes the *grade* of its membership in the set. For example, a fuzzy set may include the elements 20, 30, 40, and 50, with grades of membership 1.0, 0.7, 0.5 and 0.2, respectively.

As a relation is a subset of the product of several domains, one approach is to define relations that are fuzzy subsets of the product of fuzzy domains. Since each such relation is a fuzzy set, each of its tuples is associated with a membership grade. This definition admits imprecisions at the tuple level. For example, the tuple (Dick, Pascal) belongs to the relation PROFICIENCY(PROGRAMMER, LANGUAGE) with membership grade 0.9 [1]. To represent such relations, the usual attributes are supplemented with a column that assigns the membership grades to the tuples of the relation.

Consider the fuzzy set defined earlier, and assume it is named YOUNG. It is also possible to interpret this set as the definition of the term "young": it is a term that refers to 20 year olds with possibility 1.0, to 30 year olds with possibility 0.7, etc. Thus, fuzzy sets may be applied to describe imprecise terms.

[1] Alternatively, this tuple may be interpreted as stating that Dick's proficiency in Pascal is 0.9.

Consider now standard (nonfuzzy) relations, but assume that the elements of the domains are not values, but fuzzy sets of values. This definition admits imprecisions at the data value level. Having fuzzy sets for values permits specific cases where a value is one of five types: (1) A set; for example, the value of DEPARTMENT can be {shipping, receiving} or the value of SALARY can be 40,000–50,000. Note that the interpretation of such sets is purely *disjunctive*: exactly one of the elements of the set is the correct value. (2) A fuzzy value; for example, the value of AGE can be young. (3) An estimate; for example, the value of SMART can be 0.8. (4) A null value. (5) A simple value.

Finally, by defining a fuzzy relation as a fuzzy subset of the product of fuzzy domains of fuzzy sets, both kinds of imprecision can be accommodated.

To manipulate fuzzy databases, the standard relational algebra operators must be extended to fuzzy relations. The first approach, where relations are fuzzy sets but elements of domains are "crisp", requires relatively simple extensions of these operators. The second approach, where relations are crisp but elements of domains are fuzzy, introduces more complexity because the "softness" of the values in the tuples creates problems of value identification (e.g., in the join, or in the removal of replications after projections). Also, in analogy with standard mathematical comparators such as $=$ or $<$, which are defined via sets of pairs, the second approach introduces fuzzy comparators such as *similar-to* or *much-greater-than*, which are defined via fuzzy sets of pairs. These fuzzy operators offer the capability of expressing fuzzy (imprecise) retrieval requests.

Experimental database systems based on ideas similar to those described here have been implemented by various researchers. One example is the FRDB system [14]. Its overall architecture involves three components: (1) A database for storing extended relations (standard relations over domains of fuzzy sets). (2) An auxiliary database that stores the definitions of fuzzy sets that are used for domain values (e.g., YOUNG) and fuzzy comparators (e.g., *much-greater-than*). (3) Rules that define additional fuzzy concepts via the concepts stored in the auxiliary database (e.g., the definitions of "young age" and "high salary" may be combined to define "successful"). Each statement in the query language of FRDB performs one fuzzy relational operation, and a query is executed with a sequence of such statements. An optional argument of each statement is a *threshold* value between 0 and 1. This value is used to filter the tuples of the result, retaining only tuples whose membership grade exceeds the threshold value. Users of the FRDB system can present queries such as "select person, who is young and very smart", "select person who is much more intelligent than athletic, or who is not very young" and "select city, where summer-temperature is not much greater than winter-temperature, and is not so large".

An alternative approach to imprecise querying is to define *distances* among the elements of the domain, and then use these distances to derive measures of equality and similarity. By dividing each distance by the *diameter* of the domain (the largest distance among its elements), a measure of dissimilarity is obtained. Its complement to 1 is then a measure of similarity. Equality is defined as similarity with value 1. With an appropriate *radius* to serve as a threshold distance, every two values are either similar or dissimilar. A selection condition such as $A \sim a$ would then be satisfied by every element of the domain of A, that is similar to a. VAGUE [8] is an example of an extension to a conventional database system that uses data distances to interpret vague queries (another such example is ARES [4]). VAGUE extends the relational model with the concept of *data metrics*, and its query language with a *similar-to* comparator. A data metric defines the distance between every two elements of a domain; for example, in a personnel database there may be metrics to measure distances between titles, between salaries, between qualifications, as well as a metric to measure

distances between employees. Database designers are provided with several different methods for defining data metrics, and users are offered several features with which they can adapt the available metrics to their own needs. A *similar-to* comparison is satisfied with data values that are within a predefined distance of the specified value; for example, the vague comparison "language *similar-to* Pascal" may be satisfied by Pascal, Algol and Modula. To present queries, users need only to know about the new comparator. In contrast with a standard query, that establishes a rigid qualification and is concerned only with data that match it precisely, a vague query of the kind that can be handled by VAGUE establishes a *target* qualification and is concerned with data that are *close* to this target.

Buckles and Petri [1] define fuzzy databases differently. Relations are extended to allow values that are *sets* of domain elements. Therefore, a fuzzy tuple is a sequence of sets $(d_1, d_2, \ldots, d_n)$; it represents a *set* of specific tuples $(a_1, a_2, \ldots, a_n)$, where $a_i \in d_i$, for all i. Thus, a fuzzy tuple models uncertainty: each of the specific tuples that it represents is equally likely to be the "correct" tuple. Additionally, each database domain has an associated *similarity matrix* that assigns a value between 0 and 1 to each pair of domain elements. The output of standard relational algebra operators, such as join or project, is post-processed to merge tuples (by performing unions of their respective components), if a pre-specified similarity threshold is not violated.

Recently, Garcia-Molina and Porter [3] suggested modeling uncertainty by using traditional probability distributions (rather than the possibility distributions of the fuzzy models). A *probability distribution function* of a variable X over a domain D assigns each value $d \in D$ a value between 0 and 1, as the probability that $X = d$. One important difference between probability and possibility distribution functions is that the sum of the probabilities assigned to the elements of X must be exactly 1. The definition of a probabilistic database is similar to the second definition of fuzzy databases: standard relations, but with domain values that are, in general, probability distribution functions. A feature of this model is that it allows probability distributions that are incompletely specified: each such distribution is complemented with a "missing value" which is assigned the balance of the probability.

If a model such as a fuzzy data model is not used, then all imprecise information must be ignored, and the information is then considered *unavailable*. When the value of a particular attribute for a particular tuple is unavailable, a special value, called *null*, is stored instead. Hence, unavailability is the least informative form of imprecision. Recall that in many of the models mentioned earlier, unavailability, also called *incompleteness*, is handled as a special case. Due to the prevalence of this type of imprecision, and the relative simplicity of the "information", unavailability has been the subject of intensive research (see [6, Chapter 12]). Much of this research focuses on the precise semantics of null values, and on the appropriate extension of the relational algebra to databases with null values (e.g., [2]).

The models described so far incorporate modeling extensions that permit modeling imprecise information. An alternative approach is discussed in [7]. Instead of modeling the "imperfections" of the available data, it suggests declaring the portions of database that are assured to be prefect models of the real world (and thereby the portions that are possibly imperfect). With this information included in the database, the database system can *qualify* the answers it issues in response to queries with regard to their precision: each answer is accompanied by statements that define the portions that are guaranteed to be perfect.

3 Discussion

Commercial database systems have been relatively slow to incorporate imprecision capabilities: the only capabilities widely available are for handling null values, and for specifying imprecise queries through the use of regular expressions. Examining the possible reasons for this slow acceptance may suggest directions for further research. First, database practitioners are concerned primarily with the *performance* of a database system. However, many of the query evaluation algorithms for matching imprecise data or for processing imprecise queries are fairly complex and inefficient. Practitioners are also concerned with *compatibility*. This dictates that capabilities for accommodating imprecision should be offered as strict extensions of existing standards. Third, database practitioners have often been dissatisfied with various *idiosyncratic implementations* of imprecision capabilities (minor examples are the inability to specify an incomplete date value, or the inability to sort answers with null values flexibly). This may have had a chilling impact on further implementations.

Another hindrance for database systems with imprecision capabilities may lie in the expectations of users. A fundamental principle of database systems has been that queries and answers are never open to "subjective" interpretations, and users of database systems have come to expect their queries to be interpreted unambiguously and answered with complete accuracy. In contrast, users of information retrieval systems would be pleased with a system that delivers high rate of recall (proportion of relevant material retrieved) and precision (proportion of retrieved material that is relevant). A database system that must adhere to the principles of unambiguity and complete accuracy cannot accommodate the full range of imprecise information; for example, it can accommodate null values or disjunctive data, but not (due to its subjective nature) probabilistic or fuzzy data. Further research is required on systems that would integrate the capabilities of information retrieval systems and database systems attractively.

As mentioned earlier, most of the research effort on imprecision has been in the context of the relational data model. At present, much research and development efforts in the area of databases is focused on "next generation" database systems [2]. It appears advantageous to incorporate imprecision capabilities into early versions of these systems, as such extensions are more difficult to effect once "standards" have been agreed upon. Preliminary work in this area includes investigation of incompleteness in logical databases [5] and the design of a knowledge-rich system that can deal with various aspects of imprecision [13].

The ability to respond to imprecise queries may be considered as one aspect of *generalized* (non-standard) query answering. Other aspects include *intensional* answering (where answers are described with predicates or constraints), *cooperative* answering (where information considered relevant to the query is delivered along with, or in place of, the standard answer), and *approximative* answering (where an approximation is provided in lieu of the standard answer, possibly because of limited computational resources). Various models and techniques have been developed to handle these individual aspects of generalized query answering, and further research is required to integrate them into a single model of retrieval.

[2]The term is used loosely to refer to object-oriented or knowledge-rich systems.

References

[1] B. P. Buckles and F. E. Petry. A fuzzy representation of data for relational databases. *Fuzzy Sets and Systems*, 7(3):213–226, May 1982.

[2] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, December 1979.

[3] H. Garcia-Molina and D. Porter. *Supporting Probabilistic Data in a Relational System.* Technical Report TR-147, Department of Computer Science, Princeton University, February 1988.

[4] T. Ichikawa and M. Hirakawa. ARES: a relational database with the capability of performing flexible interpretation of queries. *IEEE Transactions on Software Engineering*, SE-12(5):624–634, May 1986.

[5] T. Imielinski. Incomplete information in logical databases. *Data Engineering*, 12(2):29–40, June 1989.

[6] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville, Maryland, 1983.

[7] A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, December 1989.

[8] A. Motro. VAGUE: a user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, July 1988.

[9] H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. *Information Sciences*, 34(2):115–143, November 1984.

[10] K. V. S. V. N. Raju and A. Majumdar. Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *ACM Transactions on Database Systems*, 13(2):129–166, June 1988.

[11] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.

[12] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1978.

[13] M. Zemankova. FIIS: a fuzzy intelligent information system. *Data Engineering*, 12(2):11–20, June 1989.

[14] M. Zemankova and A. Kandel. Implementing imprecision in information systems. *Information Sciences*, 37(1,2,3):107–141, December 1985.

Research Issues in Spatial Databases

O. Guenther
FAW Ulm
Postfach 2060
D7900 Ulm, Germany

A. Buchmann
GTE Laboratories, Incorporated
40 Sylvan Road
Waltham, MA 02254, USA

1. Introduction

With a recent emergence of applications that rely heavily on spatial data, the database research community is currently devoting considerable attention to spatial database issues. While the main thrust came initially from the geosciences and mechanical CAD, the range of possible applications has expanded to areas such as robotics, visual perception and autonomous navigation, tracking, environmental protection, and medical imaging.

Just as broad as the range of applications is the range of interpretations given to the term "spatial data management." One interpretation comes from mechanical CAD and its need for processing 3-dimensional solids, whereas VLSI CAD and cartography are typical for applications that rely mostly on 2D- or layered 2D-data. Other applications emphasize the processing of unanalyzed images, such as X-rays and satellite images from which features are extracted.

The terms spatial database and image database are often used interchangeably. Strictly speaking, however, spatial databases contain n-dimensional data with explicit knowledge about objects, their extent and position in space. The relative position of objects may be implicit (derived from the internal representation of their absolute positions) or explicit. Image databases, on the other hand, often place less emphasis on the need for analyzing the data and provide storage and retrieval for unanalyzed pictorial data. Techniques developed for the storage and manipulation of image data can be applied to more general signals as well, such as infrared sensor signals and sound. In the remainder of this paper we will assume that the goal is to manipulate analyzed spatial data, and that unanalyzed images are only handled as the source from which spatial data can be derived. The operations performed are often domain-specific, and some of the features we have come to associate with a full-fledged DBMS, such as transaction management, may be less important in some applications. However, the handling of multiple spatial representations and data models, spatial access methods, pictorial/spatial query languages, and optimization form a significant subset of today's database research.

The challenge for the developers of DBMSs with spatial capabilities lies not so much in providing yet another special-purpose data structure that is marginally faster when used in a particular application, but in defining abstractions and architectures to implement systems that offer generic spatial data management capabilities and that can be tailored to the requirements of a particular domain.

In this paper we do not attempt a comprehensive review. Instead, we will summarize application requirements, identify areas in which considerable progress has been made, and discuss several problems that require a stronger emphasis.

2. Application Requirements

The spatial data management needs differ widely between applications. Mechanical CAD is perhaps the application with the most demanding geometrical requirements. Early representations consisted of line drawings of the three views of a solid. Cell decompositions, in which solids are decomposed into simpler polyhedric cells (such as convex polytopes or simplices), are best suited

for finite element methods. Constructive solid geometry (CSG), which represents solids as compositions of primitive components and regularized set operations, is best for composing regular objects and deriving other properties. Boundary representations, which map a solid's surface into patches that are represented via edges and vertices, are best suited for modelling irregular surfaces. Because no single approach satisfies all requirements, the same object is often represented in multiple forms. In addition to the representation and manipulation of the spatial data, mechanical CAD applications often require its association with descriptive properties, such as material and weight (which may itself be derived from volume and density data). Database retrievals and updates are frequent, but often performed in individual workspaces.

VLSI CAD has received most attention in the database community, both because it is easily accessible to computer scientists and because its geometric requirements are relatively simple and tractable. Most objects are represented sufficiently well by rectilinear polylines or polygons whose edges are all parallel to the axes of the same coordinate system. This makes for regular boundaries and clean intersection of the geometric shapes, all properties that help when indexing on an object's spatial properties.

Geographic/cartographic applications are characterized by massive volumes of data, both spatial and non-spatial, as well as the need to record their evolution in time. Spatial data are mostly two-dimensional with points, lines, and polygons as the basic primitives, and are characterized by extreme irregularity. The third dimension is usually reduced through well-established conventions for the representation of elevation. Cartographic applications require the ability to overlay maps, extract common features and make independent data sources compatible. Feature extraction from aerial photographs and spatial images is common. The operations most often performed are overlay, containment, distance operations, and spatial searches. The sheer volume of geographic data prompted early application of database technology. However, the integration of spatial and non-spatial data in a common database approach, although desirable, has been limited.

Vision systems have been evolving in the areas of robotics and manufacturing, as well as autonomous navigation [Good89, Brol89]. The general problem is one of identifying and determining the position of a 3D object based on 2D images. Vision systems are driven by tight time constraints, and generally depend on the extraction of relevant features, such as descriptions of junctions, that are stored in structures used for fast matching. Features are often extracted from the underlying CAD databases to produce models of the objects off-line.

3. Research Areas in Spatial Data Management

The spatial properties of objects have to be captured both at the conceptual level, and at the physical data structure level. How tightly these two are coupled will determine the flexibility (or lack thereof) in a spatial database system. We will analyze the issues by looking first at the representations of spatial data and working from there outward to the conceptual level, query languages, optimization, and problems derived from intrinsic spatial properties, such as precision.

3.1 Representations

The goal of a spatial data structure is to provide a scheme for partitioning the entire space into cells, and to provide a two-way mapping between these cells and the region in space occupied by an object [Niev89]. A variety of representations have been proposed; for a review see [Nagy79, Requ80, Gunt88, Same89]. The simplest approach is a raster representation in which the spatial extent of an object is represented by pixels. 2D objects can also be represented by polygons made up of lines (so-called vector representation). The equivalent for 3D is the representation of a solid through surface patches. Solids can also be composed of primitive objects and a specification of their relationships, which is the basic representation in CSG systems. Each of these representations supports a certain class of spatial operators, such as intersection, containment, or the set

operations. However, some operations are not defined for certain representations. For example, outline and endpoints of a line are not defined for a pixel representation. Operations are also performed with varying efficiency in different representations. For example, raster representations support intersection well but vector representations are more efficient for distance operations. When choosing a data structure it is therefore important to take into account what operations are to be supported primarily.

However, no matter how carefully one selects a representation, operations across representations will be required. This is even more so as systems with different underlying representations will be integrated. A variety of open research problems have hindered data exchange and integration of spatial databases, and has confined spatial databases to the use of a single representation.

For most of the representations the specification of the spatial operators is incomplete. No comprehensive list of operators exists, no minimal sets of operators have been identified, and the semantics of many operators are ill-defined. Presently, no known generalizations exist for spatial operations across representations. For each of the representations an algebra is required and a "superalgebra" for cross-representation operations has to be defined if the user is to be shielded from the physical representation and if any kind of optimization across representations is to be performed. The need for representation-independent abstractions is discussed below.

3.2 Spatial Access Methods

A typical spatial database query may ask for all objects that contain a given point (point search) or that overlap a given search space (range search). In all of these cases, it is necessary to retrieve from the database those objects that occupy a given location in space. To support such search operations, one needs to use spatial index structures that enable the user to access efficiently the objects in a certain spatial neighbourhood.

A large variety of such structures have been proposed, both hash-based and tree-based; see [Gunt88] or [Ooi88] for a survey. However, due to a lack of comparative studies it is not clear yet which structure has the best expected performance for a given application, and how robust each is. It is an important issue for future research to classify spatial search applications according to a small set of parameters (such as database size, average object size, or object distribution), and to determine the most promising structures for a given set of parameters. For initial experimental results see [Ooi88, Free90, Krie90, Gunt90, Smit90]; no analytical results are known to us.

Unfortunately, many spatial access methods are rather inefficient if the database contains objects of varying shapes and sizes [Six88]. In geographic applications, for example, where a spatial database may contain geometric objects representing geographic entities such as states, roads, and buildings, this problem may have a significant impact on average performance. The reasons for these inefficiencies depend on the particular properties of the various index structures. Most of these structures have originally been designed to manage point data only, that is, to provide efficient spatial access to large sets of points. Usually, the points in the database are organized in a number of buckets, each of which corresponds to some partition of the data space. The buckets are then accessed by means of a search tree or some hashing scheme.

Problems arise when these point access structures are modified to manage extended spatial objects such as polygons. As discussed in [Seeg88], there are three basic methods: clipping, where objects are divided along the partitioning lines of the underlying access structure; overlapping regions, where each partition of the access structure may contain any object it overlaps (rather than just those objects it encloses); and transformation, where extended spatial objects are mapped into higher-dimensional points.

Transformation only works for relatively simple objects, such as rectangles. More complicated entities, such as arbitrary polygons (as they typically occur in geographic applications), can only be represented by means of their bounding boxes [Niev84]. However, this principle has led to the development of spatial filters, which return approximate (conservative) results which can then be refined by more precise searches based on the actual geometry [Oren86].

Overlapping region schemes tend to experience major performance problems when the spatial database contains objects whose size is large relative to the total size of the data space. Each insertion of a new data object may increase the overlap. Eventually, the overlap between regions may become large enough to render the index inefficient: one ends up searching a significant portion of the whole index for a single point query. A well-known example where this behavior has been observed is the R-tree [Gutt84; Gree89].

In the case of clipping schemes, the problems with large objects are of a somewhat different nature. During insertion, each data object is divided along the partitioning hyperplanes of the access structure. This fragmentation effect gets worse as the spatial database and its index continue to become more populated. The net result of this fragmentation effect is not only an increase in the average search time but also an increase in the frequency of node overflows, which in turn leads to an increase in fragmentation, and so on [Gunt89]. Such an endless loop may cause the whole index to collapse and therefore has to be avoided at all cost.

It is evident that ever more specialized access methods can be devised. While these may perform well in a narrow application with "well-behaved" data, it is more important in a database environment to develop a sound understanding of the behavior of a few robust access methods under varying conditions, and to develop performance metrics that can be employed by an optimizer. It is important to realize that spatial indexes and filters can be constructed on most underlying representations, and can be substituted at a moderate cost. However, changing the representation of the data is a major undertaking and is often coupled with a loss of information or the introduction of imprecisions or inconsistencies.

3.3 Conceptual Models and Architectural Considerations

The objects handled in spatial database systems are usually complex. Some objects may be atomic (e.g., a building in a city map) while others may have a molecular structure (e.g., a city that is made up of districts, which consist of streets and buildings). For the representation of structured objects, one may resort to semantic networks [Luge89, Ch. 9], or object-oriented techniques [Luge89, Ch. 14; Ditt88; Oost89]. Depending on the scale at which one views such an object, its components may or may not be visible. Associated with each object is a geometric entity representing its spatial extension. For the lowest level objects this is usually a point, a line, or a polygon. The spatial extent of more complex objects may be explicit or derived from that of the component objects, for example, by inferring their geometry using cartographic generalization techniques [McMa87; McMa88]. In many cases, however, this is not a practical solution, since derivation algorithms do not consistently produce "nice" geometric representations. The utilization of object-oriented techniques for spatial data management is an active research topic, but the application of inheritance has not been researched thoroughly and deserves further attention.

The fundamental question is how to embed the spatial aspects in a data model and the underlying DBMS such that acceptable interfaces (optimizable query languages and pictorial interfaces) can be defined.

Early attempts decomposed the spatial data and forced them into relations, which resulted in unacceptable performance [Kemp87]. Other systems implement the spatial operations in a "hard-wired" form using conventional DBMS technology for non-spatial data and special-purpose file

structures for the spatial data [Moor85]. A more general approach consists in providing abstract data types (ADTs) to customize the DBMS. Examples are INGRES [Ston84] and PSQL [Rous88]. Both approaches, hardwiring and ADTs, leave the spatial semantics outside the DBMS's query processor. It is desirable for the DBMS to perform set-oriented queries on abstract objects that are representation independent, leaving the detailed spatial operations to be performed by specialized object classes that understand a specific representation.

To formulate abstract operations on spatial objects it is desirable to define a structure that is independent of any underlying representation. The notion of point sets [Mant83, Mano86] serves this purpose. It is a specialization of type "object" that acts as a placeholder and introduces a collection of spatial operators, such as union, intersection, and difference, as well as spatial predicates, such as overlap, containment, and proximity. It is similar to the abstraction of integers and floating point numbers on which operations can be defined independently of the particular representation of these numbers in a computer. The type Point-set can be further specialized, for example, creating subtypes for 2D and 3D. Each of these can, in turn, be specialized by defining representation-specific point-sets. The operations (or methods) defined on these lower level types are the implementations of operations such as intersection or overlap for a particular representation, for example, boundary representation. Operations performed on spatial objects of different representations are defined as methods of more general types.

While point sets are a good abstraction for expressing relationships of equality, inclusion, and intersection, they lack the power for describing relationships such as *neighborhood* in the topological sense. This has led to the definition of data structures for topological relationships based on simplices, and the formulation of theories of topological relationships [Egen89].

Further research is required in the specification of representation-independent spatial abstractions and their operations as a basis for seamless integration of spatial and non-spatial data in query languages. These issues will be addressed further in Section 3.5.

3.4 Spatial Database Languages

The goal of a spatial query language is to allow the easy formulation of queries that involve both spatial and non-spatial predicates without loss of spatial semantics. The query should be optimizable by a DBMS query optimizer. While the former has been accomplished in several ways, the latter, optimizability, has been elusive.

Most spatial query languages are extensions of a relational query language, either QUEL (e.g.GEO-QUEL [Go75], QBE (e.g. Query-by-Pictorial-Example [Chan80] and GEOBASE [Barr81] which also incorporates inheritance and aggregation), or SQL (e.g. PSQL [Rous88] and Extended SQL [Herr88]). A good review and comparison can be found in [Egen89]. The widespread use of SQL for non-spatial data has made SQL extensions popular. Recently, a variety of object-oriented languages have been proposed (e.g. [Oost89], [Moha88]). They are able to represent the semantics of complex objects better and encapsulate specialized operations. A fully pictorial approach to constraint definition is taken in [Piza89].

As pointed out in [Egen89], all existing languages use a diverse set of spatial relationships (containment, adjacency, distance, overlay, etc.). In most cases the semantics of these spatial relationships have not been rigorously defined, and would benefit from a mathematical grounding.

Depending on the method chosen for embedding spatial characteristics in a data model and the underlying DBMS, the query languages may be of varying complexity. For example, if one chooses ADTs to represent the objects, then the internal structure of the object representation is hidden to the database system (encapsulation). In this case, it is sufficient to enhance the query

language by some simple constructs that allow the embedding of user-defined data types and operators. However, the performance characteristics of the ADTs are unknown to the optimizer.

On the other hand, it is possible to use object representations that allow the user to access the structure of an object explicitly, for example, by embedding in non-first normal form relations [Sche82]. In this case, the database contains information about the hierarchical structure of a given object and the query language may have to be extended significantly in order to access this information efficiently [Kemp87].

In systems that use an object-oriented approach, user-defined object classes and encapsulation of their behavior make it difficult to optimize queries. The optimization of object-oriented query languages and the definition of optimizable object algebras is an active research topic. Once the generic problem is solved, it will be possible to apply those solutions to spatial query languages.

3.5 Optimization of Spatial Operations

Queries that contain both spatial and non-spatial attributes may require optimization algorithms that are significantly different from those that have been used for conventional business databases. Existing spatial data management systems (many of them prototypes) have basically ignored the optimization issues, not only because they often have only one underlying representation and one spatial index, but also because little is known about the cost of executing alternate sequences of spatial operations.

Research is required to establish cost estimates for spatial operations. Many optimal algorithms for spatial operations that are derived from computational geometry have not matured enough in practice; they often optimize for the worst case and require very complex data structures. Suboptimal algorithms that perform adequately in the majority of cases, and for which detailed cost functions are known, may be preferable.

The use of spatial filters appears to be a reasonable optimization strategy. Filters that are based on space-filling curves, such as Z-order encoding, allow for a two-phased optimization strategy. The filtering phase returns an approximate answer that is conservative and includes all the objects to be retrieved, and possibly more. Filtering reduces the set of candidate objects on which detailed operations have to be performed. The second phase then uses the specific spatial operations discussed above.

Much more research is required in this direction; for a more detailed discussion of related problems see [Oren89, Kemp87, Ooi88].

3.6 Scale and Precision

In spatial applications the spatial extent of an object has a limited precision. Handling spatial objects with varying precision is difficult at best. Errors in spatial data representations can propagate leading to topological inconsistencies (known also as wandering points). To avoid the problem of points contained within a region at one scale to appear outside the region at another scale, and false intersections, it is necessary to introduce additional topological structures. This property of spatial objects affects also the algebraic operations across representations that were discussed before. A good discussion of operation robustness can be found in [Hoff89, Ch. 4]. Furthermore, the management of uncertain information is a major issue in current artificial intelligence research [Bouc88]. At this point it is not clear which of these techniques could be applied to spatial data as well.

There has been a trend in spatial database research towards so-called scaleless maps, that is, digital maps where each entity is represented only at the largest available scale to minimize redundancy.

Given a query, one first selects the qualifying objects via some access method before scaling them down to the specified target scale. In order to evaluate such queries more efficiently, it would be advantageous to have access methods that take the target scale into account from the beginning. Ideally, there should be a positive correlation between the target scale on the one hand and the response time on the other hand for a given query, that is, one could receive a coarse (small-scale) answer to a query rather quickly, whereas it would take longer to receive a more accurate response. This approach has an effect similar to filters, and has only recently been studied [Beck90].

4. Conclusions

There is a steadily increasing demand for spatial data management capabilities, and a growing volume of spatial data. The techniques developed for spatial data management have been driven by the applications and are, therefore, often case-specific. The database community can contribute to the generalization of these techniques. So far it has concentrated on spatial access methods and on providing a spatial veneer on top of existing query languages. For database technology to have more impact on spatial data management, we need to address the problems of representation-independent spatial operations, their mapping into representation-specific operations, cross-representational algebras, and optimization of spatial queries. These issues have to be addressed if we want to exploit spatial data repositories fully, and eventually integrate existing spatial repositories.

Acknowledgements

We gratefully acknowledge interesting discussions with Renato Barrera, Frank Manola, Jack Orenstein, and Peter Widmayer.

References

[Barr81] R.Barrera, A.Buchmann, "Schema Definition and Query Language for a Geographical Database System", IEEE Trans. on Computer Architecture: Pattern Analysis and Image Database Management, Nov. 1981.

[Beck90] B. Becker, P. Widmayer, "Spatial Priority Search: An Access Technique for Scaleless Maps," Manuscript, University of Freiburg, 1990.

[Bouc88] B. Bouchon, L. Saitta, R. Yager (eds.), "Uncertainty and Intelligent Systems," Springer Lecture Notes in Computer Science, 1988.

[Brol89] J. Brolio, B.A.Draper, J.R.Beveridge, A.R.Hanson, "ISR: A Database for Symbolic Processing in Computer Vision", IEEE Computer, 22, 12, Dec. 1989.

[Buch90] A. Buchmann, O. Gunther, T. R. Smith, Y.-F. Wang (eds.), "Design and Implementation of Large Spatial Databases," Proceedings SSD'89, Lecture Notes in Computer Science No. 409, Springer-Verlag, 1990.

[Chan80] N.S.Chang, K.S.Fu, "Query-by-Pictorial-Example", IEEE Trans. on Software Engineering, 6, 6, Nov. 1980.

[Ditt88] K. R. Dittrich (ed.), "Advances in object-oriented database systems," Lecture Notes in Computer Science No. 334, Springer-Verlag, 1988.

[Egen89] M.Egenhofer, "Spatial Query Languages", PhD Thesis, U. of Maine, Orono, April 1989.

[Free90] M. Freeston, "A well-behaved structure for the storage of geometric objects," in [Buch90].

[Go75] A.Go et al, "An Approach to implementing a Geo-Data System", TR ERL-M529, University of California, Berk;ey, June 1975.

[Good89] A.M.Goodman, R.M.Haralick, L.G.Shapiro, "Knowledge-Based Computer Vision," IEEE Computer, 22, 12, Dec. 1989.

[Gree89] D. Greene, "An implementation and performance analysis of spatial data access methods," Proc. IEEE 5th Int. Conf. on Data Engineering (Los Angeles), 606-615, 1989.

[Gunt88] O. Gunther, "Efficient Structures for Geometric Data Management," Lecture Notes in Computer Science No. 337, Springer-Verlag, 1988.

[Gunt89] O. Gunther, "The design of the cell tree: an object-oriented index structure for geo-metric databases," Proc. IEEE 5th Int. Conf. on Data Engineering (Los Angeles), 598-605, 1989.

[Gunt90] O. Gunther and J. Bilmes, "Tree-based access methods for spatial databases: implementation and performance evaluation." To appear in IEEE Transactions on Knowledge and Data Engineering, 1990.

[Gutt84] A. Guttman, "R-trees: Dynamic index structure for spatial searching," Proc. of ACM SIGMOD Conference on Management of Data, Boston, Ma., 1984.
[Herr88] J. Herring et al, "Extensions to the SQL Language to Support Spatial Analysis in a Topological Data Base", in GIS/LIS'88, San Antonio, TX, Nov. 1988.
[Hoff89] C.M. Hoffmann, "Geometric and Solid Modelling: An Introduction", Chap 4., Morgan Kaufmann Publishers, Inc., 1989.
[Iyen88] S.S.Iyengar, R.I.Kashyap, (eds.) special issue on Image Databases, IEEE Trans. on Software Engineering, 14, 5, May 1988.
[Kemp87] A. Kemper and M. Wallrath, "An analysis of geometric modelling in database systems," ACM Comp. Surveys 19, 1, pp. 47-91, 1987.
[Krie90] H. P. Kriegel et al., "A performance comparison of point and spatial access methods," in [Buch90].
[Luge89] G. F. Luger, and W. A. Stubblefield, "Artificial intelligence and the design of expert systems," Benjamin/Cummings, 1989.
[Mano86] F. Manola, J.A.Orenstein, "Toward a General Spatial Data Model for an Object-Orienetd DBMS", Proc. 12th Intl. Conf. on Very Large Data Bases, Kyoto, 1986.
[Mant83] M.Mantyla, M.Tamminen, "Localized set operations for solid modeling", Computer Graphics, 17, 3, 1983, (279-288).
[McMa87] R. B. McMaster, "Automated Line Generalization," Cartographica 24, 2, pp. 74-111, 1987.
[McMa88] R. B. McMaster, "Cartographic Generalization in a Digital Environment: A Framework for Implementation in Geographic Information System," Proc. GIS/LIS Conference, pp. 240-255, 1988.
[Moha88] L.Mohan, R.L.Kashyap, "An Object-Oriented Knowledge Representation for Spatial Information", in [Iyen88]
[Moor85] S.Moorehouse, "ARC/INFO: A Geo-Relational Model for Spatial Information", Proc. Seventh Intl. Symp. on Computer Assisted Cartography, ACSM.
[Nagy79] G.Nagy, S.Wagle, "Geographic Data Processing", ACM Computing Surveys, 11,2, June 1979.
[Niev84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The grid file: an adaptable, symmetric multikey file structure," ACM Trans. on Database Systems 9, 1, pp. 38-71, 1984.
[Niev89] J. Nievergelt, "7+-2 criteria for assessing and comparing spatial data structures", in [Buch90].
[NSF90] M. Brodie et al., "Database Systems: Achievements and Opportunities," Report of the NSF Invitational Workshop on Future Directions in DBMS Research, June 1990.
[Ooi88] B. C. Ooi, "Efficient query processing in a geographic information system," Ph. D. Dissertation, Dep. of Computer Science, Monash University, 1988.
[Oost89] P. v. Oosterom and J. v. d. Bos, "An object-oriented approach to the design of geographic information systems," Computers and Graphics, Vol. 13, No. 4, 1989.
[Oren86] J. A. Orenstein, "Query processing in an object-oriented database system," Proc. of ACM SIGMOD Conference on Management of Data, 1986.
[Oren89] J. A. Orenstein, "Redundancy in spatial databases," Proc. of ACM SIGMOD Conference on Management of Data, 1989.
[Piza89] A.Pizano, A.Klinger, A.Cardenas, "Specification of Spatial Integrity Constraints in Pictorial Databases", IEEE Computer, 22, 12, Dec. 1989.
[Requ80] A. A. G. Requicha, "Representations for rigid solids: theory, methods, and systems," ACM Computing Surveys 12,4, 1980.
[Rous88] N.Roussopoulos, C.Faloutsos, T.Sellis, "An Efficient Pictorial Database System for PSQL", in [Iyen88]
[Same89] H.Samet, "The design and analysis of spatial data structures," Addison-Wesley, 1989.
[Sche82] H.J. Schek, P. Pistor, "Data Structures for an Integrated Data Base Management and Information Retrieval System", Proc. 8th Intl. Conf. on Very Large Data Bases, Mexico City, 1982.
[Seeg88] B. Seeger and H. P. Kriegel, "Techniques for design and implementation of efficient spatial access methods," Proc 14th Int. Conf. on Very Large Databases (Los Angeles), 1988.
[Six88] H.-W. Six and P. Widmayer, "Spatial searching in geometric databases," Proc. IEEE 4th Int. Conf. on Data Engineering (Los Angeles), 496-503, 1988.
[Smit87] T. R. Smith, D. Peuquet, S. Menon, and P. Agarwal, "KBGIS-II - A Knowledge-Based Geographical Information System," Int. J. Geographical Information Systems, 1, 2, 1987.
[Smit90] T. R. Smith and P. Gao, "Experimental Performance Evaluations on Spatial Access Methods," Proc. 4th Int. Symp. on Spatial Data Handling, Zurich 1990.
[Ston84] M. Stonebraker, E. Anderson, E. Hanson, and B. Rubenstein, QUEL as a Data Type, Proc. of ACM SIGMOD Conference on Management of Data, June 1984.

Database Security

Teresa F. Lunt[1]
Computer Science Laboratory
SRI International
Menlo Park, California 94025

Eduardo B. Fernandez
Dept. of Computer Engineering
Florida Atlantic University
Boca Raton, Florida 33431

1 Introduction

Computer security is concerned with the ability of a computer system to enforce a security policy governing the disclosure, modification, or destruction of information. The security policy may be specific to the organization, or may be generic. For example, the DoD *mandatory security* (or multilevel security) policies restrict access to classified information to cleared personnel. *Discretionary security* policies, on the other hand, define access restrictions based on the identity of users (or groups), the type of access (e.g., select, update, insert, delete), the specific object being accessed, and perhaps other factors (time of day, which application program is being used, etc.). Different types of users (system managers, database administrators, and ordinary users) may have different access rights to the data in the system. The access controls commonly found in most database systems are examples of discretionary access controls.

This paper discusses discretionary security and mandatory security for database systems. We outline the current state of research in database security and briefly discuss some open research issues.

2 Discretionary Security

Discretionary Access Control (DAC) models can be represented by the access matrix model developed by Lampson in 1971 [23] and further refined by Graham and Denning [13]. This model defines an access matrix in which the rows represent *subjects* (users, processes), the columns represent *objects* (files, records, programs, subsystems, etc.), and the intersection of a row and a column contains the access types that the subject has authorization for with respect to the object. This model was extended with predicates and other components to make it more useful for database systems [4, 11, 18]. For example, predicates can represent content-dependent access control. Detailed discussions of the meaning of the model components can be found in Chapter 7 of [10] and in [9]. Implementation aspects are discussed in Chapter 10 of [10]. Harrison, Ruzzo, and Ullman [16, 17] showed that the general safety problem for this model is undecidable.

From 1970 through 1975, there was a good deal of interest in these models, in particular in their application to relational databases [11, 14, 18]. Then, most of the database security research turned to multilevel models (see Section 3). The appearance of semantic and object-oriented databases

[1]Lunt's work was supported by the U. S. Air Force, RADC, under contract F30602-89-C-0158.

has created a renewed interest in DAC models. These database models are much richer and more complex than the relational model, and aspects such as inheritance, semantic associations, and composite data structures allow the expression of flexible and arbitrarily specific discretionary policies. The research issues currently under investigation include:

a. A fundamental property of object-oriented systems is inheritance. This is often complemented with semantic associations such as aggregation or relationship. A class object may have many descendants or instances. If a user is authorized to access a class, should that user also be authorized to access all of its instances? Should this inherited authorization also include authorization for attributes that do not exist in the parent class?
b. For practical reasons it is necessary to define groups of users with similar authorizations. If authorizations can be defined both for individuals and for groups, or if a user can belong to more than one group, how should we determine the user's effective authorizations? If groups can be structured in hierarchies or partial orders, how does one define the effective authorizations of a given group?
c. Inheritance of authorization brings along the need to override inherited authorizations. One way to do this is to use negative authorization, which is given a higher priority than positive authorization. Another way is to use explicit authorization to override implicit inherited authorization. Related issues are the combination of access predicates defined a different levels in the object hierarchy and how to combine the effects of multiple inherited rules.
d. Should users own data or does all data belong to the institution? If we adopt the latter approach, then users with special authorizations (adminstrators) should handle the definition of authorization rules. Even in the latter case, should we still allow private subdatabases?
e. What framework can we use to ensure that new authorization rules satisfy institution policies?
f. How can we decentralize the functions of a security administrator, necessary for distributed environments?

Recent work on discretionary models include (the labels in parenthesis refer to which of the aspects above they have especially contributed):

- W. Kim and his group at MCC developed a detailed formal model for object-oriented systems using ORION as an illustration (the ORION object-oriented database system was also developed at MCC) [36]. Implied (inherited) authorization are obtained through the object hierarchy, i.e., from the class, to its components, to its instances. They use the concept of weak and strong authorizations, where strong authorizations can override weak authorizations (overriding is performed using negative authorization rules). (a)(c)
- U. Kelter at the University of Hagen, W. Germany, is developing models for distributed structurally object-oriented database systems [22]. In this type of database there is a hierarchy of nested, overlapping, complex objects describing typically a CAD or CASE design environment. Complex objects are the units of access control, and authorization conflicts may arise if complex objects share components. Group authorizations are inherited, that is, a lower group can inherit authorizations from a higher-level group, rather than through the data hierarchies. Explicit denial of authorzation (negative authorization) is handled using a four-valued logic. The first authorization model for this type of database was developed by K. Dittrich and his group at the University of Karlsruhe, W. Germany, for the DAMOKLES database system [7]. (a)(b)(c)

- E. B. Fernandez and his group at Florida Atlantic University, Boca Raton, Florida, developed an authorization model for object-oriented semantic databases specially tailored to OSAM*, a CAD/CAM database system being implemented at the University of Florida [24, 37]. This authorization model can also be applied to other data models. The model consists of a set of policies that define authorization inheritance through the data hierarchies, authorization evaluation algorithms (since the rules can be scattered through the hierarchy, tree searches are necessary), and an adminstrative structure that considers policies for delegation and revocation of administrative authorizations as well as the effect of database schema changes. Recent work also considers how to handle negative authorizations and the use of predicates [15]. (a)(c)(d)(f)
- J. D. Moffet and M. Sloman at the Imperial College of London use a model in which administration is separated from the use of the data [34]. They also include the concept of ownership, where owners are those entities in an enterprise, such as managers, that control the operations of the enterprise and that can grant administrative authorizations to security administrators. Recent work from this group considers decentralization aspects of their model [35]. (d)(e)(f)
- T. F. Lunt at SRI International, Menlo Park, California, has developed a model of discretionary security for SeaView, a secure multilevel database system [25, 26]. She is also modeling a discretionary access control mechanism that allows the implementation of arbitrary access control policies. The intent is to allow users to implement a discretionary access contol policy that is tailored to their application, rather than having to work around a specific policy that is wired into the computer system. (b)(c)

3 Mandatory Security

A multilevel database system supports data having different classifications or *access classes* and users having different clearances. In the most general case, the ability to individually classify atomic facts in a database is required. In the relational model, this means that data is classified at the level of individual data elements. Special cases of multilevel relations may be classified at the attribute level (i.e., all the data associated with a particular attribute has the same classification); at the row level (i.e., every tuple has a single classification); or at the relation level (i.e., all the data in the relation has the same classification).

The mandatory access control requirements are formalized by two rules, the first of which protects data from unauthorized disclosure, and the second of which protects data from contamination:

1. A subject S is not allowed to read data of access class c unless $class(S) \geq c$, and
2. A subject S is not allowed to write data of access class c unless $class(S) \leq c$.

In the above rules, a *subject* is a process acting on a user's behalf; a process has a clearance level derived from that of the user.

Mandatory security means that a multilevel relation will appear differently to users with different clearances, because not all data are authorized to all users. Other requirements include the ability to derive classification labels for derived data (as in views, for example).

We represent a multilevel relation R by a schema $R(A_1, C_1, \ldots, A_n, C_n)$, where each data attribute A_i has a corresponding *classification attribute* C_i. Figure 1 illustrates a multilevel relation

with three data attributes. In the figure, the label "S" means the data is classified SECRET; the label "TS" means TOP SECRET.

$A1$	$C1$	$A2$	$C2$	$A3$	$C3$
mad	S	17	S	x	S
foo	S	34	S	w	TS
ark	TS	5	TS	y	TS

Figure 1: Multilevel Relation R

The name of a relation R also has a classification, which we denote by $class(R)$. $Class(R)$ must be at least as low as the classification of any data contained in the relation. A relation R can be accessed by any user S where $class(S) \geq class(R)$. However, S can see only data that S is cleared to see. Figure 2 shows a SECRET view of the multilevel relation of Figure 1.

$A1$	$C1$	$A2$	$C2$	$A3$	$C3$
mad	S	17	S	x	S
foo	S	34	S	null	S

Figure 2: SECRET Relation Instance

Multilevel security affects the data model because not all data are visible to all users. One effect involves two basic properties: entity integrity and referential integrity. Another is polyinstantiation, which we describe shortly.

Entity integrity states that no tuple can have null values for any primary key attribute. In the multilevel case, within a tuple all the primary key elements must have the same access class. Otherwise, a low user would see null values for some of the key elements. For example, if the first two data attributes form the primary key, the tuple (10, S, X, TS, 17, TS) has primary key elements with different access classes. A SECRET user's view of the tuple is (10, S, null, S, null, S), which violates entity integrity because part of the key appears to be null.

The key class must also be at least as low as the access classes of all other elements in the tuple; otherwise a low user would see nulls for the primary key. For example, if the first two data attributes form the primary key, the tuple (20, TS, Y, TS, 34, S) has a key class greater than the class of the remaining data element. A SECRET user's view of the tuple is (null, S, null, S, 34, S) which violates entity integrity because the key appears to be null.

Referential integrity states that no tuple in a relation can refer to another tuple unless the referenced tuple exists. In a multilevel database, this means that a tuple of a low access class cannot reference a tuple of a high access class because the referenced tuple would appear to be nonexistent to users with low clearances.

Multilevel security has a further unexpected but unavoidable effect, which we call *polyinstantiation* [29]. To illustrate, consider what happens when a user inserts a tuple that has the same primary key value as an existing but invisible (because it has a high classification) tuple. Notifying the user of the conflict would tell the user the value of high information. Thus we add a *second*

tuple to the relation with the same primary key but different access class; we call these *polyinstantiated tuples.* For example, if a SECRET user adds a tuple with primary key "ark" to the relation instance shown in Figure 2, then the outcome, as seen by a TOP-SECRET user, is as shown in Figure 3.

$A1$	$C1$	$A2$	$C2$	$A3$	$C3$
mad	S	17	S	x	S
foo	S	34	S	w	TS
ark	TS	5	TS	y	TS
ark	S	22	S	z	S

Figure 3: A Polyinstantiated Tuple

Now consider what happens if a user updates what appears to be a null element in a tuple, but which actually hides data with a higher access class. For example, if a SECRET user now replaces the perceived null value for tuple "foo," attribute $A3$ (see Figure 2) with the value "u," the outcome, as seen by a TOP-SECRET user, is as shown in Figure 4. We call these *polyinstantiated elements.*

$A1$	$C1$	$A2$	$C2$	$A3$	$C3$
mad	S	17	S	x	S
foo	S	34	S	w	TS
foo	S	34	S	u	S
ark	TS	5	TS	y	TS

Figure 4: A Polyinstantiated Element

To implement mandatory security, we assign security levels to processes, derived from the clearance of the user. Traditional practice is to segregate those functions enforcing mandatory security in a *security kernel* or *reference monitor.* The reference monitor mediates each reference to an object by any process, allowing or denying the access based on a label comparison. The reference monitor must be tamperproof; it must be invoked for *every* reference; and it must be small enough to be subject to complete analysis and test. When assurance is important, the reference monitor is formally verified through a formal mathematical proof (that may be carried out using automated tools) that it correctly enforces the mandatory security policy. The DoD has developed evaluation criteria for trusted computer systems [6] that incorporate the concept of reference monitor and include requirements for assurance as well as many other security requirements. The most stringent of the evaluation classes is called Class A1.

There have been three efforts to design Class A1 relational database systems.

- SeaView, being developed at SRI by T. Lunt et al., provides element-level labeling and derives labels for derived data. It includes a multilevel query language called MSQL for defining and manipulating multilevel data. Its design has been partially verified using EHDM, a formal verification system developed at SRI [1, 2, 3, 5, 39, 40, 41, 42]. SeaView uses a conventional relational engine and a commercially available reference monitor [29, 31, 32].

- A group at Secure Computing Technology Corporation (SCTC) has produced a design for LOCK Data Views (LDV), a relational system that allows an application to specify rules for how incoming and outgoing data are to be labeled. LDV relies on special-purpose security hardware being developed at SCTC [38].
- ASD is a prototype developed at TRW. It provides row-level labeling [19].

In addition, several vendors, including Oracle Corporation and Sybase, have announced or released products designed to meet some of the DoD criteria.

Ongoing research is focused on the following areas:

- *Defining an operational semantics for multilevel database operations* [30]. There is a need to define the intended semantics for the basic data manipulation operations insert, update, and delete for a multilevel database system with classification at the element level. This will effect, for example, how much polyinstantiation can occur as a result of an update operation, and which polyinstantiated instances are deleted in a delete operation.
- *Extending multilevel security to other data models* [12, 20, 21, 28, 33]. Preliminary work has begun to develop multilevel security models for object-oriented databases, entity-relationship databases, and knowledge-based systems.
- *Extending multilevel security to distributed database systems* [8]. Researchers are beginning to address such issues as architectures for secure distributed database systems and balancing the security, consistency, and availability of distributed data.
- *Solving inference problems* [27]. The inference problem arises when a collection of data is more sensitive than the individual pieces (this is sometimes called the *aggregation problem*, or when a collection of data can be used to partially infer data of a higher sensitivity. In some cases, even learning of the existence of the information may be unacceptable. This problem commonly arises when the individual data items may be classified low, but the relationship among the data items is considered to be highly sensitive. Data design strategies have been proposed to segregate the sensitive associations and to give those sensitive associations high classifications while giving the individual data items low classifications. This has the result that low data is available to users with low clearances while users with high clearances can make sensitive associations among the data.

References

[1] EHDM *Specification and Verification System Version 4.1—User's Guide.* Computer Science Laboratory, SRI International, Menlo Park, CA 94025, November 1988. See [3] for the updates to Version 5.1.

[2] EHDM *Specification and Verification System Version 5.0—Description of the* EHDM *Specification Language.* Computer Science Laboratory, SRI International, Menlo Park, CA 94025, January 1990. See [3] for the updates to Version 5.1.

[3] EHDM *Specification and Verification System—Version 5.1 Supplement to User's and Language Manuals.* Computer Science Laboratory, SRI International, Menlo Park, CA 94025, April 1990.

[4] R. W. Conway, W. L. Maxwell, and H. L. Morgan. On the implementation of security measures in information systems. *Communications of the ACM*, 15(4), April 1972.

[5] J. S. Crow, R. Lee, J. M. Rushby, F. W. von Henke, and R. A. Whitehurst. EHDM verification environment: An overview. In *Proceedings of the 11th National Computer Security Conference*, October 1988.

[6] *Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD.* Department of Defense, December 1985.

[7] K. R. Dittrich, M. Hartig, and H. Pfefferle. Discretionary access control in structurally object-oriented database systems. In *Proceedings of the 2nd IFIP WG11.3 Workshop on Database Security*, October 1988.

[8] A. Downing, I. Greenberg, and T. F. Lunt. Issues in distributed database security. In *Proceedings of the 5th Aerospace Computer Security Conference*, December 1989.

[9] D. D. Downs, J. R. Rub, K. C. Kung, and C. S. Jordan. Issues in discretionary access control. In *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, 1985.

[10] E. B. Fernandez, R. C. Summers, and C. Wood. *Database Security and Integrity.* Addison-Wesley, Reading, Massachusetts, 1981.

[11] E. B. Fernandez, R. C. Summers, and C. D. Coleman. An authorization model for a shared data base. In *Proceedings of the 1975 ACM SIGMOD International Conference*, 1975.

[12] T. D. Garvey and T. F. Lunt. Multilevel security for knowledge-based systems. In *Proceedings the EISS Workshop on Database Security*, European Institute for System Security, Karlsruhe, W. Germany, April 1990.

[13] G. S. Graham and P. J. Denning. Protection—principles and practice. In *Proceedings of the Spring Joint Computer Conference*, volume 40, Montvale, New Jersey, 1972. AFIPS Press.

[14] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3), September 1976.

[15] E. Gudes, H. Song, and E. B. Fernandez. Evaluation of negative and predicate-based authorization in object-oriented databases. *Proceedings of the 4th IFIP WG11.3 Workshop on Database Security*, Halifax, UK, Sept. 1990.

[16] M. A. Harrison. Theoretical issues concerning protection in operating systems. *Advances in Computers*, 24, 1985.

[17] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8), August 1976.

[18] H. R. Hartson and D. K. Hsiao. A semantic model for database protection languages. In *Proceedings of the Second International Conference on VLDB.* North-Holland, 1976.

[19] T. H. Hinke, C. Garvey, N. Jensen, J. Wilson, and A. Wu. A1 secure DBMS design. In *Proceedings of the 11th National Computer Security Conference - Appendix*, October 1988.

[20] S. Jajodia and B. Kogan. Integrating an object-oriented data model with multilevel security. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, May 1990.

[21] T. F. Keefe, W. T. Tsai, and M. B. Thuraisingham. SODA: A secure object-oriented database system. Technical report, TR89-12, University of Minnesota, Computer Science Department, 1989.

[22] U. Kelter. Group-oriented discretionary access controls for distributed structurally object-oriented database systems. Informatics Report N 93, Fern Universitat Hagen, Hagen, Germany, 1990.

[23] B. W. Lampson. Protection. In *Proceedings of the 5th Princeton Symposium on Information Science and Systems*, March 1971. Reprinted in ACM Operating Systems Review, Vol. 8 (1), January 1974.

[24] M. M. Larrondo-Petrie, E. Gudes, H. Song, and E. B. Fernandez. Security policies in object-oriented databases. In *Database Security III: Status and Prospects, D.L. Spooner and C. Landwehr (Eds.).* Elsevier, 1990.

[25] T. F. Lunt. Access control policies for database systems. In C. E. Landwehr, editor, *Database Security II: Status and Prospects.* North Holland, 1989.

[26] T. F. Lunt. Access control policies: Some unanswered questions. *Computers and Security*, February 1989.

[27] T. F. Lunt. Aggregation and inference: Facts and fallacies. In *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy*, May 1989.

[28] T. F. Lunt. Multilevel security for object-oriented database systems. In D. L. Spooner and C. E. Landwehr, editors, *Database Security III: Status and Prospects.* Elsevier, 1990.

[29] T. F. Lunt, D. E. Denning, R. R. Schell, W. R. Shockley, and M. Heckman. The SeaView security model. *IEEE Transactions on Software Engineering*, June 1990.

[30] T. F. Lunt and D. Hsieh. Update semantics for a multilevel relational database system. In *Proceedings of the 4th IFIP WG11.3 Workshop on Database Security*, Halifax, England, September 1990.

[31] T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren. A near-term design for the SeaView multilevel database system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[32] T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren. Toward a multilevel relational data language. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, December 1988.

[33] J. K. Millen and T. F. Lunt. Secure knowledge-based systems. Technical Report SRI-CSL-90-04, Computer Science Laboratory, SRI International, Menlo Park, California, August 1989.

[34] J. D. Moffet and M. S. Sloman. The source of authority for commercial access control. *Computer*, 21(2), February 1988.

[35] J. D. Moffet and M. S. Sloman. Delegation of authority. Domino Rept. B1/IC/4, Dept. of Computing, Imperial College of Science and Technology, London, July 1990.

[36] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next generation database systems. Technical Report ACA-ST-395-88, MCC, November 1988.

[37] H. Song, E. B. Fernandez, and E. Gudes. Administrative authorization in object-oriented databases. In *Proceedings of the EISS Workshop on Database Security*, European Institute for System Security, Karlsruhe, W. Germany, April 1990.

[38] P. D. Stachour and B. Thuraisingham. Design of ldv: A multilevel secure relational database management system. *IEEE Transactions on Knowledge and Data Engineering*, 2:2, June 1990.

[39] F. von Henke and J. Rushby. *Introduction to* EHDM. Computer Science Laboratory, SRI International, Menlo Park, CA 94025, September 28, 1988.

[40] F. von Henke, N. Shankar, and J. Rushby. *Formal Semantics of* EHDM. Computer Science Laboratory, SRI International, Menlo Park, CA 94025, January 1990. This document describes EHDM Version 5.0, see [3] for informal descriptions of the changes in Version 5.1.

[41] R. A. Whitehurst and T. F. Lunt. The SeaView verification. In *Proceedings of the Second Workshop on the Foundations of Computer Security*, June 1989.

[42] R. A. Whitehurst and T. F. Lunt. The SeaView verification effort. In *Proceedings of the 12th National Computer Security Conference*, October 1989.

Real-Time Database Systems: A New Challenge

Sang H. Son

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903

(804) 982-2205
son@cs.virginia.edu

ABSTRACT

The design and implementation of real-time database systems presents many new and challenging problems. Compared with traditional databases, real-time database systems have a distinct feature: they must satisfy timing constraints associated with transactions. Transactions in real-time database systems should be scheduled considering both data consistency and timing constraints. In addition, a real-time database system must adapt to changes in the operating environment and guarantee the completion of critical transactions. In this paper we address the issues associated with transaction modeling, scheduling and concurrency control, consistency, and predictability requirements for real-time database systems.

1. Introduction

Real-time database systems (RTDBS) are database systems where transactions have timing constraints such as *deadlines*. The correctness of the system depends not only on the logical results but also on the time within which the results are produced. In RTDBS, transactions must be scheduled in such a way that they can be completed before their corresponding deadlines expire. For example, both the update and query on the tracking data for a missile must be processed within given deadlines.

RTDBS are becoming increasingly important in a wide range of applications, such as aerospace and weapon systems, computer integrated manufacturing, robotics, nuclear power plants, and traffic control systems. In recent real-time computing workshops sponsored by the Office of Naval Research [IEEE90, ONR89], researchers pointed to the need for basic research in database systems that satisfy timing constraints in collecting, updating, and retrieving shared data, since traditional data models and databases are not adequate for time-critical applications. Very few conventional database systems allow users to specify or ensure timing constraints. Interest in the time-critical application domain is growing also in database community. Recently, a number of research results have appeared in the literature [Abb88, Abb89, Buc89, Kor90, Lin89, Lin90, Raj89, Sha88, Sha90, Son88b, Son89, Son90].

It is useful to categorize transactions in real-time database systems as *hard* and *soft* transactions [Son88b]. We define hard real-time transactions as those transactions whose timing constraints must be guaranteed. Missing deadlines of this type of transaction may result in catastrophic consequences. In contrast, soft real-time transactions have timing constraints, but there may still be some justification in completing the transactions after their deadlines. Catastrophic consequences do not result if soft real-time transactions miss their deadlines. Soft real-time transactions are scheduled taking into account their timing requirements, but they are not guaranteed to make their deadlines. There are many real-time systems that need database support for both types of transactions.

Conventional database systems are typically not used in real-time applications due to two inadequacies: poor performance and lack of predictability. In conventional database systems, transaction processing requires access to a database stored on secondary storage; thus transaction response time is limited by disk access delays, which can be in the order of milliseconds. Still these databases are fast enough for traditional applications in which a response

This work was supported in part by ONR contract # N00014-88-K-0245, by CIT contract # CIT-INF-90-011, and by IBM Federal Systems Division.

time of a few seconds is often acceptable to human users. However, those systems may not be able to provide a response fast enough for high-performance real-time applications. One approach to achieve high performance is to replace slow devices (e.g., a disk) by a high speed version (e.g., a large RAM). Another alternative is to use application-specific knowledge to increase the degree of concurrency. For example, by exploiting the semantic information associated with transactions and data, we may use the notion of correctness different from serializability. As observed by Bernstein [Bern87], serializability may be too strong as a correctness criterion for concurrency control in database systems with timing constraints, because of the limitation on concurrency. If necessary, data consistency might be compromised to satisfy timing constraints.

In terms of predictability, current database systems do not schedule transactions to meet response-time requirements and they commonly lock data to assure consistency. Locks and time-driven scheduling are basically incompatible. Low priority transactions may block higher priority transactions, leading to timing requirement failures. Consequently, the requirements and design objectives of real-time database systems differ widely from those of conventional database systems. New techniques are necessary to manage database consistency. They should be compatible with time-driven scheduling, and meet the system response times and temporal consistency requirements. A natural question is how a conventional database system must be modified so that its performance and predictability can be acceptable for real-time applications.

In RTDBS, the timeliness of a transaction is usually combined with its criticality to calculate the priority of the transaction. Therefore, proper management of priorities and conflict resolution in real-time transaction scheduling are essential for predictability and responsiveness of RTDBS.

While the theories of concurrency control in database systems and real-time task scheduling have both advanced, little attention has been paid to the interaction between concurrency control protocols and real-time scheduling algorithms [Stan88]. In database concurrency control, meeting the deadline is typically not addressed. The objective is to provide a high degree of concurrency and thus faster average response time without violating data consistency. In real-time scheduling, on the other hand, it is customary to assume that tasks are independent, or that the time spent synchronizing their access to shared data is negligible compared with execution time. The objective here is to maximize resources, such as CPU utilization, subject to meeting timing constraints. Data consistency is not a consideration in real-time scheduling, and hence the problem of guaranteeing the consistency of shared data is ignored. In addition, conventional real-time systems assume advance knowledge of the resource and data requirements of programs.

One of the challenges of RTDBS is the creation of a unified theory for real-time scheduling and concurrency control protocols that maximizes both concurrency and resource utilization subject to three constraints: data consistency, transaction correctness, and transaction deadlines. Several recent projects have integrated real-time constraints with database technology to facilitate efficient and correct management of timing constraints in RTDBS [Buc89, Son88b, Son90]. There are several difficulties in achieving the integration. A database access operation, for example, takes a highly variable amount of time depending on whether disk I/O, logging, buffering, etc. are required. Furthermore, concurrency control may cause aborts or delays of indeterminate length.

The design and implementation of a real-time database system presents many new and challenging problems: What is an appropriate model for real-time transactions and data? What are the language constructs that can be used to specify real-time constraints? What mechanisms are needed for describing and evaluating triggers? What are the measures of system predictability? How are transactions scheduled? What is the effect of real-time constraints on concurrency control? In this paper we address some of these issues and review current approaches to the design of RTDBS.

2. Research Issues and Approaches

2.1. Scheduling and Concurrency Control

The goal of scheduling in RTDBS is twofold: to meet timing constraints and to enforce data consistency. Real-time task scheduling methods can be extended for real-time transaction scheduling, yet concurrency control protocols are still needed for operation scheduling to maintain data consistency. However, the integration of the two mechanisms in RTDBS is not straightforward. The general approach is to utilize existing concurrency control protocols, especially two-phase locking (2PL) [Bern87], and to apply time-critical transaction scheduling methods that favor more urgent transactions [Abb88, Sha88, Son89b]. Such approaches have the inherent disadvantage of being limited by the concurrency control protocol upon which they depend, since all existing concurrency control protocols synchronize concurrent data access of transactions by a combination of two measures: blocking and roll-backs

of transactions. Both are barriers to meeting time-critical schedules. Concurrency control protocols induce a serialization order among conflicting transactions. In non-real-time concurrency control protocols, timing constraints are not a factor in the construction of this order. This is obviously a drawback for RTDBS. For example, with the 2PL protocol, the serialization order is dynamically constructed and corresponds to the order in which conflicting transactions access shared data. In other words, the serialization order is bound to the past execution history with no flexibility. When a transaction T_H with a higher priority requests an exclusive lock which is being held by another transaction, T_L, with a lower priority, the only choices are either aborting T_L or letting T_H wait for T_L. Neither choice is satisfactory. The conservative 2PL uses blocking, but in RTDBS, blocking may cause *priority inversion*. Priority inversion is said to occur when a high priority transaction is blocked by lower priority transactions [Sha88]. The alternative is to abort low priority transactions when a priority inversion occurs. This wastes the work done by the aborted transactions and in turn also has a negative effect on time-critical scheduling. Various scheduling policies with lock-based concurrency control mechanisms for real-time transactions have been investigated in [Abb88, Abb89].

The priority ceiling protocol, which was initially developed as a task scheduling protocol for real-time operating systems, has been extended to RTDBS [Sha90]. It is based on 2PL and employs only blocking, but not roll-back, to solve conflicts. This is a conservative approach. For conventional database systems, it has been shown that optimal performance may be achieved by compromising blocking and roll-back [Yu90]. For RTDBS, we may expect similar results. Aborting a few low priority transactions and restarting them later may allow high priority transactions to meet their deadlines, resulting in improved system performance. A drawback of the priority ceiling protocol is that it requires knowledge of all transactions that will be executed in the future. This is too harsh a condition for most database systems to satisfy.

For a concurrency control protocol to accommodate the timeliness of transactions, the serialization order it produces should reflect the priority of transactions. An optimistic method [Bok87, Kung81] is one way of achieving this goal. Due to its validation phase conflict resolution, it can be ensured that eventually discarded transactions do not abort other transactions and transaction priorities are considered. Several concurrency control protocols based on optimistic approach have been proposed [Har90, Son90b, Lin90]. They incorporate priority-based conflict resolution mechanisms, such as *priority wait*, that makes low priority transactions wait for conflicting high priority transactions to complete. However, this approach of detecting conflicts during validation phase degrades system predictability. A transaction is detected as being late when it actually misses its deadline, since the transaction is only aborted in the validation phase. An integrated scheduler may solve this problem if a lock-based concurrency control protocol supports a mechanism to adjust dynamically the serialization order of active transactions.

An algorithm integrating priority-based locking and optimistic approach is proposed in [Lin90]. The goal is to execute transactions with higher priorities first so that high priority transactions are never blocked by uncommitted low priority transactions, while lower priority transactions may not have to be aborted despite conflicting operations. It is surprising that for RTDBS, such integrated approaches outperform lock-based pessimistic approaches over a wide range of system load and resource availability, since performance studies of concurrency control protocols for conventional database systems (e.g. [Agr87]) have concluded that locking protocols perform better than optimistic techniques. More theoretical as well as experimental studies are necessary in this area before we can draw definitive conclusions.

Another important problem that needs further study is a different notion of "correct execution" in transaction processing. Based on the argument that timing constraints may be more important than data consistency in RTDBS, attempts have been made to satisfy timing constraints by sacrificing database consistency temporarily to some degree [Lin89, Vrb88]. It is based on a new consistency model of real-time databases, in which maintaining *external data consistency* has priority over maintaining *internal data consistency*. Although in some applications weaker consistency is acceptable [Gar83], a general-purpose consistency criterion that is less stringent than serializability has not yet been proposed. The problem is that temporary inconsistencies may affect active transactions and so the commitment of these transactions may still need to be delayed until the inconsistencies are removed; otherwise even committed transactions may need to be rolled back. However, in real-time systems, some actions are not reversible.

The use of semantic information in transaction scheduling and multiversion data is often proposed for RTDBS applications [Liu88, Son88, Son90c, Song90]. Multiple versions are useful in situations that require the monitoring of data as values are changing with time. In such situations, the trends exhibited by the values of the data are used to trigger proper actions [Kor90]. Examples include falling values in stock-market trading and rising temperature of a furnace in a nuclear reactor. Another objective of using multiple versions is to increase the degree of concurrency and to reduce the possibility of transaction rejection by providing a succession of views of data.

There are several problems that must be solved in order to use multiple versions effectively. For example, the selection of old versions for a transaction must ensure the required consistency of the state seen by the transaction. In addition, the need to save old versions introduces a storage management problem.

2.2. Temporal Consistency

Often a significant portion of a real-time database is highly perishable in the sense that it has value to a mission only if used in time. In addition to deadlines, therefore, other kinds of timing constraints could be associated with data as well as transactions in RTDBS. For example, each sensor input could be indexed by the time at which it was taken. Once entered into the database, data may become out-of-date if it is not updated within a certain period of time. To quantify this notion of "age", data may be associated with a *valid interval* [Liu88, Song90]. Data outside its valid interval does not represent the current state. What occurs when a transaction attempts to access data outside its valid interval depends on the semantics of data and the particular system requirements.

A real-time transaction may include a temporal consistency requirement that specifies the validity of data values accessed by the transaction. While a deadline can be thought of as providing a time interval as a constraint in the future, temporal consistency specifies a temporal window as a constraint in the past. As long as the temporal consistency requirement of a transaction can be satisfied, the system must be able to provide an answer using available (may not be up-to-date) information. The answer may change as valid intervals change with time. In a distributed database system, sensor readings may not be reflected to the database at the same time, and may not be reflected consistently due to the delays in processing and communication. A temporal data model for RTDBS must therefore be able to accommodate the information that is partial and out-of-date. One of the aspects that distinguishes a temporal data model for RTDBS from that of conventional database systems is that values in RTDBS are not necessarily correct all the time, and hence the system must be cautious in interpreting data values.

2.3. System Modeling and Performance Evaluation

Most research in RTDBS assumes a set of transactions and associated deadlines. In such a database system model, it is the responsibility of the transaction manager to find a correct schedule that will ensure that the deadlines are met. An interesting alternative model of real-time transaction processing, in which deadlines are associated with consistency constraints, has been proposed [Kor90]. Instead of applying timing constraints directly to transactions, it applies timing constraints directly to states of the system. Timing constraints on the states of the system enforce similar timing constraints on transactions that are triggered by those states. For example, consider a RTDBS application in a manufacturing environment. Suppose that the state of the information maintained in the database indicates that the temperature in a furnace has fallen below a particular threshold value. This state of the system may necessitate the triggering of some actions that restore the temperature to a value above the threshold. The application may enforce a maximum period of time that the temperature is permitted to remain below the threshold, and that enforces a deadline on the actions that are triggered by the low value. Furthermore a selection must be made if several actions may be candidates for the restoration of the temperature.

This approach is based on a set of explicitly defined consistency constraints for the database. Each transaction ensures that upon completion, the database remains in a state that satisfies these consistency constraints. However, in addition to such transactions that maintain a correct database state, transactions may be invoked to record the effects of some external event that is generated outside the system. The ensuing change in the database state may render a consistency constraint invalid, and that constraint may need to be restored within a specific deadline. The system restores constraints by choosing one or more transactions from a pre-defined library of transactions. These transactions restore certain constraints but may invalidate other constraints. In the absence of further external events, the system must eventually return the entire database to a consistent state.

To be useful, any ideas or proposed techniques should be evaluated. There have been extensive studies on alternative queueing disciplines with lock-based concurrency control and scheduling of real-time transactions [Abb88, Abb89, Car89, Har90, Hua89, Hua90, Lin90, Son89b, Son90, Son90b]. While most of the results are based on simulation, some of techniques have been implemented on a testbed system [Hua89, Hua90]. These studies provide insight into many of the issues encountered in the design of RTDBS. However, the results reported in the literature are incomplete. For example, even though timing constraints and criticality are two important factors in specifying real-time transactions, the relation between the two factors and their combined effect with respect to system performance has not been addressed in depth. It is difficult to compare various scheduling algorithms without a proper benchmark that is representative of applications. Because research on RTDBS is still in its infancy, none of the known benchmarks addresses transaction processing in a real-time environment.

3. Research Directions

RTDBS of tomorrow will be large and complex. They will be distributed, operate in an adaptive manner in a highly dynamic environment, exhibit intelligent behavior, and be characterized as having catastrophic consequences if the logical or timing constraints of transactions are not met. Meeting the challenges imposed by these characteristics depends on a focused and coordinated research efforts in several areas listed below:

- development of modeling techniques for real-time transactions and databases to specify timing properties and temporal consistency in a precise manner. Validity of stored data and relationships between consistency constraints and timing constraints need to be specified clearly.

- development of methodologies to analyze timing properties of the system with a large number of interacting transactions and constraints. Given the timing properties of a RTDBS and a set of timing requirements expressed as assertions, the objective is to relate each assertion to the timing specification. If the assertion is a theorem derivable from the timing specification, then the system is safe with respect to the requirement denoted by the assertion, as long as the implementation is faithful to the timing specification. Otherwise, the system is inherently unsafe because the timing properties will cause the assertion to be violated.

- development of time-driven priority-based scheduling protocols and concurrency control protocols that can, in an integrated and dynamic fashion, manage complex transactions with resource and precedence constraints, manage resources (e.g., communication resources and I/O devices), and manage timing constraints of varying granularity. In particular, time-driven resource allocation policies and distributed transaction management protocols (e.g., timed atomic commit protocols) need to be developed to meet the real-time scheduling requirements. Since recovery by "undoing" operations may not be applicable in many circumstances, a form of forward recovery might be necessary.

- integration of operating system functions with data management in a highly integrated and cooperative, and fast and predictable manner. Since a database system must operate in the context of available operating system services, correct functioning and timing behavior of database management algorithms depends on the services of the underlying operating system. In many areas such as buffer management and consistency control, operating system facilities have to be duplicated by database systems because they are too slow or inappropriate. Research is needed to identify a set of efficient OS primitives required to support the database management protocols, especially addressing real-time constraints.

- integration of artificial intelligence techniques with data management protocols to provide the capability to reason about time-constrained processes and to control or schedule those processes, to adapt system behavior to dynamically changing timing constraints. A key consideration in triggered RTDBS is to determine the best available execution sequence when several choices are available with different timing constraints and precedence requirements.

- architecture support for fault-tolerance, for efficient data management, and for time-constrained communication. Due to advances in VLSI technology, it is possible to develop a new distributed architecture that is suitable for broader class of RTDBS applications. Important issues in this new architecture include interconnection topology, interprocess communications, and support of fault-tolerant database operations. It is essential to have hardware support for fast error detection, reconfiguration and recovery. In addition, new architectures need to support real-time scheduling algorithms.

REFERENCES

[Abb88] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Study," *VLDB Conference*, Sept. 1988, 1-12.

[Abb89] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data," *VLDB Conference*, Aug. 1989.

[Agr87] Argrawal, R. et al., "Concurrency Control Performance Modeling: Alternatives and Implications," *ACM Trans. on Database Systems*, Dec. 1987.

[Bern87] Bernstein, P., V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[Bok87] Boksenbaum, C. et al., "Concurrent Certifications by Intervals of Timestamps in Distributed Database Systems," *IEEE Trans. on Software Eng.*, vol. SE-13, no. 4, April 1987, pp 409-419.

[Buc89] Buchmann, A. et al., "Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control," *5th Data Engineering Conference*, Feb. 1989.

[Car89] Carey, M., R. Jauhari, and M. Livny, "Priority in DBMS Resource Scheduling," *15th VLDB Conference*, Aug. 1989, pp 397-410.

[Gar83] Garcia-Molina, H., "Using Semantic Knowledge for Transaction Processing in a Distributed Database," *ACM Trans. on Database Syst.*, vol. 8, no. 2, pp 186-213, June 1983.

[Har90] Haritsa, J., M. Carey, and M. Livny, "On Being Optimistic on Real-Time Constraints," *ACM PODS Symposium*, April 1990.

[Hua89] Huang, J., J. Stankovic, D. Towsley, and K. Ramamritham, "Experimental Evaluation of Real-Time Transaction Processing," *Real-time Systems Symposium*, Dec. 1989.

[Hua90] Huang, J., J. Stankovic, D. Towsley, and K. Ramamritham, "Real-Time Transaction Processing: Design, Implementation and Performance Evaluation," *Tech. Rep. TR-90-43*, Dept. of Computer and Information Science, University of Massachusetts, May 1990.

[IEEE90] *7th IEEE Workshop on Real-Time Operating Systems and Software*, University of Virginia, Charlottesville, Virginia, May 1990.

[Kung81] Kung, H. and J. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Syst.*, vol. 6, no. 2, pp 213-226, June 1981.

[Kor90] Korth, H., "Triggered Real-Time Databases with Consistency Constraints," *16th VLDB Conference*, Brisbane, Australia, Aug. 1990.

[Lin89] Lin, K., "Consistency issues in real-time database systems," *Proc. 22nd Hawaii Intl. Conf. System Sciences*, Hawaii, Jan. 1989.

[Lin90] Lin, Y. and S. H. Son, "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order," *11th IEEE Real-Time Systems Symposium*, Orlando, Florida, Dec. 1990.

[Liu88] Liu, J. W. S., K. J. Lin, and X. Song, "Scheduling Hard Real-Time Transactions," *5th IEEE Workshop on Real-Time Operating Systems and Software*, May 1988, pp 112-260.

[ONR89] *ONR Workshop on Foundations of Real-Time Computing*, White Oak, Maryland, Oct. 1989.

[Raj89] Rajkumar, R., "Task Synchronization in Real-Time Systems," *Ph.D. Dissertation*, Carnegie-Mellon University, August 1989.

[Sha88] Sha, L., R. Rajkumar, and J. Lehoczky, "Concurrency Control for Distributed Real-Time Databases," *ACM SIGMOD Record 17*, 1, March 1988, pp 82-98.

[Sha90] Sha, L., R. Rajkumar, S. H. Son, and C. Chang, "A Real-Time Locking Protocol," *IEEE Transactions on Computers*, to appear.

[Son88] Son, S. H., "Semantic Information and Consistency in Distributed Real-Time Systems," *Information and Software Technology*, Vol. 30, Sept. 1988, pp 443-449.

[Son88b] Son, S. H., editor, *ACM SIGMOD Record 17*, 1, Special Issue on Real-Time Database Systems, March 1988.

[Son89] Son, S. H. and H. Kang, "Approaches to Design of Real-Time Database Systems," *Symposium on Database Systems for Advanced Applications*, Korea, April 1989, pp 274-281.

[Son89b] Son, S. H., "On Priority-Based Synchronization Protocols for Distributed Real-Time Database Systems," *IFAC/IFIP Workshop on Distributed Databases in Real-Time Control*, Budapest, Hungary, Oct. 1989, pp 67-72.

[Son90] Son, S. H. and C. Chang, "Performance Evaluation of Real-Time Locking Protocols using a Distributed Software Prototyping Environment," *10th International Conference on Distributed Computing Systems*, Paris, France, June 1990, pp 124-131.

[Son90b] Son, S. H. and J. Lee, "Scheduling Real-Time Transactions in Distributed Database Systems," *7th IEEE Workshop on Real-Time Operating Systems and Software*, Charlottesville, Virginia, May 1990, pp 39-43.

[Son90c] Son, S. H. and N. Haghighi, "Performance Evaluation of Multiversion Database Systems," *6th IEEE International Conference on Data Engineering*, Los Angeles, Feb. 1990, pp 129-136.

[Song90] Song, X. and J. Liu, "Performance of Multiversion Concurrency Control Algorithms in Maintaining Temporal Consistency", *COMPSAC '90*, October 1990.

[Stan88] Stankovic, J., "Misconceptions about Real-Time Computing," *IEEE Computer 21*, 10, October 1988, pp 10-19.

[Vrb88] Vrbsky, S. and K. J. Lin, "Recovering Imprecise Transactions with Real-Time Constraints," *Symp. Reliable Distributed Systems*, Oct. 1988, pp 185-193.

[Yu90] Yu, P. and D. Dias, "Concurrency Control using Locking with Deferred Blocking," *6th Intl. Conf. Data Engineering.*, Los Angeles, Feb. 1990, pp 30-36.

Data Dredging

Shalom Tsur

Microelectronics and Computer Technology Corporation
Austin, Texas, 78759

August 20, 1990

Abstract

Data dredging is a paradigm of the next-generation DBMS applications. It involves inductive learning by the controlled scanning of large volumes of low-level historical data so as to formulate and verify certain high-level hypotheses.

In this short paper we review some of the issues related to this problem area, formulate some of the requirements of a technology to support it and briefly elaborate on a data dredging example.

1 Introduction

Data Dredging or *Data Mining*[1] is a term used to denote an emerging paradigm of next-generation DBMS applications. One feature of some of the next-generation DBMS applications involves very large volumes of data. The "Lagunita Report" on future directions in DBMS research, a copy of which appears in this issue, lists some of the sources of this data:

- Satellite image data, collected by NASA.
- Complete project data, as collected by hundreds of independent subcontractors.
- DNA sequence data, collected within the human genome initiative.
- Data collected by large commercial organizations; e.g., airline passenger and seat-reservation data.

In all cases the data are not collected for any particular purpose, but rather to serve as a historical repository that will be used for a variety of different analyses, many of which are unspecified when the data are collected. Other features of these sources are that, after their collection, already recorded data are seldom updated—only when found to be incorrect, but additional data are added over time.

Typical analyses over such repositories would include:

- Support for planning tasks. Historical cash register repositories may be scanned for outstanding demand, or the lack thereof, for particular items. Demand may be correlated with other factors such as economical or demographic data.

[1]The terms are used synonymously, this author prefers the former.

- Airline seat-reservation data are scanned to maximize yield per seat. This analysis is known as "yield management" [4].
- System performance data may be scanned for evidence of abnormal behavior. The definition of abnormal behavior is often non-trivial and may involve a large number of low-level, measurable factors that need to be combined in a variety of ways for this purpose.
- "Mature" databases. Relational database technology has been in use for more than a decade and many of the existing applications are coming of age. One sign of this aging process is that in many cases the data in these systems were carelessly entered or the systems were modified in some undocumented way, resulting in contents of incomplete, contradictory or simply unknown data. There is a need for the cleanup and reassessment of these data sources—often they are of an unexpected economical value to their owners.

In principle, it should be possible to separate between the *specification* of tasks of this kind, and their *verification* given the data at hand. In practice however, these aspects are seldom independent. A more realistic situation is that initially the task is ill-specified and only barely understood. The refinement of the specification and its subsequent verification are intertwined in an iterative process that eventually leads to a well-specified and verified task using the available data. During each of these iterations, human judgment is exercised: the user observes the results of the last iteration and decides whether the process has terminated and the task has been defined or whether further refinement is called for—involving thus new iterations.

In the next sections we elaborate on this process. We will contrast it with other forms of machine-assisted learning, formulate some of the requirements for the support of this activity, and show some specific examples.

2 Modes of Learning

Expertise about some application domain can be obtained in the following ways:

1. With the aid of a human expert whose expertise can be articulated and encoded in some knowledge representation scheme, e.g., a network of frames or a logical rule set. This is the mode commonly assumed by the AI expert system paradigm.
2. By generalizing the understood behavior of given examples. We will refer to this mode as *inductive learning*.

These modes are not exclusive. We will first review the latter. Typical examples of inductive learning include: statistical estimation of parameter values and their significance from sample data, the formulation of diagnostic rules from data or the inference of classification trees from sample data. For a recent selection of applications of this type the reader is referred to [5]. The common denominator in these applications is that the methods used in the inference process are "closed"—they progress automatically from some initial hypothesis via refinements, introduced by the method itself, to some result-form that represents the knowledge obtained. There is an implicit monotonicity assumption that can be stated as: "the more data used, the better the result approximates the 'true' knowledge". Little control can be exercised over this process barring limiting the amount of training data used. Another problem with these methods is that the results do not reflect any theory with which the user can reason: it is impossible to ask, for example, why certain parameter values are significant and others are not. The result must be accepted as is.

Knowledge derived from a human expert, on the other hand, can be formulated as an "open" theory that allows for reasoning and modification if required. The difficulty is of course that in many domains there are no human experts, and even if they were available, it is unclear whether their expertise could be articulated and formally encoded. Given the right tools however, it is possible to combine the best of both worlds by formulating a theory directly from the data. This is the essence of the data dredging method. The tool of choice in our case is $\mathcal{LDL}$ which we briefly describe in the next section.

3 Language and System Requirements

$\mathcal{LDL}$—Logical Data Language [8, 2] is a deductive database system that supports a declarative logic-based language and integrates relational database and logic programming technologies. The system can be used for data dredging by loading data into the (extensional) database and formulating a theory using rule-based intentions. The theory can be verified against the data via the execution of selected queries. The convergence from an initial idea to a well-defined concept can be described by the following abstract iterative process:

Formulate hypothesis;
while results do not verify or deny hypothesis **do**
begin (revise and) translate hypothesis into $\mathcal{LDL}$ rule set and query;
  Execute query and observe results;
end.

The convergence of this process is of course not guaranteed. Rather, it depends on the subjective judgment of the user who, after (hopefully) a small number of iterations, may decide that the modified rule set indeed defines the concept that he/she had initially in mind, or alternatively, that the data used did not support or deny the hypothesis in which case he/she might try it with different data. In principle this process might be attempted using any procedural programming language. In practice however, each iteration entails the usual coding/compiling/testing/debugging cycle, hence resulting in a lengthy process. The use of a system like $\mathcal{LDL}$ significantly shortens each iteration since the logical specifications are translated directly by the system into an executable query against the data. The major requirement for the support of tasks of this kind is thus for a high-level, declarative language that can be flexibly used in a variety of ways to formulate very complex queries against the data. Other requirements have to do with assisting the user in exploring the results at each stage of the iteration. These include support for different forms of the visualization of the results and an interaction capability which enables the user to modify his/her theory via a graphical interface. An overview of the desiderata for a system of this type is given by Brice and Alexander [1]. Parsaye *et al* [10] describe some features of an intelligent database system that includes a data discovery capability. However, this capability is of the closed form, described in the previous section, and produces rules that were statistically obtained but cannot be used for any further reasoning.

3.1 Example: the Convoy Problem

The example that we illustrate here comes from the area of computer system performance evaluation. As a model for a computer system we use a network of queues and servers. Dynamic entities, representing different parts of computational tasks, move through this network. The network may

be open i.e., entities arrive at certain nodes and depart from other nodes. It may be closed i.e., the same entities move in the network forever; or it can be mixed for different classes of computations. An attribute of normal behavior of the entities is stochastic independence—the probability of moving an entity from a given node to any of its successor nodes is independent of the other entities in the system. In practice however, anomalous behavior is occasionally observed: subsets of entities group together for some time and follow the same paths in the network. Thus, intuitively, one can think of this behavior as a "convoy". Performance data is collected at each of the network nodes and is of the type

$$data(Entity\#, Node\#, ArrTime, DepTime)$$

that is, for each entity number, the arrival and departure times at some node. The problem in this case can be stated as:

1. How do we define this high-level and intuitive notion of a convoy?
2. Given the definition, is there evidence of convoys from the given data?

We will only show a sketch of the solution here. The problem was encoded into $\mathcal{LDL}$ by L. Slepetis. For a complete description of this problem the reader is referred to [8]. A convoy can be described by the following characteristics:

1. *Lumpiness*: A transient in the queue length of some node. For some time interval the queue length is well above the average length.
2. *Succession*: The lump moves from a node to one of its successor nodes. This behavior is generalized over paths of any length in the network.
3. *Overlap*: There is a significant degree of overlap between the entity set of a lump at some node and the lump at its successor node.

Thus, the following rule[2] describes the succession relation:

$$succession(i, j, t_1, t_2) \leftarrow lump(i, t_1, \_), \\ lump(j, t_2, \_), j \in succ(i), t_2 - t_1 \leq \Delta.$$

The succession relation between nodes i and j exists if at time t_1 there exists a lump at node i, at time t_2 there exists a lump at node j, j is a successor node of i and the time interval between these occurrences does not exceed Δ. We have omitted the definition of the *lump* relation, which is derived from the given *data* relation.

The rule defining overlap is as follows:

$$overlap(i, j, t_1, t_2) \leftarrow S = S_i(t_1) \cap S_j(t_2), \\ \mid S \mid \geq \beta \mid S_i(t_1) \cup S_j(t_2) \mid, \\ j \in succ(i), t_2 > t_1.$$

$S_i(t_1)$ and $S_j(t_2)$ denote respectively entity sets at nodes i and j at times t_1 and t_2. $\beta, 0 \leq \beta \leq 1$, is a controlling threshold parameter for overlap. $\beta = 0$ implies no overlap and $\beta = 1$ implies total overlap.

A *migration* is a convoy of path length 1, defined as

[2]The syntax used here is for exposition only. It is a straightforward matter to translate this format into $\mathcal{LDL}$.

$$migration((i,t_1),(j,t_2)) \leftarrow succession(i,j,t_1,t_2), \\ overlap(i,j,t_1,t_2).$$

and *convoy* is the transitive closure over the *migration* relation, recursively defined as:

$$convoy([(i,t_1),(j,t_2)]) \leftarrow migration((i,t_1),(j,t_2)).$$
$$convoy(L) \leftarrow migration((i,t_1),(j,t_2)), \\ convoy(L1), \\ L1 = [(j,t_2),\_], \\ append([(i,t_1)],L1,L).$$

Note that these rules are the end product of the iterative process described above. In this case the number of iterations was small, the definition of the *lump* relation was modified 4 times before the definitions were derived in the form presented here.

3.2 Training, A Mixed-Mode Paradigm

How do we know that the definition of the convoy concept, presented in the previous section, is correct? In the process we have introduced numerous parameters: Δ, β and others that we have not mentioned here. Notice that many of these parameters may be node-dependent so that in reality we have introduced $\Delta_1, \Delta_2, \ldots, \beta_1, \beta_2, \ldots$ How do we estimate the values of these parameters? The answer to these questions is, as in other inductive learning situations, by training, i.e., the use of data sets that have a known interpretation. Training can be facilitated in a number of ways:

- By the use of data sets with given interpretations i.e., "expert-solutions". In this case the rule set is induced from these training sets. We are using this method in a data-dredging problem for the interpretation of DNA sequences from low-level gel electrophoresis data [9].
- By the use of artificial data that exhibits the behavior we want to analyze. This is the method that we used with the convoy problem. By means of a simulation we generated data that displayed convoy behavior and used it to induce the definitions.

The proliferation of parameters is an integral feature of data dredging, regardless of the domain studied. This is not surprising given that the introduction of these is to compensate for our limited understanding of the system behavior. The estimation of parameters can be done using any of the inductive learning methods that we have mentioned. Observed in another way, the parameters that we introduce are a way of expressing the *normal behavior* of the system. The *abnormal behavior* is given by the rule definitions. We conjecture that *any* system studied this way exhibits a mixture of normal and abnormal behavior. The normal behavior can be obtained by a variety of methods e.g., the training of neural networks [6, 7] or the use of genetic algorithms [3]; whereas abnormal behavior e.g., the knowledge of certain imperfections in the experiments that produced some of the DNA data, can only be formulated by utilizing background knowledge. It appears that the more successful methods will employ this mixed-mode paradigm.

4 Conclusion

In this short article we presented a brief overview of an emerging application area for next-generation database management systems. We reviewed some of the issues and suggested that successful solutions to the problems that will come up in this area require a blend of different technologies.

The real research problem is of course: What is the *exact* blend? And what is the relation of this blend to different domains of data? Our limited experience so far suggests that deductive database systems have a big role to play in this respect. Our future efforts in this area will concentrate on the mixed-mode paradigm that we have suggested and on related issues that we are aware of but have not examined, in particular, the control and manipulation of stream data.

References

[1] Brice, R. and W. Alexander, "Finding Interesting Things in Lots of Data", *Proceedings of the 23rd Hawaii International Conference on System Sciences*, Kona, Hawaii, January 1990.

[2] Chimenti, D. *et al.* "The $\mathcal{LDL}$ System Prototype", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 1, March 1990, pp. 76-90.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley Publishing Comp., 1989.

[4] M. D. Hopper, "Rattling SABRE—New Ways to Compete on Information", *Harvard Business Review*, May-June 1990, pp. 118-125.

[5] *Knowledge Discovery in Databases.* IJCAI-89 Workshop Proceedings, Detroit MI, August 20, 1989. G. Piatetsky-Shapiro, W. Frawley, eds.

[6] McLelland, J.L., and D. E. Rumelhart, eds., *Parallel, Distributed Processing*, Vol. 1, MIT Press, Cambridge Mass., 1986.

[7] McLelland, J.L., and D. E. Rumelhart, eds., *Parallel, Distributed Processing*, Vol. 2, MIT Press, Cambridge Mass., 1986.

[8] S. Naqvi, and S. Tsur. *A Logical Language for Data and Knowledge Bases.* W. H. Freeman, 1989.

[9] Overbeek, R., Price, M., and S. Tsur, "Automated Interpretation of Genetic Sequencing Gels", MCC Internal Report, March 1990.

[10] Parsaye K., Chignell M., Khoshafian S., and H. Wong, "Intelligent Databases", *AI Expert*, March 1990, pp. 38-47.

[11] Zaniolo, C., "Architectures of Deductive Database Systems". In Proc. *COMPCON 90'*, Feb. 26, 1990, San Francisco, CA.

THE IEEE COMPUTER SOCIETY
1730 MASSACHUSETTS AVENUE, N.W.
WASHINGTON, DC 20036-1903

NON-PROFIT ORG.
U.S. POSTAGE
PAID
SILVER SPRING, MD
PERMIT 1398

Mr. David B. Lomet
9 Cherry Lane
Westford, MA 01886
USA