③

AD-A227 362

A Mediator Architecture for Abstract Data Access

by

Wiederhold, Risch, Rathmann, DeMichiel, Chaudhuri, Lee, Law, Barsalou, Quass

Department of Computer Science

Stanford University
Stanford, California 94305

10 2

REPORT DOCUMENTATION PAGE		Form Approved OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	

4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)

6a NAME OF PERFORMING ORGANIZATION	6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION
Stanford University		

6c. ADDRESS (City, State, and ZIP Code)	7b ADDRESS (City, State, and ZIP Code)
Department of Computer Science Stanford, CA 94305	

8a. NAME OF FUNDING / SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DARPA		N00039-84-C-0211

8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS			
Arlington, VA	PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

A Mediator Architecture for Abstract Data Acces

12 PERSONAL AUTHOR(S)
Gio Wiederhold, et al.

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final report	FROM 1985 TO 1990	February, 1990	

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Databases, knowledge bases, future information systems

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This report contains some concept papers describing the general architecture that we envisage to be appropriate for future information systems, as well as a number of papers with specific research results.

The architecture presented here is conceived to deal with a wide variety of users, in many locals, served by many distinct experts, and using a wide variety of databases. It is not feasible for any single institution to address the entire problem. At Stanford, within the KBMS project, we have addressed a number of critical subtasks, but much more work needs to be done. While some results are ready for prototype implementation, the main motivation for issuing this report is to provide an overview with enough detail in some areas to allow the reader to gain insights into a direction that future information systems research needs to pursue.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
Gio Wiederhold	415-723-0872	

DD Form 1473, JUN 86   Previous editions are obsolete   SECURITY CLASSIFICATION OF THIS PAGE

A Mediator Architecture for Abstract Data Access

Gio Wiederhold*[†][§]   Tore Risch[†]   Peter Rathmann*

Linda DeMichiel*   Surajit Chaudhuri*

Byung Suk Lee*   Kincho H. Law[‡]   Thierry Barsalou*[§]

Dallan Quass*

February 23, 1990

Introduction

This report contains some concept papers describing the general architecture that we envisage to be appropriate for future information systems, as well as a number of papers with specific research results.

The architecture presented here is conceived to deal with a wide variety of users, in many locations, served by many distinct experts, and using a wide variety of databases. It is not feasible for any single institution to address the entire problem. At Stanford, within the KBMS project, we have addressed a number of critical subtasks, but much more work needs to be done. While some results are ready for prototype implementation, the main motivation for issuing this report is to provide an overview with enough detail in some areas to allow the reader to gain insight into a direction that future information systems research will have to pursue.

The mediator concept, basic to this architecture, envisages smart modules to be interposed between the users' workstations and the underlying information resources. At the base of this system hierarchy can be all types of databases, often autonomous and maintained by a variety of organizations.

Mediator modules understand the semantics of the databases. This understanding is due to knowledge which experts have contributed. The knowledge must cover both the syntax of access and the semantics of the data being accessed. Knowledge maintenance is important for long-term viability of these modules in this architecture. The users' workstations select and access those mediators appropriate for the information task at hand. User pragmatics and display functions are embodied within their own application modules.

A variety of interactions will occur among the modules; several are described in the papers collected in this report. An implementation of these notions will depend greatly on effective, high-speed communication networks.

The first two papers provide general conceptual guidance. In [1]* the data engineering motivations for the architecture is presented, while [2] presents an initial formalism to deal with complexity of combining information processed via distinct mediators.

In [3] we present our specific research architecture. An important point is that we partition pragmatic and formally managed knowledge amoung the users' and the experts' mediator modules. We believe that this partitioning is essential for the growth of knowledge-based systems. The concepts underlying one type of mediator module, namely abstraction by generalization, are presented in [4].

The interfaces with the underlying databases are a critical concern. We will often want to combine base information which has differences in semantic scope and representation. An algebra to deal with this problem is presented in [5]. In [6] triggering mechanisms are defined for a database so that the expert's and user's knowledge can be maintained as the base data, which represent the real world, change.

As data move from databases to mediators, the relational representation is typically changed to an object- or frame-based representation. A related project, PENGUIN, focuses on this issue; we include one paper illustrating the application of those concepts in a Civil Engineering application [7]. Efficiency is a concern during this transfer. Since we expect to operate in a fully distributed environment, optimization criteria change, as exemplified in [8], where the requirement for outerjoin computation for remote object generation has been addressed.

We are continuing to work on more research issues in this environment, and are planning to cooperate with a number of other institutions in this work. If possible, we will participate in annual workshops devoted to these issues. The 1990 workshop is being organized by Prof. Yuri Breitbart of the University of Kentucky. Other papers produced at Stanford related to this work are cited in these papers. Please contact the authors if you need copies.

---

*The references cite the paper number in the table of contents

Contents

iv

The Architecture of Future Information Systems

Gio Wiederhold

Stanford University

February 14, 1990

Abstract

The installation of high-speed networks using optical fiber and high bandwidth messsage forwarding gateways is changing the physical capabilities of information systems. These capabilities must be complemented with corresponding software systems advances to obtain a real benefit. Without smart software we will gain access to more data, but not improve access to the type and quality of information needed for decision making.

To develop the concepts needed for future information systems we model information processing as an interaction of data and knowledge. This model provides criteria for a high-level functional partitioning. These partitions are mapped into information processing modules. The modules are assigned to nodes of the distributed information systems. A central role is assigned to modules that *mediate* between the users' workstations and data resources. Mediators contain the administrative and technical knowledge to create information needed for decision-making. Software which mediates is common today, but the structure, the interfaces, and implementations vary greatly, so that automation of integration is awkward.

By formalizing and implementing mediation we establish a partitioned information systems architecture which is of managable complexity and can deliver much of the power that technology puts into our reach. The partitions and modules map into the powerful distributed hardware that is becoming available. We refer to the modules that perform these services in a sharable and composable way as *mediators.*

We will present conceptual requirements that must be placed on mediators to assure effective large-scale information systems. The modularity in this architecture is not only a goal, but also enables the goal to be reached, since these systems will need autonomous modules to permit growth and enable them to survive in a rapidly changing world.

The intent of this paper is to provide a conceptual framework for many distinct efforts. The concepts provide a direction for an information processing systems in the foreseeable future. We also indicate some sub-tasks that are of research concern to us. In the long range

the experience gathered by diverse efforts may lead to a new layer of high-level communication standards.

1. Introduction

Computer-based information systems, connected to world-wide high-speed networks provide increasingly rapid access to a wide variety of data resources [Mayo:89]. This technology opens up possibilities of access to data, requiring capabilities for assimilation and analysis which greatly exceed what we now have in hand. Without intelligent processing these advances will have only a minor benefit to the user at a decision-making level. That brave user will be swamped with ill-defined data of unknown origin.

1.1 The problems

We find two types of problems: for single databases the volume of data, the lack of abstraction, and the need to understand the data representation hinder end-user access; for jointly processing information from multiple databases the mismatch problem of information representation and structure is the major concern.

Volume

The volume of data can be reduced by selection. It is not coincidental that SELECT is the principal operation of relational database management systems, but selected data is still at too fine a level of detail to be useful for decision making. Further reduction is achieved by bringing data to *higher levels of abstraction*. Aggregation operations as COUNT, AVERAGE, SD, MAX, MIN, etc. provide some computational facilities for abstraction, but any such abstraction is formulated within the application using some domain knowledge.

Type of abstraction	Example		
	Base data		*Abstraction*
Granularity	Sales detail	→	Product summaries
Generalization	Product data	→	Product type
Temporal	Daily sales	→	Seasonally adjusted monthly sales
Relative	Product cost	→	Inflation adjusted trends
Exception recognition	Accounting detail	→	Evidence of fraud
Path computation	Airline schedules	→	Trip duration and cost

Figure 1. Abstraction functions.

For most base data more than one abstraction must be supported — for the salesmanager

3

the aggregation is by sales region, while for marketing aggregations by customer income are appropriate. Examples of required abstraction types are given in Fig. 1.

Computational requirements for abstraction are often complicated. The groupings to define abstractions may for instance involve recursive closures. These cannot be specified with current database query languages. Application programs are then written by specialists to reduce the data. The use of specific data-processing programs as intermediaries diminishes flexibility and responsiveness for the end user. Now the knowledge that creates the abstractions is hidden and hard to share and reuse.

Mismatch

Data obtained from remote and autonomous sources will often not match in terms of naming, scope, granularity of abstractions, temporal bases, and domain definitions, as listed in Figure 2. The differences shown in the examples must be resolved before automatic processing can join these values.

Type of mismatch	Example _Domains_		
Key difference	Alan Turing:The Enigma _reference for reader_	versus	QA29.T8H63 _reference for librarian_
Scope difference	employees paid _includes retirees_	versus	employees available _includes consultants_
Abstraction grain	personal income _from employment_	versus	family income _for taxation_
Temporal basis	monthly budget _central office_	versus	weekly production _factory_
Domain semantics	postal codes _one can cover multiple places_	versus	town names _can have multiple codes_
Value semantics	excessive_pay _per internal revenue servive_	versus	excessive_pay _per board of directors_

Figure 2. Mismatches in data resources.

Without an extended processing paradigm, as proposed in this paper, the information needed to initiate actions will be hidden in ever larger volumes of detail, scrollable on ever

larger screens, in ever smaller fonts. In essence, the gap between information and data will yet be wider than it is now. Knowing that information exists, and is accessible creates expectations by end-users. Finding that it is not available in a useful form or that it cannot be combined with other data creates confusion and frustration. We believe that the objection made by some users about computer-based systems, that they create *information overload*, is voiced because those users get too much of the wrong kind of data.

1.2 Use of a model

In order to visualize the requirements we place on future information systems, we consider the activities that are carried out today whenever decisions are being made. The making of informed decisions requires the application of a variety of knowledge to information about the state of the world. To clarify the distinction of data and and knowledge in our model we restate a definition from [Wiederhold:86B].

> **Data** describes specific instances and events. Data may gathered automatically or clerically. The correctness of data can be verified vis-a-vis the real world.
>
> **Knowledge** describes abstract classes. Each class typically covers many instances. Experts are needed to gather and formalize knowledge. Data can be used to disprove knowledge.

To manage the variety of knowledge, we employ specialists, and to manage the volume of data, we segment our databases. In these partitions partial results are produced, abstracted, and filtered. The problem of making decisions is now reduced to the issue of chosing and evaluating the significance of the pieces of information derived in those partitions and fusing the important portions.

For example, an investment decision for a manufacturer will depend on the fusion of information on its own production capability versus that of others, of its sales experience in related products, of the market for the conceived product at a certain price, of the cost-to-price ratio appropriate for the size of the market, and its cost of the funds to be invested. Specialists will consider these diverse topics, and each specialist will consult multiple data resources to support their claims. The decision maker will fuse that information, using considerations of risk and long-range objectives in combining the results.

An effective architecture for future information systems must support automated information acquisition processes for decision-making activities. By default, we approach the solution in an manner analogous seen in human-based support systems. While we model the system based on abstractions from the world of human information processing, we do not constrain the architecture to be anthropomorphic and to mimic human behavior. There are many aspects of human behavior which we have not yet been able to capture and formalize

adequately. Our goal is merely to define an architecture wholly composed of pieces of software that are available or appear to be attainable in a modest timeframe, say ten years. The demands of the software in terms of processing cost should be such that modern hardware can deal with it.

1.3   Current state

We are not starting from a zero base. Systems are becoming available now with the capabilities envisaged by Vannevar Bush for his MEMEX in 1945 [Bush:45]. We can select and scroll information on our workstation displays, we have remote access to data and can present the values on one of multiple windows, we can insert documents into files in our workstation, and we can annotate text and graphics. We can reach conclusions based on this evidence and advise others of decisions made.

Our vision in this paper is intended to carry us beyond, specifically to provide a basis for automated integration of such information. Our current systems, as well as MEMEX, do not adress that phase.

2.   A Model of Information Processing

The objective of obtaining information was already clearly stated by Shannon [Shannon:48]. Information enables us to decide among several actions which are otherwise not distinguisable. Consider again a simple business environment. A factory manager needs sales data to set production levels. A sales manager needs demographic information to project future sales. A customer wants price and quality information to make purchase choices.

Most of the information needed by the people in the example can be represented by factual data and should be available on some computer somewhere. Communication networks have the potential to make data available wherever it is needed. However, to make the decisions, a considerable amount of knowledge has to be applied as well. Today, most knowledge is made available through a variety of administrative and technical staff in an institution [Waldrop:84]. Some knowledge is encoded in data-processing programs and expert systems for automated processing.

The process of generating supporting information does not differ much in partially automated or manual systems. In manual systems the decision maker obtains assistance from staff and colleagues who peruse files and prepare summarizations and documentation for their advice. In automated systems the staff is likely to use computers to prepare these documents. The decision-maker rarely uses the computer, because the information from multiple sources is difficult to integrate automatically.

2.1 The processing and the application of knowledge

A technician will know how to select and transfer data from a remote computer to one used for analysis. A data analyst will understand the attributes of the data and define the functions to combine and integrate the data [deMichiel:89]. A statistician may provide procedures to aggregate data on customers into groups that present distinctive behavior patterns. A psychologist may provide classification parameters that characterize the groups. Finally, a manager has to assess the validity of the classifications that have been made. use the information to make a decision. and assume the risk of making the decision. A public relations person may take the information and present it in a manner that can be explained to the stockholders, to whom the risk is eventually distributed. Since these tasks are characterized by using data and knowledge gathered in the past, with the objective of affecting the future. we will refer to these tasks as *planning*. Our definition of planning is broader than that used in Artificial Intelligence research [Cohen:82], although the objectives are the same. To be able to deal with these planning actions in a focused way. we will model the information processing aspects.

There is a recurring activity here:

1 Data are made available. These are either factual observations, or results from prior processing, and combinations thereof.
2 Knowledge is made available. It derives from formal training and experience.
3 Knowledge about the data and its use is applied in two phases:
  i Selection: subsets of available data are
     (1) defined,   (2) obtained. and   (3) merged.
  ii Reduction: the data found are summarized to an appropriate level of abstraction.
4 Several such results are made available and fused.
5 The combined information is utilized in two ways:
  i Actions are taken that will affect the state of the world.
  ii Unexpected results will be used to augment the experience base of the participants, and others who receive this information, increasing their knowledge
6 The process loops are closed when
  i The actions and their effect is observed and recorded in some database.
  ii The knowledge is recorded to affect subsequent data definition, selection, or fusion.

The two distinct feedback loops and their interaction are illustrated in Figure 3. The data loop closes when the effects of actions taken are recorded in the database. The knowledge loop closes when recently gained knowledge is amde available so it can be used for further selection and data reduction decisions.

The interaction is of prime concern for future systems. Tools for the other steps are easy to identify, we list some in Section 3. We now consider in detail aspects of Step 3 identified above.
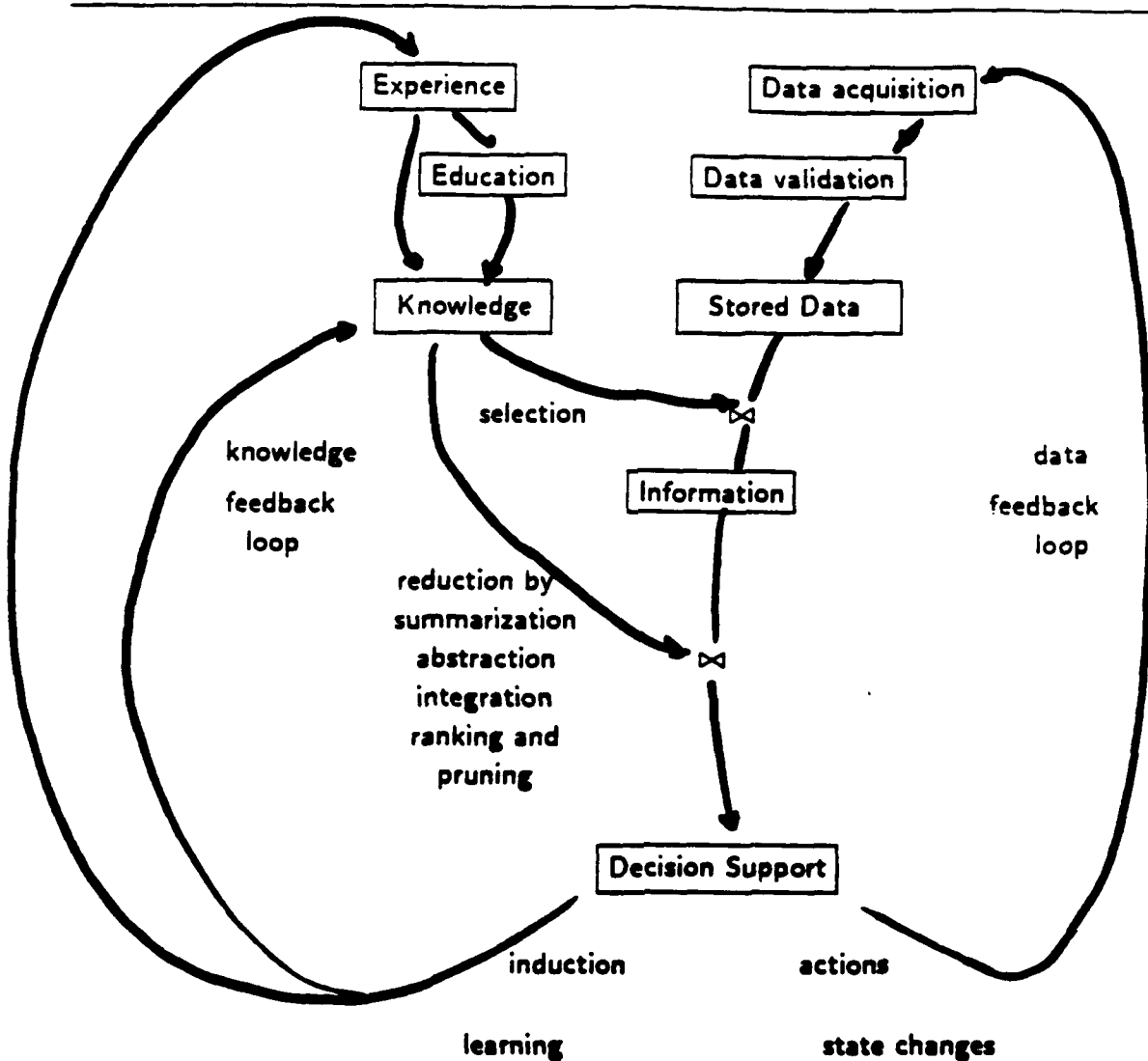
Figure 3. The knowledge and data feedback loops, and their interaction.

2.2 Creation of information

Let us identify where during processing information is created. Since getting information means that something has been obtained that was not known prior to the event, one or more of the following conditions have to hold:

1 The information is obtained from a remote source and was previously not known locally (Step 3.i.2 above). Here the information system must provide communication support.

 A special case of this condition is when a database is used for recall, to provide data we knew once, but cannot remember with certainty. Here the database component is used to communicate over time — from the past to the present.

2 Two facts, previously distinct are merged or unified (Step 3.i.3 above). A classic, although trivial, example is finding one's grandfather via transitivity of parents. In

realistic systems, unification of data also involves computation of functions, say the average income and its variation of groups of consumers.

3 Multiple results are fused using pragmatic assessments of the quality and risks associated within Step 4. Here abstractions are combined rather than facts, and the processing techniques are those associated with symbolic processing in rule-based expert systems, although they are also found coded in application programs. In our example the market specialist may want to unify incomes of current consumers with their reading habits to devise an advertising strategy.

Databases record detailed data for each of many instances or events. Reducing this detail to a few abstract cases, raises the information content per element. Of these abstractions only a small, feasible number of justified results is brought to the decision-maker. For instance, the market analyst has made it possible that decisions do not have to deal with individual consumers, but with consumer groups. Certain groups may be unlikely purchasers and are not targeted for promotions.

While the behavior of any individual may not be according to the rules hypothesized in the prior steps, the expected behavior of the aggregate population should be close to the prediction. Failures occur only if we have many errors in the underlying data or serious errors in our knowledge. Uncertainty, however, is common.

2.3 Uncertainty

We cannot predict the future with certainty. For automation of full-scale information systems the processing of uncertainty must be supported, although there are subtasks which can be precisely defined. Uncertainties within a domain may be captured by a formal model. Although we have the argument that all uncertainty computation can be subsumed by probabilistic reasoning [Cheeseman:85] [Horvitz:86], it seems that the variety of assumptions made is based on differences in domain semantics. During analysis uncertain events or states have to be combined for extrapolation into the future. We still have no overall model to predict which uncertainty-combining computations will be best for some domain.

To assess the extent of uncertainty affecting predictions we must combine the uncertain precision of source information, and the uncertainty created at each step where we combine them. In the various steps, we collect observations based on some criterion — say people living in a certain postal-code area. We also have data to associate an income level with that postal-code area, and perhaps even an income distribution. At the same time we may know the income distribution of people buying some product.

It is hence natural to make the conceptual unification step of postal code to potential

sales. Unfortunately, there is no logical reason why such a unification should be correct. We have some formal classes — namely people with a certain postal code, and some other formalizable classes based on income; in addition, there are some *natural* classes, not formalized, but intuited. In our example some natural classes of interest are the potential purchasers, of which there are several subgroups, namely those that bought in the past, those that will buy today, and those that will be buying the planned products. For the future class only informal criteria can be formulated.

The planner, at the decision-making node will use definable classes — by postal code, by observed and recorded purchasing patterns — to establish candidate members for the natural class of potential consumers. These classes overlap — the better the overlap is the more correct the decision-maker's predictions will be. If we infer over classes that do not match well, the uncertainty attached to the generated plans will be greater. But uncertainty is the essence of decision-making and is reflected in the risk that the manager of our example takes on. We would not need a manager with decision-making skills if we only have to report the postal codes for our base group of potential consumers.

2.4 Summary of the information model

Effective information is created at the confluence of knowledge and data. For prediction we use knowledge to conceive rules applicable to natural classes. Selection of data for decision-making is constrained by being based on formal class definitions. Uncertainty is created when formal and natural classes are matched.

Communication of knowledge and data is necessary to achieve this confluence. The communication may occur over space or over time. The information systems we consider must support both communication and fusion of data and knowledge.

Our systems must also be able to deal with continuing change. Both data and knowledge change over time because the world changes and because we learn things about our world. Rules that were valid once eventually become riddled with exceptions, and a specialist who does not adapt will find his work to become without value. An information system architecture must deal explicitly with knowledge maintenance.

3. Information System Components

We will now characterize the components available today to build information systems. All these components are positioned along a *data highway*, provided by modern communication technology. The interactions of the components will be primarily constrained by logical and informational limitations, and not by physical linkages. When we place the components into the conceptual architecture we will recognize lacunae, i.e., places where there are currently no, inadequate, or uncooperative components. We will see where the components must work together. Effective systems can be achieved only if the data and knowledge interfaces of the components are such that cooperation is feasible.

3.1 Data and knowledge resources

There is a wide variety of data resources. We might classify them by a measure of closeness to the source. Raw data obtained from sensors, such as purchase records obtained by point of-sale scanners, or, on a different scale, images recorded by earth satellites, are at the factual extreme. Census and stock reports contain data that have seen some processing, but will be considered as facts by most users.

At the other extreme are textual representations of knowledge. Books, research reports, and library material, in general, contain knowledge, contributed by the writers and their collegues. Unfortunately, from our processing-oriented view, that knowledge is not exploitable without a human mediator. The text, tables, and figures contained in such documents is data as well, but rarely in a form that can be transcribed for database processing.

If document information is stored in bibliographic systems, such as DIALOG [Summit:67] or MEDLINE [Sewell:87], only selection and presentation operations are enabled. Textual material does not lend itself well to automated analysis, abstraction, and generalization. Some types of reports, produced routinely, tend to have a degree of structure that makes some extent of analysis feasible, as demonstrated for chemical data [Callahan:81], pathology reports [Sager:85], or military event reporting messages [McCune:85].

3.2 Workstation applications

The systems environment for planning activities is provided by the new generations of capable workstations. For planning the users need to interact with their own hypotheses, save intermediate results for comparison of effects, and display the alternate projections over time. This interaction with information establishes the insights needed to gain confidence in one's decisions.

The user exercises creativity at the workstation. We hence do not try to constrain the user here at all. By providing comprehensive support for access to information we hope that

the complexity of the end-user's applications can be reduced to such an extent that quite involved analyses remain manageable.

Modern operating and network systems help much with simplifying the users' tasks by removing concern about managing hardware resources. The architecture presented here is meant to address the management of information resources, residing on such hardware.

The base information for planning processes has to be obtained from a variety of data resources. A capable interface is required.

3.3 Mediation

An interface from the users workstation to the database servers which only defines communication protocols and formats in terms of database elements does not deal with the abstraction and representation problems existing in todays' data and knowledge resources. The interfaces must take on an active role.

We will refer to the dynamic interface function as *mediation*. This term includes the processing needed to make the interfaces work, the knowledge structures that drive the transformations needed to transform data to information, and any intermediate storage that is needed.

To clarify the term we will list some examples of mediation found in current information systems. This list could be greatly expanded in length an depth. The citations provide linkages for follow-up; we expect that most readers will have encountered these or similar functions. Types of mediation functions that have been developed are:

1 Transformation and subsetting of databases using view definitions and object templates [Chamberlin:75] [Wiederhold:86] [Lai:88] [Barsalou:88]

2 Methods to access and merge data from multiple databases [Smith:81] [Dayal:83] [Saccá:86]

3 Computations that support abstraction and generalization over underlying data [Hammer:78] [Adiba:81] [Ozsoyoglu:84] [Downs:86] [Wiederhold:87] [deZegher:88] [Cichetti:89] [Chen:89] [Chaudhuri:90]

4 Intelligent directories to information bases, as library catalogs, indexing aids, and thesaurus structures [Humphrey:87] [Doszkos:86] [McCarthy:88]

5 Methods to deal with uncertainty and missing data because of incomplete or mismatched sources [Callahan:81] [Chiang:82] [Litwin:86] [deMichiel:89]

Many more examples can be added. Most readers will have used or created such interface modules at some time, since we all need information from large, autonomous, and mismatched sources. A major motivation for expert database systems is mediation. We excluded from the

list simple processing techniques that do not depend on knowledge that is extraneous to the database proper, such as indexes, caching mechanisms, network services, and file directories.

The examples of mediation shown shown are specialized, and tied either to a specific database or to a particular application, or both. We will now define *mediators* as modules occupying an explicit, active layer between the users' applications and the data resources. Our goal is a sharable architecture.

4. Mediators

In order to provide intelligent and active mediation, we envisage a class of software modules which mediate between the workstation applications and the databases. These *mediators* will form a distinct, middle layer, making the user applications independent of the data resources. What are the transforms needed in such a layer, and what form will the modules supporting this layer have? The responses to these questions are interrelated. We will first identify problems with generalizing the concept of mediation identified in current information systems, and then define some general criteria. We will also justify the necessity of having a modular, domain-specific organization in this layer.

4.1 The Architectural Layers

We create a central layer by distinguishing the function of mediation from the user-oriented processing and from database access. Most user tasks will need multiple, distinct mediators for their subtasks. A mediator uses one or a few databases.

The interfaces that are to be supported provide the cuts where communication network services are needed, as shown in Figure 4.

---

result → decision making

Layer 3   Independent **applications** on workstations   — managed by decision makers

- - - -   network services to information servers - - - -

Layer 2   Multiple **mediators**   — managed by domain specialists

- - - -   network services to data servers   - - - - -

Layer 1   Multiple **databases**   — managed by database administrators

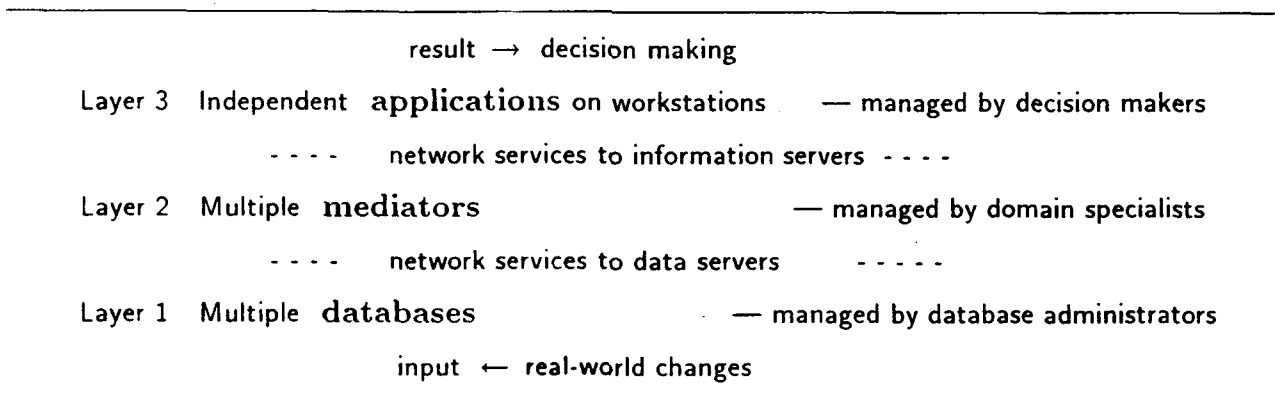input ← real-world changes

---

Figure 4: The three layers of this architecture.

Unfortunately, the commonality of function seen in the examples cited in Section 3.3 does not extend to an architectural commonality: the various examples are bound to the data resources and the end-users applications in different ways. It is here where new technology must be established if fusion at the application level is to be supported. Accessing one mediator at a time does not allow for fusion; and seeing multiple results on distinct windows of one screen does not support automation of fusion.

4.2 An inappropriate approach to mediation

The concept of mediation is related to the notion of having corporate *information centers*, as promoted by IBM and others [Atre:86]. These are to be corporate resources, staffed, and

14

equipped with tools to aid any users needing information. The needs and capabilities of an information center, however, differ in two respects from those of automated mediators:

1 A single center, or mediator for that matter, cannot deal with all the varieties of information that is useful for corporate decision-making.

2 Automation of the function will be necessary to achieve acceptable response time and growth of knowledge and its quality over time.

3 The user should not be burdened with the task of seeking out information sources. This task, especially if it is repetetive, is best left to an interaction of application programs in a workstation and automated mediation programs.

The information center notion initiates yet another self-serving bureaucracy within a corporation. Effective staff is not likely to be content in the internal service roles that an information center provides, so that turnover of staff and its knowledge will be high. The only role foreseen in such a center is mediation — bringing together potential users with candidate information.

In order to manage mediation, modularity instead of centralization seems to be essential. Modularity is naturally supported by a distributed environment, the dominant environment of computing in the near future.

4.3 Mediators

Now that we have listed some examples of mediators in use or planned for specific tasks, we can give a general definition.

> A **mediator** is a software module which exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications.

We place the same requirements on a mediation module that we place on any software module: it should be small and simple [Boehm:84] [Bull:87] so that it can be maintained by one expert or at most by a coherent group of experts.

An important, although perhaps not essential, requirement we'd like to place on mediators is that they be *inspectable* by the potential users. For instance, the rules used by a mediator using expert system technology can be obtained by the user as in any good cooperative expert system [Wick:89]. In this sense the mediators provide data about themselves in response to inspection and such data could be analyzed by yet another, an *inspector* mediator.

Since eventually there will be a great number and variety of mediators, the users have to be able to choose among them. Inspectability enables that task. For instance, distinct mediators which can provide the ten best consultants for database design may use different

evaluation criteria: one may use the number of publications and another the number of clients.

Some *meta*-mediators will have to exist that merely provide access to catalogues listing available mediators and data resources. The search may go either way: for a given data source one may want to locate a knowledgeable mediator and for a desirable mediator we need to locate an adequate data resource. It will be essential that the facilities provided by these meta-level mediators can be integrated into the general processing model, since search for information is always an important aspect of information processing. Where search and analyses are separated — as is still common today — for instance, in statistical data-processing, trying to find the data is often the most costly phase of information processing.

For databases that are autonomous, it is desirable that only a limited and recognizable set of mediators depend on anyone of them. Focusing data access through a limited number of views maintained by these mediators provides the data independence which is necessary for databases that are evolving autonomously. Currently, compatibility constraints are hindering growth of databases in terms of structure and scope, since many users are affected. As the number of users and the automation of access increases, the importance of indirect access via mediators will increase.

4.4   The Interface to Mediators

The most critical aspect of this three-layer architecture are the two interfaces that are now created. Today's mediating programs employ a wide variety of interface methods and approaches. The user learns to use one or a few of them, and then remains committed until its performance becomes wholly unacceptable. Unless the mediators are easily and flexibly accessible, the model of common information access we envisage is bound to remain a fiction. It is then in the interface and its support that the research challenge lies. Since our hardware environment implies that mediators can live on any nodes, not only on the workstations and database hosts, their interfaces must be grounded in communication protocols.

The User's Workstation Interface to the Mediators

The range of capabilities that mediators may have is such that a high-level language should evolve to drive the mediators. We are thinking of language concepts here, rather than of interface standards, to indicate the degree of extensibility that must be provided if the mediating concepts are to be generalized.

Determining an effective interface between the workstation application and the mediators will be a major research effort in refining this model. It appears that a *language* is needed to provide flexibility, composability. iteration, and evaluation in this interface. Descriptive.

but static interface specifications seem not be able to deal with the variety of control and information flow that must be supported. The basic language structure should permit incremental growth so that new functions can be supported as mediators join the network to provide new functionality.

It is important to observe that we do not see a need for a *user-friendly* interface. We need here a machine- and communication-friendly interface. Programs on the user's workstations can provide the type of display and manipulation functions appropriate to its type of user. This attitude avoids the dichotomy that has lead to inadequacies in SQL, which tries to be user-friendly, while its predominant use will be for programmed access [Stonebraker:88]. Standards needed here can only be defined after experience has been obtained in sharing of these resources to support the high-level functions needed for decision-making.

The Mediator to DBMS Interface

Existing database standards as SQL and RDA provide a basis for database access by mediators. Relational concepts as selection, views, etc., provide an adequate starting point. A mediator dealing mainly with a specific database need not be constrained to a particular interface protocol, while a mediator which is more general will be effective through a standard interface. A mediator which combines information from multiple databases may use its knowledge to control the merging process and use a relational algebra subset. Joins may, for instance, be replaced by explicit semi-joins, so that intelligent filtering can occur during processing. Still, dealing with multiple sources is likely to lead to incompleteness. Outer-joins are often required for data access to avoid losing objects with incomplete information [Wiederhold:83].

The separation of user applications and databases that the mediating modules provide also allows reorganization of data structures and redistribution of data over the nodes without affecting the functionality of the modules. The three-level architecture then makes an explicit tradeoff in favor of flexibility versus integration. The arguments for this tradeoff focus on the variety of uses made of databases, and by extension, the results of mediator modules [Wiederhold:86].

1 Sharability of information requires that database results can be configured according to one of several views. Mediators, being active, can create objects for a wide variety of orthagonal views [Barsalou:88].

2 On the other hand, making complex objects themselves persistent binds knowledge to the data, which hinders sharability [Maier:89].

3 The loss of performance, due to the interposition of a mediator, can be overcome by techniques listed in Section 5.4.

These arguments do not yet address the distribution of the mediators we envisage.

Available interfaces

We need interface protocols for data and knowledge. For data transmission there are developing standards. The level (in the OSI sense [Tanenbaum:87]) that we are concerned is within the application layer. We have faith that communication systems will soon handle all lower level support layers without major problems. The Remote Data Access (RDA) protocol provides one such instance [ISO/RDA:87].

Other technologies that are pushing capabilities at the interface is the National File System, being promoted by [Spector:88] at CMU. However, as mentioned in the introduction, communication of data alone does not guarantee that the data will be correctly understood for processing by the receiver. Differences often exist in the *meaning* assigned to the bits stored, some examples were shown in Figure 2.

4.5 Sharing of mediator modules

In either case, since we are getting access to so much more data, from a variety of sources, arriving at ever higher rates, automated processing will be essential. The processing tasks needed within the mediators are those sketched in the interaction model of Fig. 3: selection, fusion, reduction, abstraction, and generalization. Diverse mediator modules will use these functions in varying extents to provide the support for user applications at the decision making layer above.

The mediator modules will be most effective if they can serve a variety of applications [Hayes-Roth:84]. The applications will compose their tasks as much as possible by acquiring information from the set of available mediators. Unavailable information may motivate the creation of new mediators.

Sharing reinforces the need for two types of partitioning: one, into horizontal layers for end-users, mediators, and databases, respectively, and two, vertically into multiple user applications, each using various configurations of mediators. A mediator in turn will use distinct views over one or several databases. Just as databases are justified by the shared usage they receive, mediators should be sharable. Note that today's expert systems are rarely modular and sharable, so that their development and maintenance cost is harder to amortize.

For instance, the mediation module which can deal with inflation adjustment can be used by many applications. The mediator which understands postal codes and town names can be used by the post office, express delivery services, and corporate mail rooms.

We foresee here an incentive for a variety of specialists to develop powerful, but generally

useful mediators, which can be used by multiple customers. Placing one's knowledge into a mediator can be more rapidly effective, and perhaps more rewarding, than writing a book on the topic.

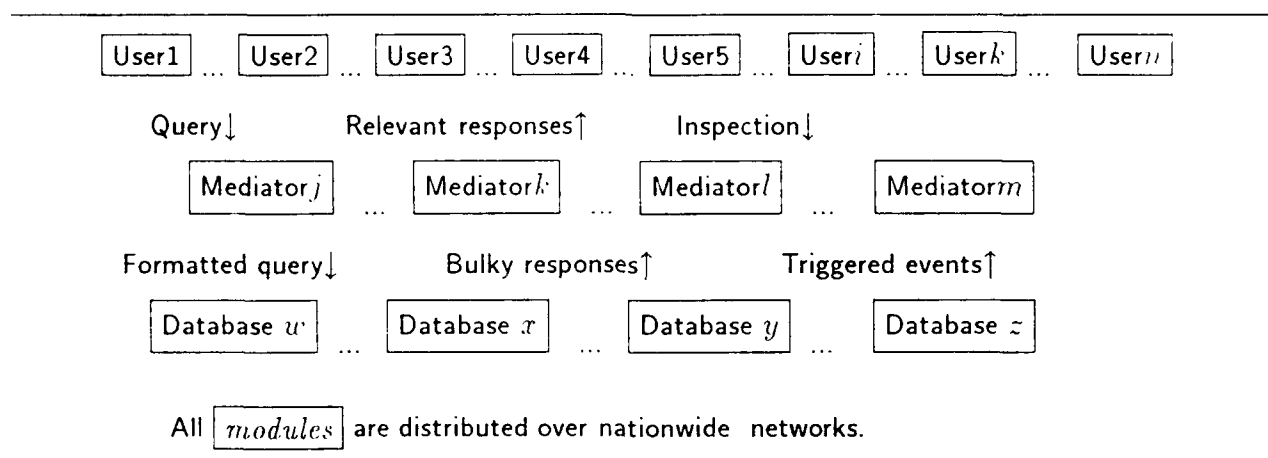We can now summarize these observations in Figure 5.

| User1 | ... | User2 | ... | User3 | ... | User4 | ... | User5 | ... | User$i$ | ... | User$k$ | ... | User$n$ |

Query↓   Relevant responses↑   Inspection↓

| Mediator$j$ | ... | Mediator$k$ | ... | Mediator$l$ | ... | Mediator$m$ |

Formatted query↓   Bulky responses↑   Triggered events↑

| Database w | ... | Database x | ... | Database y | ... | Database z |

All | $modules$ | are distributed over nationwide networks.

Figure 5. Interfaces for information flow.

4.6 Distribution of Mediators

We have implied throughout that mediators are distinct modules, distributed over the network. Distribution can be motivated by greater economy for access, by better locality for maintenance, and by issues of modularity. For mediators the two latter arguments are the driving force for distribution.

Why should mediators not be attached to the databases? In many cases it may be feasible; in general it is not appropriate.

1 The mediator contains knowledge that is beyond the scope of the database proper. A database programmer, dealing with, say, a factory production control system, cannot be expected to foresee all the strategic uses of the collected information.

2 Concepts of abstraction are not part of database technology today; the focus has been on reliable and consistent management of large volumes of detailed facts.

3 Intelligent processing of data will often involve dealing with uncertainty, adding excessive complexity to database technology.

4 Many mediators will access multiple databases to combine disjoint sets of data prior to analysis and reduction.

Similarily we can argue that the mediators should not be attached to the users' workstation applications. Again, the functions that mediators provide are of a different scope than the tasks being performed on the workstations. Workstation applications may use a variety of mediators to explore the data resources.

19

A major motivation for keeping mediators distinct is maintenance. During the initial stage of most projects which developed expert systems, their knowledge bases simply grew, and the cost of knowledge acquisition dominated the cost of knowledge maintenance. Many projects, in fact, assumed implicitly that knowledge, once obtained, would remain valid for all times. History teaches us that this is not true; although some fundamental rules may indeed not change for a long time, the application heuristics for its use change as the demands of the world change. Concepts as KNOSs [Tsichritzis:87] recognize the problem, and focus on the problem within a specific domain.

Maintenance by an outside expert of knowledge stored within an application system is intrusive and risky. The end-user should make the decision when new knowledge should be incorporated. We hence keep the mediator modules distinct.

The efficiency concerns of separating knowledge and data can be mitigated by replication. Since mediators (incorporating only knowledge and no factual data) are relatively stable, they can be replicated as needed and placed on nodes along the data highway where they are maximally effective. They certainly should not change during a transaction. As long as the mediators remain small, they can also be easily shipped to a site where substantial data volumes have to be processed.

4.7 Triggers for knowledge maintenance

We have added one aspect of mediation into Figure 5 which has not yet been discussed. Since the knowledge in the mediator must be kept up-to-date, it will be wise for many mediators to place triggers or active demons into the databases [Buneman:79] [Stonebraker:86]. Now the mediators can be informed when the database, and, by extension, the real-world changes. The owner of the mediator should ensure that such changes are in time reflected in the mediator's knowledge base.

We consider, again justified by the analogy to human specialists, that a mediator is fundamentally trusted, but is inspectable when suspicions of obsoleteness arise. For instance, an assumption, say that well-to-do people buy big cars, may be used in the marketing mediator, but it is possible that over time this rule becomes invalid. We expect then that the base data be monitored for changes and that exceptions to database constraints will trigger information flow to the mediator. In a rule-base mediator the certainty factor of some rule can be adjusted [Esculier:89]. If the uncertainty exceeds a threshold, the mediator can advise its creator, the domain expert, to abandon this rule. The end-user need not get involved [Risch 89].

5. Related Concepts

We have shown throughout that the individual concepts underlying this architecture are not original. The problems that future information systems must address exist now, and are being dealt with in many specific situations. We do want to point out some significant relationships, as well as our own focus.

5.1 Independent actors and agents

There is an obvious corollary between the mediators proposed here and the concept of ACTORS [Hewitt:73]. However, a considerable constraint is imposed on our mediators — namely, they do not interact *intelligently* with each other — so that a hierarchy is imposed for every specific task. The reason for this constraint is to provide computational simplicity and manageability. An actor model can, of course, be used within a mediator to implement its computational task.

The network of connections within the global architecture means that distinct tasks can intersect at nodes within this information processing structure. The same mediator type may access distinct sets of data, and information from one data source can be used by distinct mediators.

5.2 Hierarchical task control

Automation requires an understanding of the control mechanisms that maintain balance and motivate progress or growth. To what extent networks of autonomous agents can be motivated, is unclear.

Rather than extrapolating into the unknown we define an architecture that we can conceptually manage today, and keep our minds open to extensions that are beyond todays conceptual foundations. We take again a cue here from Vannevar Bush, who could identify all units needed for the MEMEX, although its components were based on technology that did not exist in 1945.

Todays organizations depend greatly on hierarchical structures. Many information processing functions, now carried out in organizations, are performed by lower levels to obtain, aggregate, use, and rank information for the decision-making levels of management. Current development of the actors model [Hewitt:88] stresses the necessary match between actors and functions seen in real-world organizations. The ORG model also follows an anthropomorphic paradigm [Malone:87], focusing on a wider set of problem-solving interactions, and provides insights into distributed mediation. In all this research, concepts that model understood organizational practices form a basis, a notion that is broadly argued by [Litwin:89].

The relationships of users to mediators is organizationally similar to that presented in

the contract net model by [Smith:80], although the focus of a mediator is on partitioning for management and not on least-cost of computations. Contract net concepts may provide ideas for charging and accounting in this environment, an issue not discussed in this paper, but dealt with in the FAST [Cohen:89] project. The distributed cooperating agents of [Koo:88] deal with non-adversary contracting, a very desirable model for a future world.

5.3 Maintenance and learning

In effective organizations, lower levels of management involved in information-processing also provide feedback to superior layers.

The knowledge embodied in the mediators cannot be assumed to be static. Some knowledge may be updateable by human experts, but for active knowledge domains some automation will be important. Providing advice on inconsistencies between acquired data and assumed knowledge is the first step.

Eventually some mediators will be endowed with learning mechanisms. Feedback for learning may either come from performance measures or from explicit induction over the databases they manage [Blum:82] [Wilkins:87]. The trigger for learning is monitoring. Changes in the database can continuously update hypotheses of interest within the mediator. The validity of these hypotheses can be assessed by inspecting corollary observations in the view of the mediator [DeZegher:88]. The uncertainty of hypotheses can provide a ranking when an application task requests assessments.

5.4 Techniques

Mediators will embody a variety techniques now found both in freestanding applications and in programs that perform mediation functions. These programs are now often classified by the underlying scientific area rather than by its place in information systems.

Techniques from artificial intelligence

The nature of mediators is such that many techniques developed in AI will be employed. We expect that mediators will often use

1 Declarative approaches.
2 Capability for explanation.
3 Heuristic control of inference.
4 Pruning of candidate solutions
5 Evaluation the certainty of results

The literature on these topics is broad [Cohen:84]. Heuristic approaches are likely to be important because of the large solution spaces. Uncertainty computations are needed to deal with missing data and mismatched classes.

Techniques from logical databases

Since the use of mediators encourages a formal approach to the processing of data techniques being developed within logical and deductive database are likely to be equally important. As long as conventional DBMS do not provide well for recursive search the computations that achieve closure are likely to be placed into mediators [Beeri:87] [Ramakrishnan:89]. Since proper definition of the rules for stable and finite recursion is likely to remain difficult, these techniques are best managed by experts.

Another example of logical mediation is dealing with generalization, in order to return results of specified cardinality [Chaudhuri:90]. A frequent abstraction is to derive temporal interval represe4ntations from detailed event data. Here we also see a need to attach techniques that depend on domain semantics to the attributes in the database [Jajodia;90].

Techniques for efficient access

In this paper we do not focus on the efficiency of mediated access. We realize that interposing a layer in our information systems is likely to have a significant cost. We will argue that the flexibility and adaptability of a modular approach will overcome in time the inefficiencies induced by an inflexible, rigid structure. The partitioning of the tasks will also make research subtasks more managaable. It is today infeasible to carry out a really significant integrated system development in any single academic institution. Within specialized laboratories substantial systems can be developed and tested, but those are still hard to integrate into the world outside.

We can show some examples of systems research that focuses on overcoming processing bottlenecks:

1 Caching and materialized views and view indexes within the mediators [Roussopoulos:86] [Hanson:87] [Sheth:88]

2 Rule bases for semantic query optimization [King:84] [Chakravarthy:85]

3 data reorganization to follow dominant access demands [Fursin:86]

4 an ability to abandon ineffective object bindings [Gifford:88].

5 Privacy protection for sensitive data through interface modules [Cohen:88]

Sharing

Artificial intelligence, logical, and systems techniques can of course be shared among the mediators. Only knowledge specific to the application needs to be local to each mediator. In the framework which we present a variety of techniques can cooperate and compete to improve the production of information.

5.5 SoDs

The KBMS Project group at Stanford is in the process of formulating a specific form of mediators, called SoDs [Wiederhold:90]. A SoD provides a declarative structure for the domain semantics, suitable for an interpreter. We see multiple SoDs being used by an application executing a long and interactive transaction. We summarize our concepts here as one research avenue in providing components for the architecture we have presented.

Specific features and constraints imposed on SoDs are:

1 The knowledge should be represented in a declarative form

2 SoDs should have a well-formed language interface

3 They contain feature descriptions exploitable by the interpreter

4 They should be inspectable by the user applications

5 They should be amenable to parallel execution

6 They access the databases through relational views

7 During execution source and derived data objects are bound internally

8 They consider object sharing.

By placing these constraints on SoDs as mediators, we hope to be able to prove aspects of their behavior and interaction. Provable behavior is not only of interest to developers, but also provides a basis for prediction of computational efficiency.

However, the modularity of SoDs causes two types of losses:

1 Loss in power, due to limitations in interconnections

2 Loss in performance, due to relying on symbolic binding rather than on direct linkages.

We hope to offset these losses through gains obtained by having structures that enable effective computational algorithms. An implementation of a demonstration using frame technology is being expanded. The long-range benefit is of course that small, well-constructed mediators will enable knowledge maintenance and growth.

An Interface Language

Our research identifies the language problem as a major issue. For application access to SoDs we start from database concepts, where high-level languages have become accepted [WWHC+:89]. The SoD access language (SoDaL) will have the functional capabilities of SQL, plus iteration and test, to provide Turing machine level capability [Qian:89]. New predicates are needed to specify intelligent selection. Selection of desirable objects requires an ability to rank objects according to specified criteria. These criteria are understood by the SoD and are not necessarily predicates on underlying data elements, although for a trivial

SoD that may be true. These criteria are associated with result size parameters as 'Give me the 10 best X', where the 'best' predicate is a semantically overloaded term interpreted internally to a particular SoD.

The format of SoDaL is not envisaged to be user-friendly — after all, other subsystems will use the SoDs, not people. It should have a clear and symmetric structure which is machine friendly. It should be easy to build a user-friendly interface, if needed, on top of a capable SoDaL.

6. Limits and Extensions

The separation into layers reduces the flexibility of information transfer. Especially structuring the mediators into a single layer between application and data is overly simplistic. Precursors to general mediators already recognize hierarchies, as the contract net [Smith:80], and general interaction, as actors [Hewitt:73].

A desire to serve large-scale applications is the reason for the simple architecture presented here. To assure effective exploitation of the mediator concepts we propose to introduce complexity within their layer, and the associated processing cost, slowly, only as the foundations are established to permit efficient use.

Structuring mediators into hierarchies should not lead to problems. We already assumed that directory mediators could inspect other mediators. Inspection of lower-level mediators is also straightforward. Low-level mediators may only have database access knowledge, and understand little application domain semantics. On the other hand, high-level mediators can take on minor decision-making functions.

More complex is lateral information sharing among mediators. Some such sharing will be needed to maintain the lexicons that preserve object identity when distinct mediators group and classify data. Optimizers may restructure the information flow, taking into account success or failure with certain objects in one of the involved SoDs.

Fully general interaction among mediators is not likely to be supported at this level of abstraction. Just as human organizations are willing to structure and constrain interactions, even at some lost-opportunity cost, we impose similar constraints on the broad information systems we envisage.

Learning by modifying certainty parameters in the knowledge-base is relatively simple. Learning of new concepts is much more difficult, since we have no mechanisms that relate observations automatically to unspecified symbolic concepts. By depending initially fully on the human expert to maintain the mediator, then moving to some parameterization of rule priorities, we can gradually move to automated learning.

Efficiency is always a concern. Once derived data are available the need to analyze large databases is reduced, but the intermediate knowledge has to be maintained. Binding of the knowledge to provide the effect of compilation of queries is an important strategy. Work in rule-based optimizers and automatic creation of expert systems points in that direction [Schoen:88]. These tactics exacerbate the problems of maintaining integrity under concurrent use. Research into truth-maintenance is relevant here [Filman:88] [Kanth:88].

Requirements of data security may impose further constraints. Dealing with trusted mediators however, may encourage database owners to participate in information sharing to

a greater extent than they would if all participants would need to be granted file-level access privileges.

7. Summary

We envisage a variety of information processing modules residing in nodes along the data highways that advances in communication technology can now provide. A conceptual layering distinguishes nodes by function: decision-making exploration, information support by mediation of data by knowledge, and base data resources.

The mediation function, now seen in a variety of programs, is placed into explicit mediation modules, or *mediators*. For clarity we place all the mediators into one horizontal layer. These mediators are to be limited in scope and size to enable maintenance by an expert as well as inspection and selection by the end-user. Mediators are associated with the domain expert, but may be replicated and shipped to other network nodes to increase their effectiveness. Specialization increases the power and maintainability of the mediators and provides choices for the users.

Applications obtain information by dealing with abstractions supported by the mediators, and not by accessing base data directly. A language will be needed to provide flexibility in the interaction between the end-users' workstation and the mediators. We discuss the partitioning of artificial intelligence paradigms into pragmatics (at the user-workstation layer) and the formal infrastructure (in the mediation layer) further in [Wiederhold:90].

For query operations the control flow goes from the application to the mediator, who would interpret the query to plan optimal database access. The data would flow to the mediator, be aggregated, reduced, pruned, etc., and the results reported to the query originator. Multiple mediators serve an application with pieces of information from their subdomains.

The knowledge-based paradigms inherent in intelligent mediators indicate the critical role of artificial intelligence technology foreseen when implementing mediators. Knowledge sources of the mediation examples listed in Section 3 range from type information to business rules. The mediating modules we are developing, SoDs, stress structure and and declarative domain semantics for interpretation and inspection.

Mediators may be strengthened by having learning capability. Derived information may simply be stored in a mediator. Learning can also lead to new tactics of data acquisition and control of processing.

The intent of the architectural model is not to be exclusive and rigid. It is intended to provide a common framework under which many new technologies can be accommodated. As shown throughout, many existing concepts can be viewed as implementations of mediation. Especially of techniques listed in Section 5.4 have validity within and outside of this framwork, but will become more accessible and sharable. Now they are at best enbedded within intelligent application programs.

An important objective of the architecture is the ability to utilize a variety of information sources without demanding that they be brought into a common format and with only minimal requirements on their interfaces. Maintenance of the knowledge bases in the mediators requires specialization and a manageable size.

In a recent report the three primary issues to be addressed in knowledge-based systems were maintenance, problem modeling, and learning and knowledge acquisition [Buchanan+:89]. The architecture we presented here contributes to all three issues, largely by providing a partitioning that permits large systems to be composed from modules that are maintainable, that can implement specific submodels, and that access domain data for learning and knowledge validation.

8. Acknowledgement

References

[Adiba:81] Michel E. Adiba: "Derived Relations: A Unified Mechanism for Views, Snapshots and Distributed Data"; *VLDB 7*, Zaniolo and Delobel(eds), Sep.1981, pp.293–305.

[Atre:86] Shaku Atre: *Information Center: Strategies and Case Studies*, Vol. 1; Atre Int. Consultants, Rye NY, 1986.

[Barsalou:88] Thierry Barsalou: "An Object-based Architecture for Biomedical Expert Database Systems"; *SCAMC 12*. IEEE CS Press, Washington DC, November 1988.

[Basu:88] Amit Basu: "Knowledge Views in Multiuser Knowledge Based Systems"; *IEEE Data Engineering Conference 4*. Feb.1988, Los Angeles.

[Beeri:87] C. Beeri and R. Ramakishnan: "On the Power of Magic"; *ACM-PODS*, San Diego, Mar.1987.

[Blum:82] Robert L. Blum: *Discovery and Representation of Causal Relationships from a Large Time-Oriented Clinical Database: The RX Project*; Springer Verlag, Lecture Notes in Medical Informatics, no.19, 1982.

[Boehm:84] Barry W. Boehm: "Software Engineering Economics"; *IEEE Trans. Software Eng.*, Vol.10 No.1, Jan.1984, pp.4–21.

[Bull:87] M. Bull, R. Duda, D. Port, and J. Reiter: "Applying Software Engineering Principles to Knowledge-Base Development"; *Proc. Expert Systems and Business*

87, NY, Learned Information, Meadford NJ, Nov.1987, pp.27-37.

[Buchanan+:89] B.G. Buchanan, D. Bobrow, R. Davis, J. McDermott, and E.H. Short-liffe: "Research Directions in Knowledge-Based Systems"; Stanford KSL report 89-71, to appear in J. Traub (ed.): Annual Review of Computer Science.

[Buneman:79] O.P. Buneman and E.K. Clemons: "Efficiently Monitoring Relational Databases"; ACM Trans. on Database Systems, Vol.4 No.3, Sep.1979, pp.368-382.

[Bush:45] Vannevar Bush: "As We May Think"; Atlantic Monthly, Vol.176 No.1, 1945, pp.101-108.

[Callahan:81] M.V. Callahan and P.F. Rusch: "Online implementation of the CA SEARCH file and the CAS Registry Nomenclature File"; Online Rev., Vol.5 No.5, Oct.1981, pp.377-393.

[Chamberlin:75] D.D. Chamberlin, J.N. Gray, and I.L. Traiger: "Views, Authorization, and Locking in a Relational Data Base System"; Proc.1975 NCC, AFIPS Vol.44, AFIPS Press, pp.425-430.

[Chakravarthy:85] U.S. Chakravarthy, D. Fishmann and J. Minker; "Semantic Query Optimization in Expert Systems and Database Systems"; Expert Databases, Kersch-berg(ed), Benjamin Cummins, 1985.

[Chaudhuri:90] S. Chaudhuri: "Generalization and a Framework for Query Modification"; IEEE Data Engineering 6, Los Angeles, Feb. 1990.

[Cheeseman:85] Peter Cheeseman: "In Defense of Probability"; Proc. IJCAI, Los Angeles, 1985, pp.1002-1009.

[Chen:89] M.C. Chen and L. McNamee: "A Data Model and Access Method for Summary Data Management"; IEEE Data Engineering Conf. 5, Los Angeles, Feb.1989.

[Chiang:82] T.C. Chiang and G.R. Rose: "Design and Implementation of a Production Database Management System (DBM-2)"; Bell System Technical Journal, Vol.61 No.9, Nov.1982, pp.2511-2528.

[Cicchetti:89] R. Cicchetti, L.D. Lakhal, N. LeThanh, and S. Miranda: "A Logical Summary-data Model for Macro Statistical Databases"; DASFAA 1, Seoul Korea, KISS and IPSJ, Apr.1989, pp.43-51.

[Cohen:82] P.R. Cohen and E. Feigenbaum (eds.): The Handbook of Artificial Intelligence; Morgan Kaufman. 1982.

[Cohen:88] H. Cohen and S. Layne (editors) Future Data Management and Access. Workshop to Develop Recommendations for the National Scientific Effort on AIDS Modeling and Epidemiology; sponsored by the White House Domestic Policy Council, 1988.

31

[Dayal:83] U. Dayal and H.Y. Hwang: "View Definition and Generalization for Database Integration in Multibase: A System for Heterogeneous Databases"; *IEEE Trans. Software Eng.*, Vol.SE-10 No.6, Nov.1983, pp.628–645.

[DeMichiel:89] Linda DeMichiel: "Performing Operations over Mismatched Domains"; *IEEE Data Engineering Conference* 5, Feb.1989; *IEEE Transactions on Knowledge and Data Engineering*, Vol.1 No.4, Dec. 1989.

[DeZegher:88] I. DeZegher-Geets., A.G. Freeman, M.G. Walker. R.L. Blum and G. Wiederhold: "Summarization and Display of On-line Medical Records"; *M.D. Computing*, Vol.5 no.3, March 1988, pp.38-46.

[Downs:86] S.M. Downs, M.G. Walker, and R.L. Blum: "Automated summarization of on-line medical records"; IFIP *Medinfo'86*, North-Holland 1986, pp.800-804.

[Doszkocs:86] Tamas E. Doszkocs: "Natural Language Processing in Information Retrieval"; *J.Am.Soc.Inf.Sci.*, Vol.37 No.4, Jul.1986, pp.191–196.

[Esculier:89] Christian Esculier: *Introduction a la Tolerance Sematique*; PhD thesis, Un. Joseph Fournier, Grenoble, 1989.

[Filman:88] Robert E. Filman: "Reasoning with Worlds and Truth Maintenance in a Knowledge-Based Programming Environments"; *Comm. ACM*, Vol.31 No.4, Apr.1988, pp. 382–401.

[Fursin:86] Gennadiy I. Fursin: "Estimating and Decision Making Techniques in Database Design"; *Control Systems and Machines*, Kiev, No.1, Jan.1986, pp.141–155.

[Gifford:88] D.K. Gifford, R.M. Needham, and M.D. Schroeder: "The CEDAR File System"; *Comm. ACM*, Vol.31 no.3, Mar.1988, pp.288–298.

[Gray:86] Jim N. Gray: "An Approach to Decentralized Computer Systems"; *IEEE Trans. Software Eng.*, Vol.Se-12 No.6, Jun.1986, pp.684–692.

[Hammer:78] M. Hammer and D. McLeod: "The Semantic Data Model: A Modelling Machanism for Data Base Applications"; *Proc.. ACM SIGMOD* 78, Lowenthal and Dale(eds), 1978, pp.26–36.

[Hanson:87] Eric Hanson: "A Performance Analysis of View Materialization Strategies"; *Proc. ACM-SIGMOD* 87, May 1987.

[Hayes-Roth:84] Frederick Hayes-Roth: "The Knowledge-based Expert System, A tutorial"; *IEEE Computer*, Sep.1984, pp.11-28.

[Hewitt:73] C. Hewitt, P. Bishop, and R. Steiger: "A Universal Modular ACTOR Formalism for Artificial Intelligence"; *IJCAI* 3, SRI, Aug.1973, pp.235-245.

[Hewitt:88] Carl Hewitt: Knowledge Processing Organizations proposal; Nov.1988.

[Horvitz:86] E.J. Horvitz, D.E. Heckerman, and C. Langlotz: "A Framework for Compar-

ing Alternate Formalisms for Plausible Reasoning"; *Proc. AAAI-86*, 1986, pp.210–214.

[Humphrey:87] S.M. Humphrey, A. Kapoor, D. Mendez, and M. Dorsey: "The Indexing Aid Project: Knowledge-based Indexing of the Medical Literature"; NLM, LH-NCBC 87-1, Mar.1987.

[ISO/RDA:87] Working draft of ISO Remote Access Protocol; ISO/TC97/SC21 N 1926 (ANSI X3H2-87-210), Jul.1987.

[Kanth:88] M.R. Kanth and P.K. Bose: "Extending an Assumption-based Truth Maintenance System to Databases"; *IEEE CS Data Engineering Conference* 4, Feb.1988, LosAngeles.

[Kaplan:84] S.Jerrold Kaplan: "Designing a Portable Natural Language Database Query System"; ACM TODS, Vol.9 No.1, Mar.1984, pp.1–19.

[King:84] Jonathan J. King: *Query Optimization by Semantic Reasoning*; Univ.of Michigan Press, 1984.

[Koo:88] C.C. Koo and G. Wiederhold: "A Commitment-based Communication Model for Distributed Office Environments"; *ACM-COIS*, Mar.1988, pp.291-298.

[Lai:88] K-Y. Lai, T.W. Malone, and K-C. Yu: "Object Lens: A Spreadsheet for Cooperative Work"; *ACM Ttans. on Office Inf. Systems*, Vol.6 No.4, Oct.1988, pp.332–353.

[Litwin:86] W. Litwin and A. Abdellatif: "Multidatabase Interoperability"; *IEEE Computer*, Vol.19 No.12, Dec.1986, pp.10–18.

[Litwin:89] W. Litwin and N. Roussopolous: "A Model for Computer Life"; University of Maryland, Institute for Advanced Computer Studies, UMIACS-TR-89-76, July 1989.

[Maier:89] David Maier: "Why isn't there an Object-oriented Data Model"; *Information Processing* 89, Ritter (ed), IFIPS North-Holland 1989, pp.793-798.

[Malone:87] T.W. Malone, K.R. Grant, F.A. Turbak, S.A. Brobst, and M.D. Cohen: "Intelligent Information-Sharing Systems"; *Comm. ACM*, Vol.30 No.5, May.1987, pp.390–402.

[Mayo:89] J.S. Mayo and W.B. Marx, jr.: "Introduction: 'Technology of Future Networks'"; *AT&T Technical Journal*, Vol.68 No.2, Mar.1989.

[McCarthy:88] John L. McCarthy: "Knowledge Engineering or Engineering Information: Do We Need New Tools?"; *IEEE Data Engineering Conference* 4, Feb.1988, Los Angeles.

[McCune:85] B.P. McCune, R.M. Tong, J.S. Dean, and D.G. Shapiro: "RUBRIC: A System for Rule-based Information Retrieval"; *IEEE Trans. Software Eng.*, Vol.SE-

11 no,9. Sep.1985, pp.939-945.

[McIntyre:87] S.C. McIntyre and L.F. Higgins: "Knowledge Base Partitioning For Local Expertise: Experience In A Knowledge Based Marketing DSS "; *Hawaii Conf. on Inf. Systems* 20, Feb.1987.

[Ozsoyoglu:84] Z.M. Ozsoyoglu and G. Ozsoyoglu: "Summary-Table-By-Example: A Database Query Language for Manipulating Summary Data"; *IEEE Data Engineering Conf.* 1, Los Angeles, Apr.1984.

[Qian:89] XiaoLei Qian: "The Deductive Synthesis of Iterative Transaction"; PhD thesis, Stanford University, June 1989.

[Ramakrishnan:89] Raghu Ramakrishnan; "Conlog: Logic + Control"; Un. Wisconsin-Madison, CSD, 1989.

[Risch:89] Tore Risch: "Monitoring Database Objects"; *Proc. VLDB* 15, Amsterdam, Aug. 1989, Morgan Kaufmann Pubs.

[Roussopoulos:86] N. Roussopoulos and H. Kang: "Principles and Techniques in the Design of ADMS"; *IEEE Computer*, Vol.19 No.12, Dec.1986 pp.19-25.

[Saccà:86] D. Saccà, D. Vermeir, A. d'Atri, A. Liso, S.G. Pedersen, J.J. Snijders, and N. Spyratos: "Description of the Overall Architecture of the KIWI System"; *ESPRIT'85*, EEC, Elseviers, 1986, pp. 685-700.

[Sager:85] N. Sager, E.C. Chi, C. Friedman, and M.S. Lyman: "Modelling Natural Language Data for Automatic Creation of a Database from Free-Text Input"; *IEEE Database Engineering Bull.*, Vol.8,No.3, Sep.1985, pp.45-55.

[Schoen:88] E. Schoen, R.G. Smith, and B.G. Buchanan: "Design of Knowledge-based Systems with a Knowledge-based Assistant"; *IEEE Trans. Software Eng.*. Vol.14 No.12, Dec.1988, pp.1771-1791.

[Sewell:86] W. Sewell and S. Tietelbaum: "Observations of End-user Online Searching Behavior over Eleven Years"; *J.Am.Soc.Inf.Sci.*, Vol.37 No.4, Jul.1986, pp.234-245.

[Shannon:48] C.E. Shannon and W. Weaver: *The Mathematical Theory of Computation*;1948, reprinted by The Un.Illinois Press, 1962.

[Shen:85] Sheldon Shen: "Design of a Virtual Database"; *Information Systems.*. Vol.10 No.1, 1985, pp.27-35.

[Sheth:88] A.P. Sheth, J.A. Larson, and A. Cornellio: "A Tool for Integrating Conceptual Schemas and User Views"; *IEEE Data Engineering Conference* 4. Feb.1988. Los Angeles.

[Smith:80] R.G. Smith: "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver"; *IEEE Ttans. Computers*, Vol.C-29 No.12,

Dec.1980, pp.1104–1113.

[Smith:81] J.M. Smith et al: "MULTIBASE — Integrating Heterogeneous Distributed Database Systems"; *Proc.NCC*, AFIPS Vol.50, Mar.1981. pp.487–499.

[Stonebraker:85] M. Stonebraker, D. DuBourdieux, and W. Edwards: "Triggers and Inference in Database Systems"; Brodie, Mylopoulos, and Schmidt (eds) *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, Springer, Feb.1986, pp.297–314.

[Stonebraker:88] Michael Stonebraker: "Future Trends In Database Systems"; *IEEE Data Engineering Conf.* 4, Feb.1988, Los Angeles.

[Summit:67] R.K. Summit: "DIALOG: An Operational On-Line Reference Retrieval System"; *ACM Nat. Conf.* 22, 1967, pp.51–56.

[Tanenbaum:88] Andrew S. Tanenbaum: *Computer Networks*, 2nd ed; Prentice-Hall, 1988.

[Tsichritzis:87] D. Tsichritzis, E. Fiume, S. Gibbs, and O. Nierstrasz: "KNOs: Knowledge Acquisition, Dissemination and Manipulation Objects"; *ACM Trans. on Office Inf. Sys.*, Vol.5 No.1, Jan.1987, pp.96-112.

[Waldrop:84] M.Mitchell Waldrop: "The Intelligence of Organizations"; *Science*, Vol.225 No.4667, Sep.1984, pp.1136–1137.

[Wetherbe:85] J.C. Wetherbe and R.L. Leitheiser: "Information Centers: A Survey of Services, Decisions, Problems, and Successes"; *Inf. Sys. Manag.*, Vol.2 No.3, 1985, pp.3–10.

[Wick:89] M.R. Wick and J.R. Slagle: "An Explanation Facility for Today's Expert Systems"; *IEEE Expert*, Spring 1989, pp .26–36.

[Wiederhold:83] Gio Wiederhold: *Database Design*: McGraw-Hill, 1983.

[Wiederhold:86B] Gio, Wiederhold: "Knowledge versus Data"; Chapter 7 of Brodie, Mylopoulos, and Schmidt (eds.) 'On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies' Springer Verlag, June 1986, pages 77–82.

[Wiederhold:86] Gio Wiederhold: "Views, Objects, and Databases"; *IEEE Computer*, Vol.19 No.12, December 1986, Pages 37–44.

[Wiederhold:87] G. Wiederhold and X-L. Qian: "Modeling Asynchrony in Distributed Databases"; *IEEE Data Engineering Conference* 3, Los Angeles, Feb. 1987.

[Wiederhold:90] G. Wiederhold, P. Rathmann, T. Barsalou, B-S. Lee, and D. Quass: "Partitioning and Combining Knowledge"; *Information Systems*, to appear 1990.

[WWBD:86] G.C.M. Wiederhold, M.G. Walker, R.L. Blum, and S.M. Downs: "Acquisition of Knowledge from Data"; *Proc. ACM SIGART ISMIS*, Oct.1986, pp.74–84.

[WWHC+:89] G.CM Wiederhold, M.G. Walker, W. Hasan, S. Chaudhuri, A. Swami, S.K. Cha, X-L. Qian, M. Winslett, L. DeMichiel, and P.K. Rathmann: "KSYS: An Architecture for Integrating Databases and Knowledge Bases"; in Amar Gupta (ed.), *Heterogenous Integrated Information Systems*, IEEE Press, 1989.

[Wilkins:87] D.C. Wilkins, W.J. Clancey, and B.G. Buchanan: "Knowledge Base Refinement by Monitoring Abstract Control Knowledge"; Stanford, TR. STAN-CS-87-1182, Aug.1987.

Monotonic Combinations of Non-Monotonic Theories*

Peter K. Rathmann
Computer Science Dept.
Stanford University
Stanford, CA 94305

February 23, 1990

Abstract

If a logical theory with nonmonotonic properties is partitioned for ease of maintenance, it is possible that a conclusion derived from one of the partitions will not be supported by the theory as a whole.

This paper argues that such behavior is undesirable and that large knowledge bases should be built so that local deduction is globally sound.

A sufficient condition is presented which ensures such soundness for partitioned knowledge bases based on prioritized circumscription.

1 Introduction

Many of us remember evenings spent with Perry Mason, when witnesses swore to "tell the truth, the WHOLE truth, and nothing but the truth". Imagine hearing the following exchange:

37

Q: Did Mr. X see you take his car?
A: Yes.
Q: And he didn't object?
A: No, he didn't say a thing.

If later we learn that Mr. X was bound and gagged at the time in question, we may feel deceived, even though neither of the answers given is actually false.

However, in designing knowledge bases, we sometimes can't "tell the whole truth". A large knowledge base must be built in partitions if it is to be maintainable, and users often want only a simplified view of a complex system. In other words, both the builders and users of a system are often dealing with only a partial truth. How can we prevent it from being a misleading one?

This paper presents tools and conditions for ensuring that views and partitions of knowledge bases are not misleading, even when the knowledge base allows nonmonotonic inference.

2 Partitioning and Nonmonotonic Reasoning

The basic motivation for this work comes from the problem of assembling the next generation of large knowledge bases. Such knowledge bases will need to be flexible enough to evolve over time and to serve diverse groups of users. This maintenance requirement argues strongly for a partitioned architecture, since partitioning is our most successful strategy for dealing with complexity.

In addition, such knowledge bases will need to support some form of nonmonotonic reasoning. Even if a system does not specifically bill itself as employing a formalized version of nonmonotonic reasoning, nonmonotonicity is unavoidable if a system needs to deal with uncertainty, to weigh evidence, or to employ any sort of heuristic. In any of these cases, the system is drawing conclusions which may later be retracted in light of new evidence.

However, problems arise if both partitioning and nonmonotonic reasoning are used together in the same system. If a human or mechanical agent restricts its attention to a single partition, and if that partition omits something critical, the agent may draw (nonmonotonic) conclusions which are

contradicted by other partitions. This is dangerous and results in awkward semantics for the global knowledge base. It may seem that we are forced to disallow any inference mechanism which does not have access to the entire knowledge base, but in disallowing such local deduction, we forego many of the benefits of partitioning.

We argue for an different approach. We allow local inference, but design the knowledge base so that nonmonotonic deductions are limited to those which are sound globally. This ensures that the process by which we combine partitions into a global theory is monotonic, even if the theories themselves are not. Specifically, we will be adjusting the priority schemes of prioritized circumscription to assure such a sound combination process.

3 Definitions and Background

3.1 Combining Theories

First, let us make at least a preliminary definition of what it means to integrate component theories. Our definition is perhaps the simplest possible; the combination of two theories is the set theoretic union of their sentences. So, if A and B are theories, i.e., sets of sentences in some form of logic, then the combination of them is given by $T = A \cup B$. T is then the global theory, and A and B are its components, or subtheories. This definition is essentially syntactic – there is no requirement that any of the above theories be closed under logical deduction, and in general, they will not.

3.2 Circumscription

Circumscription [BS85, EMR85, Eth88, Lif86, McC80, McC86, Per87, Sho87] seeks to solve the problem of common-sense reasoning by "preferring" certain models of a theory T to others. More precisely, circumscription picks out those models of a theory that are minimal with respect to some partial order on models [Shoham 87]. Thus, a circumscriptive partial order encodes our intuitions about which of the logically plausible alternative models of T are the most "normal" and reasonable.

The results of this paper are all given in terms of the most common version of circumscription, in which the partial order on models is based on

set inclusion of predicate extensions. For some applications, especially those involving equality, it is useful to use an alternate version of circumscription in which the partial order is based on the presence or absence of model homomorphisms [RW88, RW89]. Results very similar to those presented in this paper hold for this alternate version. For a presentation of these results, see [Rat90].

3.3 Prioritized Signatures

In circumscription, the preference criteria for models, including holding predicates fixed, allowing them to vary, and minimizing with priorities are all a property of the signature alone, and do not depend on the theory. We can take advantage of this, and define as a notational convenience the concept of a *prioritized signature*. A prioritized signature includes the function and predicate symbols. The predicate symbols are divided into three non-overlapping subsets, corresponding to those predicates held fixed, those allowed to vary, and those minimized. In addition, there is a partial order imposed on the subset of minimized predicates, corresponding to the priority with which the predicates are to be minimized. We will write $Circ(T, \Omega)$ for $Circ(T, A < B)$, if we have previously defined Ω to be a prioritized signature where $A < B$. This notation will be convenient when we manipulate and compare different priority schemes.

3.4 Comparing Theories with Different Signatures

In ensuring soundness of local deduction, we will need to be able to compare the semantics of the partition to the semantics of the global theory. In making such a comparison, we need to deal with the possibility that the partition will have a different (smaller) signature than the global theory. Normally, when we want to see if one theory is stronger than another, we can rely on a simple semantic definition. If the models satisfying a theory or statement T_1 are a subset of those satisfying T_2, we say that T_1 implies or semantically entails T_2. However, when T_1 and T_2 are defined with different underlying signatures, this definition is not applicable, since the models of T_1 have a different structure from the models of T_2. Here the thing to do is to make the model theory follow the proof theory. If a theory does not mention a

particular predicate, any syntactically allowed extension of that predicate is consistent with the theory.

In comparing circumscribed theories with different signatures, we use the same basic condition. A predicate not mentioned in the signature of a circumscribed theory should have an unconstrained extension. It turns out that such an unconstrained extension is equivalent to including the predicate in the signature, but holding it fixed in any circumscription.

Lemma 1 *If signature Ω' is a subset of signature Ω, but all predicates which appear in Ω but not Ω' are held fixed in Ω, then, by the definitions of this section $Circ(T, \Omega)$ is logically equivalent to $Circ(T, \Omega')$.*

Proof: First of all, it is syntactically implicit that T can not mention any predicate from $\Omega - \Omega'$. Otherwise, the sentence $Circ(T, \Omega')$ would not make any sense.

A rigorous proof would be quite lengthy, here we will give a plausability argument. Let M be a minimal model of $Circ(T, \Omega')$. We can extend M with any syntactically allowed extensions for the predicates of $\Omega - \Omega'$. Since T does not mention these extra predicates, the extended model, $\overline{M}$, still satisfies T. Moreover, it must be a minimal model of $Circ(T, \Omega)$, since it is only comparable with other models with these same extensions of the extra fixed predicates. In effect, each different extension of the extra (fixed) predicates creates a new independent copy of the partial order in $Circ(T, \Omega')$. Since M is minimal in $Circ(T, \Omega')$, $\overline{M}$ is minimal in $Circ(T, \Omega)$. In other words, the extended model is minimal, precisely if the original one was.

To get a minimal model, of $Circ(T, \Omega)$, we take a minimal model of $Circ(T, \Omega')$, and extend it with any extension at all of the new predicates. Thus, $Circ(T, \Omega)$ is a conservative extension of $Circ(T, \Omega')$, which means that the proof theories will be equivalent.

□

4 Using Priorities

Prioritized circumscription gains considerable control over the semantics of a theory by specifying the order in which predicates are to be minimized. We can make use of this control by adjusting the priorities within the partition,

with the goal of limiting local inference to that which is sound globally. We shall approach this goal in stages, first proving some more specialized results. Our first condition shows that holding a predicate fixed weakens the theory.

Theorem 1 *Let T be a theory. Let Ω_f and Ω be prioritized signatures for T, differing only in that Ω_f holds a predicate Q fixed, while Ω minimizes Q or allows Q to vary. Then,*

$$Circ(T, \Omega) \Rightarrow Circ(T, \Omega_f)$$

Proof: We consider first the special case where all minimized predicates in the signature Ω have the same priority. In this case, the models which satisfy $Circ(T, \Omega)$ and $Circ(T, \Omega_f)$ are those minimal in the respective partial orders. The partial order on models determined by Ω_f is a suborder of that determined by Ω, since holding a predicate fixed just adds an extra condition for two models to be comparable. Hence any model minimal by the partial order of Ω must also be minimal in the order determined by Ω_f, and $Circ(T, \Omega) \Rightarrow Circ(T, \Omega_f)$.

Now we turn to the general case, where predicates may be minimized with differing priorities. Let $\Omega = (F, P_1 < P_2 < \ldots < P_n, V)$ where F is a set of fixed predicates, $P_1, \ldots, P_n$ are sets of minimized predicates with those in P_1 having greater priority than those in P_2, and so on, and V is a set of predicates which are allowed to vary.

Prioritized circumscription is defined is terms of a conjunction of unprioritized circumscriptions, in particular,

$$Circ(T, \Omega) = \bigwedge_{i=1}^{n} Circ(T, \Omega_i),$$

where the Ω_i's are defined by

$$
\begin{aligned}
\Omega_i = \quad & F \text{ fixed} \\
& P_1, \ldots, P_i \text{ minimized} \\
& P_{i+1}, \ldots, P_n \text{ allowed to vary} \\
& V \text{ allowed to vary}
\end{aligned}
$$

The rest of the proof will depend on whether Q, the predicate being held fixed, was originally fixed, minimized, or allowed to vary.

42

Case 1 – Q is fixed in Ω

Formally, this case is not allowed in the statement of the theorem. However, if Q is already held fixed, holding it fixed does not change anything, and the theorem is trivially true.

Case 2 – Q is allowed to vary in Ω

In this case, $Q \in V$, and we decompose the prioritized circumscription into

$$Circ(T, \Omega_f) = \bigwedge_{i=1}^{n} Circ(T, \Omega_{fi}),$$

where the Ω_{fi}'s are defined by

$$\Omega_{fi} = \begin{array}{l} F \cup \{Q\} \text{ fixed} \\ \bigcup_{j \le i} P_j \text{ minimized} \\ \bigcup_{j > i} P_j \text{ allowed to vary} \\ V - \{Q\} \text{ allowed to vary} \end{array}$$

For each i, Ω_{fi} differs from Ω_i only in that Ω_{fi} holds Q fixed , while Ω_i allows Q to vary. So, by the result for the unprioritized case,

$$Circ(T, \Omega_i) \Rightarrow Circ(T, \Omega_{fi})$$

holds for each i. Substituting, we get

$$\bigwedge_{i=1}^{i=n} Circ(T, \Omega_i) \Rightarrow \bigwedge_{i=1}^{i=n} Circ(T, \Omega_{fi}),$$

or equivalently,

$$Circ(T, \Omega) \Rightarrow Circ(T, \Omega_f),$$

For the case where Q is allowed to vary.

Case 3 – Q is minimized in Ω

If Q is a minimized predicate, it means $Q \in P_k$ for some k, $1 \le k \le n$. Then, as before, we decompose the prioritized circumscription into a conjunction of circumscriptions, where we define the Ω_{fi}'s for $i \le k$, by

$$\Omega_{fi} = \begin{array}{l} F \cup \{Q\} \text{ fixed} \\ \bigcup_{j \le i} P_j \text{ minimized} \\ \bigcup_{j > i} P_j - \{Q\} \text{ allowed to vary} \\ V - \{Q\} \text{ allowed to vary} \end{array}$$

43

and for $i > k$, by

$$\Omega_{fi} = \begin{array}{l} F \cup \{Q\} \text{ fixed} \\ \bigcup_{j \le i} P_j - \{Q\} \text{ minimized} \\ \bigcup_{j > i} P_j \text{ allowed to vary} \\ V \text{ allowed to vary.} \end{array}$$

For either case, we know that fixing a predicate weakens a circumscription, so $Circ(T, \Omega_i) \Rightarrow Circ(T, \Omega_{fi})$ for each i. As above for case 2, we substitute each implication into the definition of prioritization to get that $Circ(T, \Omega) \Rightarrow Circ(T, \Omega_f)$, where Q is a minimized predicate.

Thus, because each of the three possible cases results in a weaker theory, we conclude that fixing a predicate of a prioritized theory weakens the theory.

$\square$

So, holding a predicate fixed weakens the circumscribed theory. In fact, if we hold all predicates fixed, our next lemma shows that circumscription adds nothing to the theory.

Lemma 2 *If T is a theory and Ω_f is a signature in which all predicates are held fixed,*

$$T \Rightarrow Circ(T, \Omega_f)$$

Proof: With all predicates fixed, no model is preferred to any other, so all models which satisfy T are minimal, and hence satisfy $Circ(T, \Omega_f)$. $\square$

Lemma 2 shows that if we hold all predicates fixed in the view or component, circumscription is equivalent to ordinary first-order logic. First order logic is monotonic, and so deduction within a component is sound. This provides us with a first, admittedly drastic way of assuring that views and components are not misleading. If we take a component (subset) of a theory, and restrict the circumscription of the component to hold all predicates fixed, any deduction we make in the component will be sound in the context of the global theory.

So far we have been varying the prioritized signature, i.e., the recipe for circumscription, and seeing what this does to the strength of the circumscribed theory. In our next lemma, we take a complementary approach. We keep the signature constant, and change the theory. In particular, we will see that we can drop any sentence involving only the fixed predicates of a circumscribed theory, without losing soundness.

Lemma 3 *Let T_1 and T_g be theories, such that $T_1 \subseteq T_g$, and let Ω be a signature in which all the predicates from sentences in $T_g - T_1$ are held fixed. Then $Circ(T_g, \Omega) \Rightarrow Circ(T_1, \Omega)$*

Proof: Let M be a minimal model of $Circ(T_g, \Omega)$. Is it possible that M is not a minimal model of $Circ(T_1, \Omega)$? Let us assume it is not, and derive a contradiction. The model $M \models T_1$, since $T_1 \subseteq T_g$. Thus, the only way M can be not minimal in $Circ(T_1, \Omega)$ is if $\exists M' < M$ and $M' \models T_1$. Now, $M' \not\models T_g$, since M is minimal in $Circ(T_g, \Omega)$. Hence there must be a sentence $S \in T_g - T_1$, such that $M' \not\models S$. But $M' < M$ in Ω and Ω holds all predicates mentioned in S fixed. However, we now look at the sematic definition of truth for first order logic, given in, for example, [End72]. The important thing to note about the definition is that it does not refer to any predicates mentioned in the sentence. This means that it is impossible for one model to satisfy a sentence S, and for another model with the same universe, and the same extensions on every predicate mentioned in S not to satisfy S. □

In order to state a more general condition, we shall define some additional terminology. Let us have a global theory T_g and a component theory $T_1 \subseteq T_g$. We call a predicate *full* in T_1 if every mention of it in T_g also appears T_1. Conversely, we call a predicate *partial* if it is mentioned in sentences from $T_g - T_1$. Finally, let Ω_1 be a prioritized signature for T_1 and let Ω_g be a prioritized signature for T_g. Then we say Ω_1 is a *consonant* with Ω_g if

1. Ω_1 is a subsignature of Ω_g. This means that every sort, predicate and function symbol of Ω_1 also occurs in Ω_g. Further, if a symbol appears in both Ω_1 and Ω_g, it has the same arity and type in each.

2. The ordering on the predicates of Ω_1 is a subordering of that on the predicates of Ω_g. In other words, if R and R' are minimized predicates in Ω_1, such that $R < R'$ in $struct(\Omega_1)$, R and R' are also minimized in Ω_g, and $R < R'$ in Ω_g.

3. If a predicate is fixed in Ω_1, it may be fixed or minimized in Ω_g, but not varying.

Now, we come to the main result of this paper, which ties all of the previous results together to yield a useful condition for when deduction in a circumscribed component theory will be sound relative to the global theory.

Theorem 2 *If T_1 is a subtheory of T_g, with prioritized signatures Ω_1 and Ω_g respectively, and Ω_1 is consonant with Ω_g, and all predicates of T_1 which are not full in T_1 are held fixed in Ω_1, then*

$$Circ(T_g, \Omega_g) \Rightarrow Circ(T_1, \Omega_1).$$

Proof:

The proof proceeds in stages, showing that

$$Circ(T_g, \Omega_g) \Rightarrow Circ(T_g, \overline{\Omega_1}) \Rightarrow Circ(T_1, \overline{\Omega_1}) \equiv Circ(T_1, \Omega_1),$$

where $\overline{\Omega_1}$ is an intermediate signature obtained by extending Ω_1 with any predicates present in Ω_g but not Ω_1. These added predicates are held fixed in $\overline{\Omega_1}$.

The left hand implication is a generalization of theorem 1. Again we must decompose the prioritized circumscription into a conjunction of unprioritized circumscriptions, and apply the theorem to each conjunct.

The definition of prioritized circumscription states that,

$$Circ(T_g, \overline{\Omega_1}) = \bigwedge_i Circ(T_g, \overline{\Omega_{1i}}),$$

where each $\overline{\Omega_{1i}}$ is defined by

$$
\begin{aligned}
\overline{\Omega_{1i}} = \quad & F \text{ fixed} \\
& \textstyle\bigcup_{j \leq i} P_j \text{ minimized} \\
& \textstyle\bigcup_{j > i} P_j \text{ allowed to vary} \\
& V \text{ allowed to vary}
\end{aligned}
$$

where F is the set of predicates held fixed, and the P_i's are sets of predicates making up the priority order of $\overline{\Omega_1}$. Any predicate within a particular set P_i is equivalent in the priority order to any other in predicate in P_i, it is greater in the priority order than a predicate from any P_j, for $j < i$ and lower in the priority order than any predicate from P_j, for $j > i$.

We shall prove that $Circ(T_g, \Omega_g)$ implies each of the conjuncts which make up $Circ(T_g, \overline{\Omega_1})$. Consider a particular one of these conjuncts, $Circ(T_g, \overline{\Omega_{1i}})$. From among the predicates which are minimized in $\overline{\Omega_{1i}}$, we select a particular predicate Q which is maximal *in the priority ordering of* Ω_g. Q must also be maximal in the priority scheme of $\overline{\Omega_1}$, since the predicate ordering of $\overline{\Omega_1}$ is a

46

subordering of that in Ω_g. This Q is in some P'_k, one of the sets of predicates determining the priority scheme of Ω_g.

Now, the following facts are true:

1. $Circ(T_g, \Omega_g) \Rightarrow Circ(T_g, \Omega_{gk})$. This follows from the definition of prioritized equality circumscription, since the right hand side is one of the conjuncts of that definition.

2. Ω_{gk} minimizes every predicate minimized in $\overline{\Omega_{1i}}$, and allows to vary any predicate allowed to vary in $\overline{\Omega_{1i}}$. This follows from the definition of consonant, and the fact that Q was maximal among the minimized predicates of $\overline{\Omega_{1i}}$.

3. Some predicates fixed in $\overline{\Omega_{1i}}$ may be minimized or allowed to vary in Ω_{gk}. However, repeated application of theorem 1 yields the conclusion $Circ(T, \Omega_{gk}) \Rightarrow Circ(T, \overline{\Omega_{1i}})$.

Combining (1) and (3), we find that for any i,

$$Circ(T_g, \Omega_g) \Rightarrow Circ(T_g, \overline{\Omega_{1i}}).$$

taking the conjunction of these for all i we find

$$Circ(T_g, \Omega_g) \Rightarrow Circ(T_g, \overline{\Omega_1}),$$

which is what we wanted to verify the lefthand implication.

The middle implication is follows from repeated application of lemma 3 and the right hand equivalence follows from lemma 1.

These stages, taken together, prove the theorem.

$\square$

5   Using the Theorem

Now we shall give a flavor of how this theory can be used by working through example of partitioning, and how it affects non-monotonic inference within a component.

Consider a global knowledge base T_g of employee information which is partitioned into two components. The component T_1 contains records for

47

exempt (salaried) employees, while T_2 contains records for non-exempt employees. These components partition the knowledge base, i.e., every employee is either exempt or non-exempt and no one is both.[1]

In both partitions, the predicate is called *Employee*, and takes the same arguments. Since neither partition has a complete set of the ground instances of *Employee*, neither can use a closed world assumption. In the terminology of this paper, *Employee* is not full in either partition, and must be held fixed in any circumscription within the component.

However, we can regain a partial closed world assumption if we adjust the signature and theory to reflect the partitioning more closely. We rename the predicate used in T_1 to be Emp_1, and the predicate used in T_2 to be Emp_2. The statements in T_1 will be terms of Emp_1, and since Emp_1 will be full in T_1, we can safely circumscribe, minimizing Emp_1. The situation is symmetric for T_2. We can connect the new predicates to the global predicate *Employee* with the rules:

$$Emp_1(x, \ldots) \Rightarrow Employee(x, \ldots)$$
$$Emp_2(x, \ldots) \Rightarrow Employee(x, \ldots)$$

We will assume that the intended global semantics is for all three predicates, Emp_1, Emp_2, and *Employee* to be minimized in a circumscription. If *both* partitions contain the above rules, and these rules are the only sentences which mention *Employee*, then *Employee* will be full in both partitions, since each will have a complete set of all sentences mentioning *Employee*. Both Emp_1 and *Employee* are full in T_1, while Emp_2 is partial in T_1. Therefore, using theorem 2 it is sound to circumscribe T_1, minimizing Emp_1 and *Employee* and holding Emp_2 fixed.

Note that this is still a partial open world assumption for the predicate *Employee*, since Emp_2 is held fixed, (i.e., we have an open world assumption for Emp_2) and any element of Emp_2 must also be an element of *Employee*. However, in this case, the partitioning is based on a known semantics – the difference between exempt and non-exempt employees. For example, if we also have a rule that all managers are exempt, we can answer a query such as "What is the total payroll expenditure for managers?", entirely within T_1. Thus, there are useful nonmonotonic inferences which are guaranteed to be sound within the partition.

---

[1]Assume that this is a globally maintained integrity constraint.

6 Completeness

The main results of this paper have been concerned with safety – that deduction within a view should be sound. However, soundness is not all we are looking for. If it were, we could have stopped when we realized that soundness can be assured by avoiding nonmonotonic inference entirely. Instead, we want some assurance that our methods are, if not complete, at least powerful enough for our applications.

In one sense, our results are the most powerful possible. If we stay within the framework of adjusting the priority scheme to weaken inference within a view, it is easy to find examples where it is not sound to minimize partial predicates or allow them to vary.

On the other hand, our definition of full is not the most general possible. Perhaps, this should not be too surprising, since it is a simple syntactic condition on the intensions of theories, and it ensures a rather complex semantic property.

We can weaken the definition of full by requiring, not that every every mention of the predicate in the global theory actually occur in the partition, but only that the partition contain every sentence in some subset from which all the sentences in the global theory can be monotonically derived. This generalization can make a difference if the global knowledge base contains many redundant sentences, perhaps because it caches derived information. As long as the derived sentences are monotonically derivable from the subset, the partition has no semantic need to include and maintain copies of such derived sentences.

Verifying that a predicate is full by this generalized definition requires deduction in first order logic, which can be very expensive.

In addition, some global sentences, even if they are not redundant, may not restrict the possible extensions of a mentioned predicate. A common example is provided by view definitions, which may mention base predicates, but only to derive new predicates from them. Recognizing such sentences in general is likely to be difficult, although no doubt useful special cases exist.

It will probably take more extensive experience with applications to determine what is an appropriate balance between generality and efficiency of the condition for full.

7 Conclusion

We have argued that partitioned knowledge bases should be organized so that even the nonmonotonic conclusions derivable from a partition should also be derivable from the knowledge base as a whole. This soundness of local deduction is useful for comprehensibility and maintenance, and may also affect efficiency as well. For example, it makes it possible to confidently apply efficient special purpose inference techniques, even if they only apply to a limited partition of a knowledge base.

Our scheme for ensuring the soundness of local deduction directly applies only to knowledge bases which represent knowledge as a circumscribed theory, but we expect that the basic principles can be used as a guideline for other knowledge representations as well.

References

[BS85]   G. Bossu and P. Siegel. Saturation, nonmonotonic reasoning and the closed-world assumption. *Arificial Intelligence*, 25:13–63, 1985.

[EMR85]   D. W. Etherington, R. E. Mercer, and R. Reiter. On the adequacy of predicate circumscription for closed-world reasoning. *Computational Intelligence*, 1:11–15, 1985.

[End72]   Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, 1972.

[Eth88]   David W. Etherington. *Reasoning With Incomplete Information.* Morgan Kaufmann, Los Altos, California, 1988.

[Lif86]   Vladimir Lifschitz. Pointwise circumscription. In *Proceedings, AAAI-86, Philadelphia, PA*, 1986. Also appears in *Readings in Nonmonotonic Reasoning*, M. L. Ginsberg, ed., Morgan Kaufmann, Los Altos, California, 1987.

[McC80]   John McCarthy. Circumscription—a form of non-monotonic reasoning. *Arificial Intelligence*, 13(27), 1980. Also appears in *Read-*

ings in Nonmonotonic Reasoning, M. L. Ginsberg, ed., Morgan Kaufmann, Los Altos, California, 1987.

[McC86]   John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Arificial Intelligence*, 28, 1986. Also appears in *Readings in Nonmonotonic Reasoning*, M. L. Ginsberg, ed., Morgan Kaufmann, Los Altos, California, 1987.

[Per87]   D. Perlis. Circumscribing with sets. *Arificial Intelligence*, 31, 1987.

[Rat90]   Peter K. Rathmann. *Nonmonotonic Semantics for Partitioned Knowledge Bases*. PhD thesis, Stanford University, 1990. in preparation.

[RW88]   Peter K. Rathmann and Marianne Winslett. Circumscribing equality. Technical Report TR UIUCDCS-R-88-1473, University of Illinois, 1988.

[RW89]   Peter K. Rathmann and Marianne Winslett. Circumscribing equality. In *Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan*, pages 468–473, 1989.

[Sho87]   Yoav Shoham. A semantical approach to nonmonotonic logics. In *Proceedings of the Tenth IJCAI, Milan, Italy*, pages 388–392, 1987. Also appears in *Readings in Nonmonotonic Reasoning*, M. L. Ginsberg, ed., Morgan Kaufmann, Los Altos, California, 1987.

[WRR+90]   Gio Wiederhold, Peter Rathmann, Tore Risch, Byung Suk Lee, Surajit Chaudhuri, Thierry Barsalou, Kincho H. Law, and Dallan Quass. A mediator architecture for abstract data access. Technical report, Stanford University, 1990.

Partitioning and Composing Knowledge

Gio Wiederhold, Peter Rathmann, Thierry Barsalou, Byung Suk Lee, and Dallan Quass

Stanford University

Abstract

This paper argues for an approach which places the management of large knowledge bases into a comprehensive, engineering-oriented framework, and reports on an initial demonstration of these concepts. The underlying concepts are well-recognized as being effective in many areas of science:

1 Partitioning of the knowledge into manageable segments.

2 Rules for the composition of these segments.

3 A language to provide access to these segments, control their composition, and provide the power of the system in a flexible and clear way.

The motivation for this research is to deal with problems that are beginning to occur in large knowledge-based systems. As current developments of such systems lead to further growth, we foresee that their management needs will exceed the capabilities of the existing system infrastructure. In particular we find that in the past issues related to knowledge maintenance have been ignored. Maintenance of knowledge-bases is critical if the systems are to persist.

1. Introduction

Problems of knowledge maintenance in large knowledge-based systems motivate our research. Today these problems are evident in only some instances, but will become more prevalent as knowledge-based systems grow in scope and depth, and last beyond the lifetime of a PhD thesis. Some researchers from the AI community have looked towards database technology to help in dealing with issues of size and update management [Kerschberg]. Database systems have focused on simple structuring and normalization to deal with large bodies of information, and do not deal well with the complexities of structures needed to represent knowledge.

We are using concepts from database research here as well, but must be very careful in intermingling database and knowledge-base representations. We need to avoid creating a combination with the weaknesses of the two fields, rather than the strengths. Future information systems will benefit from distributed knowledge sources and distributed computation. An architecture to deal with future systems must consider the technological opportunities that are becoming available. We see these systems supporting decision-makers through a two-phase process:

1 Locating and selecting relevant factual data and aggregating it according to the decision alternatives.

2 Processing and reducing the data so that the number of alternative choices to be decided among is small, and the parameters for each choice are aggregated to a high conceptual level.

Today most of these support tasks are carried out by human experts who mediate between the database and the decision maker. For many tasks in medicine, warfare, emergency relief, and other areas requiring rapid actions, dependence on human intermediaries introduces an intolerable delay. Future information systems will increasingly need to use automatic mediators to speed up these support processes [Wiederhold89].

The databases, the mediators, and the applications will all reside on nodes of powerful networks. The end-users will always have computers available to serve their specific tasks. We refer to those machines as application workstations, although they may at times be large and powerful processors.

1.1 Large Knowledge Bases

We expect that future information systems will contain large quantities of knowledge in order to support high-level decision-making tasks [Feigenbaum88; Methlie85]. A few large systems of this type exist today [Bachant84] and more are being planned, some of extremely large size [Lenat86]. In the process of building these systems and endowing them with great deductive power, the issue of long-term maintenance is underemphasized. This issue is recognized by the people actually using large knowledge bases [Barker89].

The lack of emphasis on maintenance in early systems is easy to understand. At first, knowledge seems to be a static resource to be acquired, represented, and utilized. However, the world changes, and both the underlying data and the knowledge we derive from this data change, albeit at different rates. Large and long-lived systems need a clear approach on how changes to data and knowledge are to be managed.

In database design, update has always been a concern and has affected the storage representation and, hence, the methods of retrieval that are feasible. Methods for representation of knowledge which seem best for retrieval may become inadequate when updates to knowledge become a concern. In turn, a representation suitable for maintenance will require adaptation of the methods used to exploit the stored knowledge.

This paper focuses on the specifics of knowledge management. We will need to deal with knowledge update and retrieval. We have argued earlier for a distinction between data (that portion of information which can be mechanically maintained) and knowledge (the portion requiring expertise for its maintenance) [Wiederhold86a]. Expertise is required for knowledge maintenance because changes can have wide implications. The distinction between knowledge and data is less sharp in utilization, since here integration is essential.

In addition to distinguishing knowledge and data, our approach further partitions knowledge along two dimensions: horizontally and vertically. Before describing the partitioning however, we present some background to justify our criteria for *horizontal* partitioning.

1.2 Overview of the paper

This paper deals with a specialization of the mediator concept elucidated in [Wiederhold89]. The partitions we will define are the SoDs* introduced in that paper.

Our partitioning involves data and two categories of knowledge-based processing. Access to data was surveyed in [Wiederhold89]. In the next section we elaborate a conceptual distinction within knowledge-based systems as pragmatic versus formal approaches. This distinction defines a boundary we use for an engineered partitioning of large knowledge

---

* The term SoD is a new term to correspond with the new concept described here. We found that all other words we could think of already had excessive semantic baggage.

bases. We will assign pragmatic processing predominantly to the application layer and formal processing predominantly to the SoD layer.

Subsequently we discuss how knowledge may be partitioned into manageable units, and in Section 4 we present the approaches available for their synthesis. We follow a traditional engineering principle here: analysis of a problem into solvable subcomponents, followed by a synthesis phase into a product. Section 5 of the paper presents a simple demonstration. A mapping of the conceptual architecture into modern, distributed hardware follows. Finally we list some hard topics yet to be addressed. In the conclusion, we discuss some generalizations now foreseen, but in our work best delayed until we have gained experience with the concepts presented here.

2. Two Paradigms of Artificial Intelligence

The problems we are addressing are not novel, but are related to what we view as the source of some controversy in artificial intelligence research. We find two equally valid paradigms in artificial intelligence: the *pragmatic* paradigm, and the *formal* paradigm. We use these two terms simply as convenient labels, and include in the formal paradigm the logic-based approaches, which seek a formal, typically mathematical, grounding, and in the pragmatic paradigm those that focus on the cognitive aspects of human knowledge.

The knowledge-base partitioning we propose recognizes their differences and is intended to support and profit from both of them. We will briefly discuss some salient features of each.

2.1 The Pragmatic Paradigm

Much knowledge exists in the minds of experts. It is obtained from education and experience, and forms the most powerful tool we have for solving problems [Lenat87]. One of the great powers of such knowledge is that an expert, when confronted with a new set of facts, can use extrapolations and analogies to predict and evaluate the future effects of actions. The internal models in the experts' minds are undoubtedly quite deep and extremely difficult to extract. However, the rules by which these experts operate can be extracted, at least in part.

The acquisition of knowledge from experts has led to a large and successful activity starting from MYCIN [Shortliffe76] and documented in [Feigenbaum88]. In general, only surface knowledge needs to be obtained to have effective systems focusing on advice-giving on one specific topic. Modest numbers of rules, often fewer than one hundred, have provided effective encodings of some experts' domain knowledge. For many domains, however, more rules are needed.

More depth in the knowledge base is needed when expert systems are to encompass knowledge covering more than one topic, i.e., knowledge from more than one expert. Due to their interaction, the number of rules for problems covering multiple topics increases faster than their sum.

Even more serious is the issue of mutual consistency, when disparate topics are joined. We cannot expect the surface extraction of the internal models of two experts, covering dissimilar but overlapping topics, to match. More depth, i.e., the explicit representation of internal causal events and the logic which leads to their external expression, is likely to be needed. Mismatches of terms used to describe internal phenomena makes the results hard to validate. The issue of mismatch in databases has been addressed by a recent thesis [DeMichiel89]; in expert systems the problem is harder.

User interfaces and explanation facilities further greatly increase the size of systems. When the set of rules becomes large, problems of performance, validation, and knowledge maintenance become critical.

2.1.1 The Formal Paradigm

The alternative paradigm is the formal paradigm, which has received a major impetus since logic programming languages have appeared on the scene and made experimentation in this direction effective [Gallaire84]. Here we often see a direct exploitation of underlying data resources, and a wide variety of schemes to make data access effective [Ceri89].

The formal paradigm derives all its answers from well founded base rules and their composition. Heuristics are mainly used to improve the performance of the systems, typically by focusing search. Most accepted heuristics can be shown to have no affect on the result values [alpha beta]; others have a small risk of missing some potentially useful results [maximal objects].

The formality of the approach provides much confidence in the results, but also leads to some obvious weaknesses. We perceive as the fundamental weakness that any provable scheme is restricted to deal with the past up to the present. Any extrapolation of results into the future can never be *proven*, since unpredictable events can always occur. Unfortunately, the beneficial use of information by decision-makers is always due to a prediction of the future.

2.2 Combining the Two Paradigms

We need both the power of the formal approach, to make large systems predictable and manageable, and the power of expert abstraction and extrapolation. The interaction of the rules in an expert system is such that the user cannot predict the result—and that is of the essence of the service which is provided. In multi-expert systems, the roles of experts and users are intertwined. As these systems grow, a point is reached where an expert can no longer predict the outcome. Formal structures will help with the managing the knowledge, but the complexity of interacting bodies of knowledge is such that truly large systems need a partitioning.

The right combination will let us build future systems which are both reliable and nontrivial. Combining concepts from these two paradigms is not novel; we see it everywhere in today's practice, wherever systems are effectively used. However, today's tools do not promote any partitioning of the two types of knowledge, it is even hard to separate deductive rules and ground facts.

When we analyze practical systems today, we find a mixture of both paradigms, but often a dominance of one over the other, according to the application and the taste of the designer. In a recent paper we survey a number of projects, tools, and approaches that provide a knowledge-based layer for dealing with data [Wiederhold89]. We used the term *mediator* to capture the general concept of a knowledge layer between the user and the data. Mediators may be programs, written by an expert, in which heuristic knowledge is fully integrated with the formal techniques.

2.3 Heuristics

In our discussion we often focused on the issue of heuristics. We found that use of heuristics does not in itself provide a discrimination of the pragmatic and formal paradigms. Heuristics are nearly always used to deal with computational complexity. Most knowledge processing paradigms would not be feasible in practice without their use, and optimization strategies used heuristics based on parameters such as expected domain sizes, user needs, etc., to

develop practical solutions. The results obtained by such strategies are typically correct but not necessarily optimal.

In pragmatic systems we see a further exploitation of heuristics. Here application knowledge may provide heuristics about adequate approximate solutions. These may have errors in terms of set membership or rankings, but without taking such risks them no answers would be obtained. The pragmatic systems, in that sense, model with an unfortunate accuracy the situations faced by decision makers in practice.

2.4 Large Systems

We have stated earlier that we primarily concerned with *large systems*. Unfortunately, there are no simple criteria for the size of knowledge-based systems. A simple count of rules is a deceptive measurement. Some apparently large systems may use a substantial number of rules to store ground facts or static data. The expert knowledge may still consist of only a few hundred deductive rules.

Some other expert systems that do embody much knowledge use fairly simple knowledge representations; for instance, AI/RHEUM [Kingsland83] uses an interaction matrix of symptoms and diagnoses. Expanding such a simple representation, however, (for instance, to include issues such as time dependencies [Bolour82]), has been difficult.

3. Partitioning

There are two dimensions to the partitioning of the information systems we foresee. Horizontal partitioning divides the architecture into three main layers, as summarized in the following table:

Layer	Type of information	Deductions supported	Implemented With
H3	Broad application knowledge	Pragmatic reasoning	Expert Applications
H2	Formal domain knowledge	Logical inference	SoDs
H1	Factual knowledge or data	Relational algebra	Relational Database

These layers have been sketched already in [Wiederhold89]. The need for distinguishing updates to factual data and knowledge (for instance constraint rules) is reiterated in [Kat-Men89].

There is another dimension of partitioning in our model. Layers H1, H2, and H3, cannot be monolithic entities, and each will be vertically partitioned. The bottom layer, H1, may contain multiple autonomous databases and H2 will contain many SoDs. These SoDs may have some limited interaction as peers, but more importantly, they will share the underlying databases by means of views. At the top level (H3), multiple applications will exist, sharing and combining knowledge from the SoDs of layer H2.

This paper focuses on the central issue of knowledge partitioning in the relatively formal layer H2, but must, of course, also deal with the interfaces to the supporting data layer H1 and the supported application layer H3.
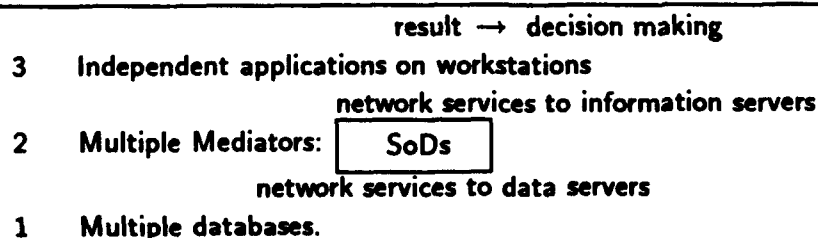
result → decision making

3 Independent applications on workstations

network services to information servers

2 Multiple Mediators: | SoDs |

network services to data servers

1 Multiple databases.

56

Figure 1: Interfaces for three horizontal layers of this architecture.

3.1 Desiderata for the Partitioning of Knowledge into SoDs

Recall that our objective is to make knowledge manageable. Two prerequisites have to be fulfilled to achieve this goal:

1 The knowledge can be formally Structured.

2 The knowledge is limited to manageable Domain of discourse.

Since eventually, we also wish to support automatable combinations of the results of the SoDs, we also must be concerned that the SoDs use mergable representations for their knowledge. It is hence not adequate to have arbitrarily dissimilar SoDs, whose results are presented on, say, distinct windows of a terminal. This approach forces the end-user to perform all integration visually, or manually by cut-and-paste methods. While having multiple terminal windows provides an advance over a desk piled high with multiple pieces of paper and terminals, it does not cover our vision of the future.

The principle for the vertical partitioning into SoDs is based on *Domains of Knowledge*. These are seen to correspond simultaneously in multiple dimensions.

1 They are limited in scope so that *one expert* can cover them and recognize inconsistencies.

2 They each have *one consistent structure* imposed on them so that changes in knowledge (or the underlying facts which led to a piece of knowledge) can be accommodated with secondary changes of limited and predictable scope.

3 They deal with *one constrained set of base data* so that updates to the underlying data and data structure required by new knowledge can be handled effectively and unambiguously.

4 They produce *one range of results*, understandable in terms of scope and depth by the end-user applications.

An essential hypothesis for our research is that the partitions for these four dimensions can be made congruent. Complementary to this partitioning hypothesis will be a combination hypothesis, to follow in Section 4.

From these criteria we can derive some more specific observations on the natural structures we expect to find in the domains of a SoD. We plan to exploit these structures whenever possible.

3.2 The Structure and Related Semantics of SoDs

The structure of a SoD expresses the structure of its semantics. We hope to have many structurally similar SoDs, since we expect that that common structures will appear in many different domains, although their labels and cardinalities may differ greatly. Our goal is as well to decompose knowledge-bases so that the structure of many SoD will be simple. We prefer hierarchical structures, but realize that some SoDs will need to use sets, DAGs, or more complex representations.

3.2.1 Hierarchical Structures

In any specific domain there is strong tendency to impose a hierarchy on the knowledge structures, which often corresponds with organizational requirements of the organizations dealing with the information. Hierarchies are instantiations of the divide-and-conquer paradigm we are trying to exploit also within the SoDs. When manipulating data through a hierarchy, we

have a predefined generalization-specialization structure. Processing in such a structure is much easier to manage than in arbitrarily connected networks of knowledge – many important problems which are intractable for general graphs have $\mathcal{O}(n \log n)$ solutions for hierarchies.

The hierarchical structure is beneficial in operations of grouping and aggregating base data into higher level abstractions, searching for specific information defined by predicates which describe such abstractions, and disambiguating updates.

While predicates can specify abstraction levels directly (department in a personnel hierarchy) quantitative goals may be satisfied by finding the right level of the hierarchy. If we need more programmers than we can find in our department, a move up to the division level may satisfy that request [Chaudhuri89].

3.2.2 Closed Worlds

We expect our SoDs to be self-describing and inspectable, and an important part of a SoD is a description of what kinds of assumptions we can make about the domain the SoD represents. Some of the SoDs will be able to support the closed world assumption (CWA) [Reiter78]. This assumption is commonly made when dealing with databases, but is risky for general expert systems. If the maintaining expert's confidence and the intrinsic definition of a domain is such that the CWA holds, then operations requiring universal quantification and negation can be supported in SoD, otherwise they should not be supported.

In a SoD dealing with corporate personnel and maintained by an expert attached to the personnel department, the CWA is likely to be valid. A SoD dealing with database consultants may be able to locate many instances of consultants, but is unlikely to be able to locate all of them until an ACADEMY OF DATABASE CONSULTING is established, and all non-members are disbarred.

Since SoDs are not restricted to relational data we will eventually need to support more flexible formalisms such as circumscription [McCarthy80].

3.2.3 Closure

We will often look for a SoD to provide all instances satisfying some precisely stated criteria of relatedness. For example we may want to find all the descendants of a given person, or find all the papers which are "similar" to a given example paper. Such queries look for some sort closure in the domain, and for SoDs with a hierarchical structure, these queries are often expressed most naturally in terms of transitive closure. At other times, like in the "similar" papers example, we will want to use distance-based concepts which are not transitive. While for simple structures, such closure-based queries can be dealt with by simple extensions to database query languages, we will need to provide some fairly complex computations to answer such queries over more general SoDs. This issue intertwines closely with the ideas of closed-world SoDs expressed above. Whether or not the closed world assumption holds in a SoD may affect the implementation of a closure-based query, but more importantly, drastically affects the interpretation and confidence to be attached to the results of the query.

3.3 Evaluation Functions

For decision-making processes we often need only the n *best* alternatives according to some ranking. The rank is obtained by an evaluation-function; such functions may be simple (say, the highest paid programmers) or complex (say, the best programmers). The SoDs for these two queries may be distinct, although the database views needed for the evaluation may be overlapping. The highest paid programmers are obtained from the personnel SoD; the challenge here is to find an efficient algorithm that can avoid unnecessary database accesses.

The SoD to find the best programmer will be complex and will depend both on some experts' insights and on complex database access functions in order to collect all the correlative data. In fact, there may be more than one SoD available to answer the best-programmer query, say the Brooks-best-programmer SoD and the Orr-best-programmer SoD.

3.3.1 Inspectability

We now arrive at a new criterion for SoDs. We wish them to be inspectable. Whereas simple formal systems may hide lower level information in order to maintain application independence, we cannot see doing this for SoDs because the application user should have the capability of determining whether the Brooks-SoD or the Orr-SoD is best for the current objective. Such an inspection may be mediated by an inspector SoD, and may not support copying of the SoD or direct access to the base data.

3.3.2 Declarative Approaches

To support inspectability it is desirable that, as much as possible, the processes within a SoD be driven by declarations and formal parameters. We would hope to capture the differences of Brooks's evaluation and Orr's evaluation by parameter settings and that the same processing routine can be employed by both SoDs.

3.4 From Data to SoDs

Because we propose a partitioned architecture for future information systems, an important issue is the interface between the supporting data layer **H1** and the SoDs of layer **H2**. Although the SoDs are most naturally implemented with object structures (as discussed in Section 5), we use relational databases as the storage scheme for factual information of layer **H1**.

Storing information in the form of complex objects can seriously inhibit sharing—different groups of users will need to assign different object boundaries to the same information [Wiederhold86b]. However, object-oriented presentations of information can be clearer and more concise than long tables of voluminous text. A desirable compromise is to provide an object-oriented interface to relational data, combining many of the better features of each representation [Barsalou88]. Such an interface serves as an effective mapping from databases to SoDs, translating **H1**'s relational tuples into **H2**'s object instances. An active area of our research has been directed toward this goal.

We introduce an object-based interface on top of a relational database system. This architecture does not call for storing objects explicitly in the database, but rather for generating and manipulating temporary object instances by binding data from base relations to predefined object templates. The three components of the object interface are:

1 The *object generator* maps relations into *object templates*; each of which can be a complex combination of join and projection operations on the base relations. In addition, an *object network* groups together related templates, thereby identifying different object views of the same database. The set of object networks constructed over a given database form an *object schema*, which, like the data schema for a relational database, represents the domain-specific information needed to gain access to the objects. The whole process is knowledge-driven, using the semantics of the database structure.

2 The *object instantiator* provides nonprocedural access to the actual object instances. A declarative query specifies the template of interest. Combining the database-access function (stored in the template), and the specific selection criteria, the system automatically generates the relational query and transmits it to

59

the DBMS, which in turn transmits back the set of matching relational tuples. In addition to performing the database-access function, the object template specifies the structure and linkage of the data elements within the object. This information is necessary for the tuples to be correctly assembled into the desired instances.

3 The *object decomposer* implements the inverse function; that is, it maps the object instances back to the base relations. This component is invoked when changes to some object instances need to be made persistent at the database level. An object instance is generated by collapsing (potentially) many tuples from several relations. By the same token, one update operation on an object may result in a number of update operations that need to be performed on the base relations. We plan to apply here results of research in the KBMS project, which deals with updating through relational views [Keller86].

An object template therefore represents a view of the database. Instantiation selects, retrieves and aggregates relevant data into object instances that can now be manipulated by a SoD. In addition, the SoDs can share factual information by sharing the object templates and their access functions. The same object, say a person, can be instantiated by more than one SoD, let's say in one SoD as a faculty member and in another SoD as a database consultant.

The formal design for this approach is domain-independent. It is then our belief that ideas, principles and programs developed in this process will be applicable to other knowledge-based interface approaches.

4. Composition

A single SoD has a power which is comparable to that of a simple expert system with access to a database or, in the database paradigm, of an advanced database query processor. Such systems are typically limited to one domain and implemented using one type of structure. Simple hierarchical systems can be effective for some tasks, as classification, ranking of alternatives, etc., but are rarely adequate for multi-objective assessments and decision-making support [Miller 70]. We do not wish to make our SoDs more complex, lest we lose maintainability.

Instead we wish to make them composable. Successful composition is the second hypothesis in this research.

One way in which SoDs can cooperate is as peers, working like a team of expert advisors to a top executive, to solve a common problem. In order to cooperate, they will need a way to exchange information. To facilitate this, the high-level language, by which applications query and command SoDs should have good algebraic properties. We discuss the basic language features in this section and will return to present further work needed in Section 7.

There is another kind of composition that must be supported, and this composition relates to the internal structure of a SoD. A SoD is a complex entity, containing data, inference techniques, knowledge and abstractions. All these subunits should be sharable, and in fact must be shared wherever possible. This is the exactly same reason that databases are normalized or software is built of reusable modules—duplicated structure leads to inefficiency and update anomalies.

4.1 An Access Language for SoDs

We envisage SoDs to be used by high-level, heuristic applications. Flexibility of access requires that the interface be non-rigid, and the intent to be able to deal with multiple SoDs in an application requires composability, as further addressed in the next section.

We hence specify an access language, SAL, which provides access to information produced by the SoDs. This information is seen to have the form of instantiated complex objects, similar to the nested-relation tuples described by [RothKS89]. The mappings from data resources to these objects is hidden within SoDs. We do need, however, some additional functionality.

This language is not yet fully specified, but it must support primitives to specify

1 Selection of subsets of objects satisfying user defined criteria.
2 Transitive closure
3 Constraints on the cardinality r of answer sets.
4 A *best* predicate to select from a ranking.
5 Computation over temporal data.

We will not discuss this last feature in this paper, although it is obvious that to project results into the future, some temporal processing is needed, as shown in some of our earlier research [Blum82, deZegher88].

It is important to note that distinct SoDs may support the extended primitives (2,3,4 above) in different ways, dependent on the structure of their domains. The *best* predicate is especially likely to be interpreted in a domain-sensitive manner. Without a defined *best* predicate a SoD can just return the r first object instances when a cardinality constraint is imposed.

Note that this language is intended to provide a smooth and sensible transition between the traditional database and PROLOG styles of data retrieval. The database style, exemplified by DATALOG, retrieves all instances, i.e., implies a cardinality constraint $r = \infty$ [Maier]. The PROLOG style retrieves initially the first instance found, implying $r = 1$. Having the cardinality specified explicitly also addresses a vexing problem in the database-to-programming-language interface. Most programming languages deal only with fixed length structures, or at best with variable length structures up to a certain maximum size. (As our current demonstration is implemented in LISP, this issue does not now arise.)

Today, without the knowledge encoded in SoDs, the methods for retrieving the *best* information are explicitly specified by the user. It is likely to require distinct methods for multiple domains. Both in database and PROLOG access styles, these specifications require knowledge of each the underlying domains and their structure. In todays' database languages a sensible specification is likely impossible to state, so that all the data has to be retrieved into memory, and then processed and reduced by application programs.

The application at layer **H3** takes the information provided by the SoDs, composes it, and reduces it as desired by intersecting results of distinct SoDs with each other. It also presents the information in the most appropriate forms to the user. To service the **H3** layer we are looking for a language similar in style to a relational algebra, rather than to a language such as SQL which attempts to provide a user-friendly interface as well as programmed access, and fails at both.

Having a language interface simplifies the tasks of the SoDs at layer **H2** since direct external presentation issues are ignored. Enough corresponding meta-data must be made available to layer **H3** so that smart formatting and pleasant presentation is feasible [Mackinlay 85]. We have not addressed this issue yet. We are experimenting with a smart menu system, using such knowledge.

4.2 A Sod Result Language

In order to make SoDs composable, one SoD must be able to act on the results of another.

We therefore define a SoD result language SOREL by which this kind of communication can take place. SOREL will be extremely simple and limited, especially in comparison to the SoD access language. One way of looking at this is to realize that SAL is in some sense a union of capabilities, since it must be powerful enough to express anything we would ask of a SoD, while the SoD result language is more like an intersection, since it should be understood by all SoDs.

The exact form of SOREL will depend on the SoDs and the needs of the application, but for most applications, we expect the SoDs to return only ground data, i.e., tuples, relations, and object identifiers.

The answers given by a SoD in SOREL will be returned to the application at **H3**. The application may then use these results directly or as input to another SoD.

4.2.1 Identifying Shared Objects

One of the first problems that must be handled for SoDs to work together cooperatively is to get them to agree on a common ground for communication. Human experts often disagree as to the meanings of words or of concepts, and this will be a problem for SoDs as well. In one common case, the architecture and its support for definitional composition, can help greatly in identifying shared objects. Because our SoDs can be built from simpler, shared components, it will often happen that two SoDs will be using an object created at a lower level. In this case, it is easy for the SoDs to recognize that they are sharing the same object — they are both looking at the same object identifier.

In the more general case, identification is not this easy. If the SoDs are using objects created on different computer systems, or if the objects are created at a high level, we can easily have two computer "objects" (with distinct identifiers) that nevertheless denote the same abstract object in the real world. When this happens, we will have to compare the objects, relying on key values and matching heuristics. Perhaps we can invoke a SoD to help us with the merging tasks. If the domains of the attributes to be merged are actually mismatched, then we certainly need intelligent processing, and we may need rankings based on the *best* match [DeMichiel 89].

4.3 Power of the Combined System

By partitioning the knowledge base, we gain the ability to use and combine special purpose SoDs and their knowledge representations without having to build one super-interpreter which understands all knowledge representations (and all the combinations of the knowledge representations.) This partitioning then makes maintenance much more tractable. However, in partitioning the data into SoDs, and allowing them to communicate only via the restricted SoD result language, we lose some of the arbitrary connectiveness associated with knowledge representations such as semantic nets.

This loss of connectivity may reduce the expressive power of the system. For example, let's say that when designing a wing, an aircraft designer looks at two SoDs, one of which can evaluate and optimize a design for aerodynamic performance, and another SoD which looks at mechanical strength and weight. Since the wing should have good performance according to the criteria of both SoDs, the designer is faced with an iterative (or even trial and error) process, of checking designs though both SoDs and looking for a global optimum.

This iterative process might have been avoidable, if it the two SoDs were unified into one super-SoD able to to find a global optimum for aerodynamics, strength, and weight. What this example tells us is that the design process which splits knowledge into SoDs is quite critical. A given partitioning may gain us a great deal in terms of implemer' tion, maintenance,

and assignment of responsibility, but may also incur a significant cost in expressive power.

5. A Demonstration

To demonstrate the concepts, the students on the KSYS project have chosen the task of assigning reviewers for journal papers submission. Four SoDs serve the task:

1 Relevance: we need reviewers with a background relevant to the submitted paper. This task is performed by matching in a keyword classification hierarchy.

2 Quality: we prefer the most qualified reviewers. For this task we rank potential reviewers based on their published output in books, journals, etc.

3 Conflict avoidance: we cannot assign reviewers to friends or colleagues. Here we match people based on institutional affiliation in overlapping intervals.

4 Responsiveness: The reviewers must produce their reviews in time. Here we can look at a log of electronic-mail interactions.

We can use this example to elucidate the difference of the AI paradigms allocated to level **H3** and **H2**. The tasks in the SoDs at layer **H2** can all be defined quite formally. At the top layer **H3** some unwarranted pragmatic heuristics are used to implement the reviewer selection task. For instance:

1 Having written high quality publications in a topic area does not assure one that the candidate does equally well as reviewer. It is the best guess that our application task can make, but we all know some excellent critics who do not write much. The mapping of `qualified_writer` → `qualified_reviewer` is pragmatic. The establishment of a set of `qualified_writers` is adequately formal to justify its allocation to a SoD.

2 Having worked together does not make one a friend, and being a friend does not imply favoritism. But we do need to weed out risky matches — in fact, due to prior publications the most likely *best* match is the submittor of the paper.

3 Electronic mail responsiveness is probably only weakly correlated with fast reviewing — there are people who respond instantly to email and never respond to review requests.

The language currently used between layers **H1** and **H2** is LISP because it supports the extensibility essential to rapid research progress.

The data accessed by the first three SoDs are distinct views of an extensive bibliography of knowledge and database references, collected over about 18 years, with about 6,000 entries. Information kept includes type of publication (for the Quality-SoD), authors (the principal identifiers), author's location (for the Conflict-avoidance-SoD), publication details and sequence with dates (for the Conflict-avoidance-SoD), title, abstract, and classification (the last three are used by the Relevance-SoD).

Two of the SoDs are currently implemented – relevance and conflict avoidance. They are implemented as Lisp programs which have access to the object system and a commercial relational database. As we gain more experience, we intend to replace the Lisp code with a more declarative representation.

At the simplest level, the relevance SoD takes a keyword (or list of keywords) describing the subject of the paper to be reviewed, and looks in the database for authors who have written papers on the keyword(s). If the enough authors are returned by this database query, this is all that happens. If however, the database query does not find enough authors,

or if the application asks for more candidate reviewers later, the relevance SoD will replace the original query by a more general one, in order to increase the cardinality of the result.

This capability is an example of query generalization [Chaudhuri89]. It is possible because the SoD makes use of some of the semantics of the keywords. The keywords are arranged in a hierarchy, in which the parent is the more general keyword, and the children the more specific. If a query is does not return sufficiently many results, a concept of semantic distince in the hierarchy is used to suggest alternate keywords to try.

The data structures used by the SoD are designed to efficiently support this kind of iterated query style efficiently. A set of authors can be found by a succession of related queries can be answered with about the same total effort as would be needed to find that same set of authors with one more general query.

The application interface is simply a set of Lisp functions which the application can use. As our system evolves, we intend to built a higher-level interface. In our design, an application which is looking for reviewers would submit a query of the form:

```
select best 3 reviewer
from relevance-SoD
where relevant = 'knowledge-base'
  and reviewer not in
      (select all friend
       from conflict-avoidance-SoD
       where author = 'Gio Wiederhold')
```

Note that this query refers to two different SoDs. Since a particular SoD can only answer queries about its own domain, this query is translated into a slightly lower level form, which specifies the individual queries to the SoDs, and the information flow between them.

```
a := select all friend
     from conflict-avoidance-SoD
     where author = 'Gio Wiederhold'
b := select best 3 reviewer
     from relevance-SoD
     where relevant = 'knowledge-base'
     and not in a
RETURN b
```

Note that even this second query is fairly high level. It refers to such abstractions as relevant, which are implemented by the SoDs.

6. The Implementation Architecture

The demonstration is implemented in fairly straightforward way, but a short description will illustrate some issues better than an abstract discussion can.

6.1 SoD Implementation

The criteria we have listed encourage an implementation which supports object-oriented type definitions. We are building a simple LISP-frame structure to support SoDs. Low level frames correspond to database schema entries and support retrieval from databases; data that is retrieved must be bound into the object-type structures and represent object instances as discussed in the previous section. Concepts such as trackers [Ceri89] deal with effective handling of partially or fully instantiated sets of data.

64

Functions and predicates from the object type definitions are inherited by the object instances. Default values are overridden by any actual data retrieved from the database.

Common methods, as selection and transitive closure, will be shared by multiple SoDs, especially when their general structure is similar. Sharing should be possible even if their object types and instances differ. General parameters, as the CWA, can cause alternate variants of methods to be invoked. Note again that SoDs are not distinguished by their program structure or algorithms, but rather by their structure and domain knowledge.

6.2 Elements of a SoD

The specific structure of a SoD is shown in Fig. 2.

<div align="center">

Interface: high-level language

Primitives
Objects and Rules
Binding

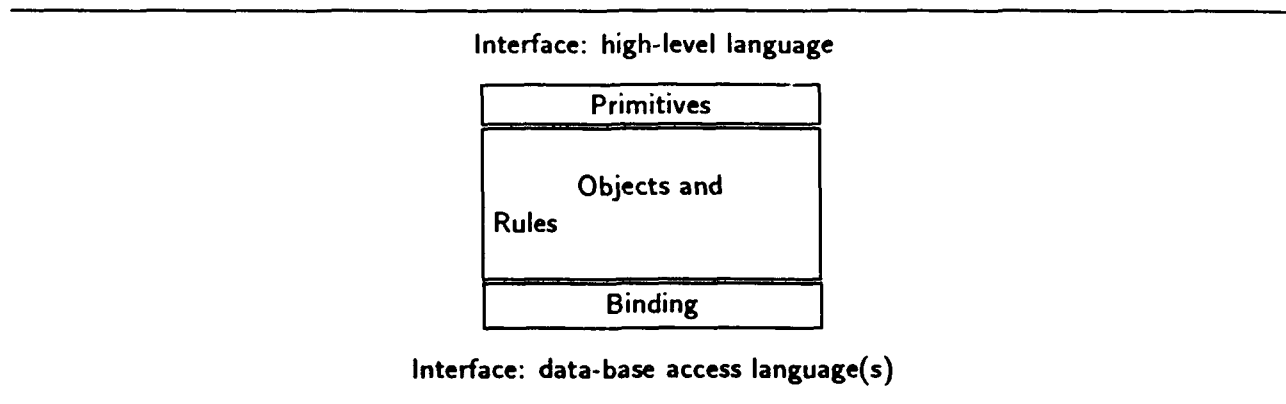**Interface: data-base access language(s)**

</div>

Figure 2: The components of a SoD.

Each SoD contains a simple hierarchy of objects pertaining to its domain. The views provided by the objects compose the entire view of the SoD over the database. Rules embedded in the object structures are used to control the instantiation of objects and compute dynamic slot values. Each SoD provides primitive operations it can support and accepts the parameters it needs from applications. Binding interfaces the SoD objects with the underlying databases by retrieving object instances generated from the databases. For efficiency, the instances of the SoD objects are bound into memory as early as possible.

In terms of their external interface we expect SoDs to be free-standing units, accessible on the high-speed communication networks now in the planning stage. For efficient execution, SoDs can be replicated on other computing nodes where the data (**H1**) or the applications (**H3**) reside.

6.3 Structure of an Object

We indicated earlier that we are implementing the SoDs using frames, similar as seen in the UNITS system [Stefik79] and its successors such as KEE and RX [Blum82]. This means that an object is implemented as a frame in a LISP structure.

A frame is composed of a number of slots. Each slot is labeled, and contains the following elements:

1 An indication of its SoD membership

2 A domain definition, to constrain its values

3 A value

Values may be constants or references to other objects. Constants occur mainly in frames that have been instantiated from the database.

An object frame inherits its slots from the SoD that it is a member of. A frame in our system that belongs to a single SoD differs but little from frames seen in other systems. The differences arise when on object becomes a member of multiple SoDs.

6.4 Structural Support for Composition

An object may be a member of multiple SoDs. It is the task of the binding layer to recognize that information for an object already exists and perform the binding for the two overlapping instances.

The joint object will inherit the slots from all the SoDs it belongs to. We see here a departure from the common schemes used when multiple inheritance is needed: the information is not intermingled according to local rules. Since we use mainly information from the database, it is not likely that there will be rules to cover the variety of interactions that can be realized among objects from distinct SoDs. The disjoint inheritance is also imposed on the values in the object slots:

- An inherited slot value is inherited from only one specified SoD.

We hence provide for multiple inheritance into objects, but not into the same slots of objects. This rule eliminates the multiple inheritance problem for which no general solution is likely to be found for multiple inheritance. We find solutions that have been proposed too specific for a general system, but recognize that multiple inheritance is a valid and useful concept.

An example will clarify our approach. Say that the Personnel-SoD has retrieved an individual (John) with location, job_classification, salary, etc., information from a PEOPLE database. John is also being retrieved by a Skills-SoD as possessing the skills and a willingness to do weekly consulting on some topic. The slots identifying John are identical and shared, not requiring inheritance. The salary slot belongs to the Personnel-SoD, and may either be explicitly retrieved or filled in by inheritance for all employees of that classification. The fee slot belongs to the Skills-SoD, and may be estimated by averaging known fees of similar individuals. There is less likely to be a well established hierarchy here.

For some decision-making process at layer H3 we may actually need an income estimate. The application can obtain the distinct components and combine them as it pleases.

If the task of estimating incomes is frequent and consistency is desired, then it should be formalized. This means we assign a new expert to the task and let her define a SoD for income estimation. An income slot, inherited from that SoD may be adjoined to the object for John and income is then computable on the basis of salary, fee, alimony, and any other financial reward slots that other SoDs may instantiate in this object. The values in this slot will not be subject to inheritance, only the formula is inherited.

Label	SoD	domain	value
ID	—	identifier	internal
name	—	identifier	John
job_class	Personnel	code	G21
salary	Personnel	dollars	35000
deductions	Payroll	count	3
skill	Consult	code	2324, 2386, 3756
fee	Consult	dollars	1000
willingness	Consult	+scale	4
income	Estimator	formula	salary + alimony*12 + fee*52

Figure 3:   Frame with SoD labeled Slots.

It is clear why inspectability of SoDs is needed. The questions of composability are so complex that it is often desirable to determine how a value as income is computed. Still, we wish to delegate the actual computation to a SoD, in which we place normally some trust. The confidence in the SoD emulates confidence we have in the reports and summaries provided by specialists from our Personnel department, the Skills specialists, and in our assistants who compose the information. Only if we need to question the result do we inquire into their methods.

7.   Subproblems to be Addressed

The task of managing large knowledge-bases, which undergo growth and change is daunting. While we have sketched those aspects of our approach that seem clear to us, there are many tasks which require expansion and generalization.

We will list some here. For some of these we have some ideas on how to address them, other problems are quite open.

7.1   Object Identification

Correct object identification is critical for the matching operations at layers **H3**. While objects instantiated with SoDs at layer **H2** have a simple linkage with the underlying database, we can use database keys or derived surrogates from layer **H1** to identify objects.

When derived objects are created within SoDs such identifiers may become difficult to link. The fact that SoDs will share computational processes can help, but probably not guarantee correct matching when information follows different processing paths.

7.2   Dynamic Slot Generation

Dynamic slot values are derived using knowledge about the data in the database. This may take the form of a default values when the base data are unpopulated, procedural functions over the base data, or declarative rule sets.

The issues in this area involve deciding at what point to compute the derived value and determining how to re-compute this value when the base data changes. It may even be that some derived values are stored in the database for efficiency. In this case we may need trigger mechanisms to update the values when the base data changes.

At a higher level of abstraction we must consider how objects acquire new slots. In our example a slot was acquired by merging selected objects with the **Estimator** SoD. How such a procedure can be generalized has not yet been defined. A follow-on phase could have the

application at Layer **H3** define a private SoD, or its equivalent, so that private computations can be attached to materialized objects in layer **H2**. We do not foresee dynamic generation of data accessing slots.

7.3 Language Optimization

Choosing an algebraic language, SAL, for communicating with the SoDs should enable optimization. Currently we process the SoDs in the order mentioned in the task definition, but other sequences are likely to provide better performance. While we understand issues of join ordering [Swami88], we have new operations now that will require new optimization rules.

This SAL language operates on larger granules of primitives than current 4GL languages. Semantically similar primitives of the language will be executed differently in the various SoDs. To perform global optimization the SoDs have to be able to provide abstractions or evaluation functions of their methods to the global optimizer.

Note that SoDs interact at the language interface level in at least two ways:

1 The output from one SoD may help another SoD reduce its search

2 The output from one SoD may necessitate a previously-executed SoD to be re-executed.

For example, when searching for "three competent and responsive reviewers," the list of competent reviewers could help reduce the search for responsive reviewers, but if only one of the competent reviewers turns out to be responsive, then perhaps the "competency" test should be relaxed and re-executed in order to return the requested three reviewers. In neither of two cases will it be necessary to ship large volumes of data for resolution of the intersection result to the computer used for the application.

7.4 Object Instantiation

In the system design adopted for KSYS, a binding module interfaces between the frame system layer and database layer. It provides object instances generated from databases data into the frame system.

The instances of frames used by SoDs are generated from relational databases. Each frame prototype for a SoD defines a view of the database for selecting a subset of the database as frame instances. When frame instances are needed, its view is translated into a relational query and delivered to the database. The query results are stored in main memory and processed. We expect that for many complex queries delivered to the database we cannot achieve reasonable performance by simply delivering the queries to the database.

We are thus developing a *binding* strategy for minimizing accesses to secondary storage databases. The binding strategy is to cache the multiple query results in a nested, prejoined form for compact storage for retrieval of frame instances. Queries delivered to the database are modified as needed whenever the binding module detects that relevant reusable query results have already been bound into main memory.

7.5 Interacting SoDs

At present the top application layer is the executive responsible for the integration of knowledge obtained from SoDs. An extension of this architecture we must investigate is the hierarchical composition of a SoD from sub-SoDs. In this way the parent SoD would perform the task of integrating knowledge from sub-SoDs, and itself might be a sub-SoD of another SoD. For this to be possible, the interface exported from a SoD (i.e., the query language supported) must provide a superset of the functionality used by a SoD.

This direction moves us closer to the interacting ACTORS paradigm [Hewitt;73]. We do, however, still expect to impose constraints on their composition, and in that sense are closer to concepts of the ORG approach [Malone 87].

8. Conclusion

We have presented an approach to deal with the management large knowledge-based systems. The approach is based on a domain and structure sensitive partitioning of the data and knowledge to be managed, and careful and limited interactions among the partitions. A simple demonstration illustrates our approach.

We define the criteria for SoDs, our principal unit for the partitioning, and discuss the effects of those criteria. With the benefits of partitioning a loss of power is induced; we can no longer navigate in seemingly arbitrary ways throughout the knowledge base. It is difficult to assess the cost-benefit ratio of this tradeoff. We are optimistic that it is high; analogies can be found in human organizations as well as in other large computer systems.

In our current demonstration the efficiency cannot be measured. We know that acceptance of new technology requires both conceptual benefits as well as reasonable efficiency, and we hope to gain efficiency with our binding approaches. These will benefit from the structure information that SoDs provide.

Automation of techniques of knowledge management will be essential in a wide range of future applications. We hope and expect that the principles we have laid out will contribute to an orderly and productive growth of the field.

References

[Bachant 84] J. Bachant and J. McDermott: "R1 revisited: Four years in the trenches"; *The AI Magazine*, 5(3):21–32: 1984.

[Barker89] Virginia E. Barker and Dennis E. O'Connor: Expert Systems for Configuration at Digital: XCON and beyond"; *Comm ACM*, Vol.32 No.3, March 1989, pp. 298–318.

[Barsalou 88] T. Barsalou: "An object-based architecture for biomedical expert database systems"; Proceedings of the Twelfth Symposium on Computer Applications in Medical Care, IEEE Computer Society, pp. 572–578, 1988.

[Blum 82] R.L. Blum: "Discovery and representation of causal relationships from a large time-oriented clinical database: The RX project"; Lecture Notes in Medical Informatics, Springer-Verlag, New York, 1982.

[Bolour 82] A. Bolour, T. Anderson, L. Dekeyser, and H. Wong: "The role of time in information processing: A survey"; *ACM SIGART Newsletter*, 80:28–48, 1982.

[Chaudhuri 89] S. Chaudhuri: "Generalization as a Query Modification Operation"; Submitted to VLDB 89.

[Ceri 89] S. Ceri, G. Gottlob, and G. Wiederhold: "Interfacing Relational Databases and PROLOG Efficiently"; *IEEE Transactions on Software Engineering*, pp.153–164, Feb.1989.

[DeMichiel 89] L. DeMichiel: "Performing Operations over Mismatched Domains"; *IEEE Data Engineering* 5, Los Angeles, Feb.1989.

[deZegher 88] I. deZegher-Geets, A. Freeman, M. Walker, R. Blum, and G. Wiederhold: "Summarization and display of on-line medical records"; *MD Computing*, 5(3):38–45, 1988.

[Feigenbaum 88] E. Feigenbaum, P. Nii, and P. McCorduck: *The rise of the expert company: How visionary companies are using artificial intelligence to achieve higher productivity and profits*; Times Books, 1988.

[Gallaire 84] H. Gallaire, J. Minker, and J-M. Nicolas: "Logic and Databases: A Deductive Approach"; *ACM Comp.Surveys*, Vol.16 No.2, Jun.1984, pp.153–185.

[Hewitt:73] Carl Hewitt, P. Bishop., R. Steiger.: "A Universal Modular ACTOR Formalism for Artificial Intelligence"; IJCAI 3, SRI, Aug.1973, pp.235-245.

[Miller 70] James R. Miller: *Professional Decision Making — A Procedure for Evaluating Complex Alternatives*; Praeger pubs., 1970.

[KatMen 89] H. Katsuno and A.O. Mendelzon: "A Unified View of Propositional Knowledge Base Updates"; Univ. of Toronto, rcvd. Jan.1989.

[Keller 86] A.M. Keller: "The Role of Semantics in Translating View Updates"; *IEEE Computer*, 19(1):63–73, 1986.

[Kerschberg 85] Larry Kerschberg (editor): *Expert Database Systems*; Benjamin-Cummins, 1985.

[Kingsland 83] L.C Kingsland, D.A.B. Lindberg, and G.C. Sharp: "AI/RHEUM: A consultant system for rheumatology"; *Journal of Medical Systems*, 7:221–227, 1983.

[Lenat 86] D. Lenat, M. Prakash, M. Shepherd: "Cyc: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks"; *The AI Magazine*, 6(4):65-85, 1986.

[Lenat 87] D. Lenat and E. Feigenbaum: "On the thresholds of knowledge"; IJCAI 87, Milan, Italy, 1987.

[Mackinlay 85] J. Mackinlay and M. Genesereth: "Expressiveness and Language Choice"; Data and Knowledge Engineering, 1(1):17–29, June 1985.

[Malone 87] T.W. Malone, K.R. Grant, F.A. Turbak, S.A. Brobst, and M.D. Cohen: "Intelligent Information-Sharing Systems"; CACM, 30(5):390–402, May 1987.

[McCarthy 80] J. McCarthy: "Circumscription—a form of non-monotonic reasoning"; *Artificial Intelligence*, 13:27–39, 1980.

[Methlie 85] L.B. Methlie and R.H. Sprague: "Knowledge Representation for Decision Support Systems"; 1985.

[ORG]

[Rathmann 89] P. Rathmann and M. Winslett: "Circumscribing Equality"; IJCAI 89, Detroit, Michigan, 1989.

[Reiter 78] R. Reiter: "On Closed World Data Bases"; in H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, pp. 119–140, Plenum, New York, 1978.

[Reiter 80] R. Reiter: "A Logic for Default Reasoning"; AI, Vol.13, Apr.1980.

[RothKS88] Mark A. Roth, Henry F. Korth, and Abraham Silberschatz: "Extended Algebra and Calculus for Nested Relational Databases"; ACM TODS, Vol.13 No.2, dec.1988, pp.

389–417.

[Shortliffe 76] E.H. Shortliffe: "Computer-based medical consultations: MYCIN"; American Elsevier, New York, 1976.

[Stefik 79] M. Stefik: "An examination of a frame-structured representation system"; IJCAI 79, Tokyo, Japan, 1979.

[Swami88] A. Swami and A. Gupta: "Optimization of Large Join Queries"; Proceedings of ACM-SIGMOD International Conference on Management of Data, 1988.

[Wiederhold 86a] G. Wiederhold: "Knowledge versus data"; in M.L. Brodie and J. Mylopoulos (eds.), *On knowledge base management systems*, pp. 77–82, Springer-Verlag, New York, 1986.

[Wiederhold 86b] G. Wiederhold: "Views, objects and databases"; *IEEE Computer*, 19(12):37–44, 1986.

[Wiederhold 89] G. Wiederhold: "The architecture of future information systems"; to appear in the Proceedings of the International Symposium on Database Systems for Advanced Applications, KISS and IPSJ, Seoul, Korea, 1989.

[WWHCSCWDR 87] G. Wiederhold, ,M. Walker ,W. Hasan, S. Chaudhuri, A. Swami, S.K. Cha, X-L. Qian, M. Winslett, L. DeMichiel, and P.K. Rathmann: "KSYS: An Architecture for Integrating Databases and Knowledge Bases" in Gupta and Madnick (eds) *Technical Opinions Regarding Knowledge Based Integrated Information Systems Engineering*, MIT, 1987.

Generalization and a Framework for Query Modification

Surajit Chaudhuri
Stanford University

Abstract

The rigidity and the limited expressiveness of the relational queries often force us to iteratively modify a query. We pose an initial query and once we discover that the answer does not meet the additional constraints, which are not expressed in the relational query, we try to modify the query in a way such that those constraints are satisfied. Our aim in this paper is to capture this iterative process by extending the query model. We define *extended queries* which express additional constraints on the answer set and designate some of the conditions in the relational query as flexible. The *query modification operators* modify flexible constraints to satisfy an extended query. We describe in detail the query modification operation *Generalization*. We identify the conditions under which generalization is applicable. We propose rules of generalization and suggest an algorithm for picking a minimal generalization.

1 Introduction

Current relational database systems retrieve tuples from databases which satisfy a relational query. Efficient query processing techniques have been developed to evaluate the select-project-join class of queries. However, it is often hard or impossible to express many useful constraints on the answer relation (e.g., aggregate properties) in relational languages. Therefore, it is likely that the the answer to the query may not meet the expectations of the user. In such a case, the user modifies the query. This iterative process continues until the answer set meets the expectations of the user.

The process of query modification puts an undue burden on the user. He must have considerable background knowledge about the semantics of the database in order to perform such iterative modifications well. As we attempt to model complex applications using databases, we perceive a pressing need to provide tools and techniques to perform such modifications automatically.

In this paper, we take a first step in providing a framework that formalizes the basic ideas in query modification. The framework opens up the possibility of partially automating the above iterative process. We illustrate the framework with the example of *generalization* as a query modification operation.

The paper is organized as follows. In the next section, we informally discuss the concept of query modification and outline our approach. In section 3, we present our framework for query modification. The following section introduces generalization as a query modification operation. In section 5, the rules of generalization are presented. An algorithm to pick an acceptable generalization is outlined in section 6. Section 7 summarizes the related work in this area. We conclude with an outline of future work.

2 What is Query Modification?

Query Modification[1] is the process of refining a query when the answer to the query does not meet the expectations of the user. Let us begin by examining two simple examples where query modification is required:

Example 2.1: We need to find at least five reviewers for the paper: "Application of Object Oriented Databases to CAD". Therefore, we ask the database for the reviewers who have both CAD and object oriented databases as their areas of expertise. However, there may not be five such reviewers. In that case, we modify the query to accept reviewers who are knowledgeable in engineering and object oriented databases. The set of people who have expertise in engineering includes people who have expertise in CAD. We may need to continue weakening our constraints further in order to meet the cardinality constraint.

Example 2.2: Our task is to get together an international fleet drawing manpower from smaller fleets each of which must have a frigate, and is based in the Persian Gulf. We want to have a total manpower of 20,000 or more. The condition that each participant must have

---

[1]This definition is distinct from the sense in which the term is used in INGRES.

a frigate may not be violated. However, if necessary, we are willing to weaken the query to allow naval ships based nearby to satisfy the manpower requirement. W note that the condition on manpower requirement can not be expressed in the relational language. Also, it is not possible to indicate that the user is willing to accept answers from the weakened query.

The two examples above illustrate the following shortcomings of the relational query models:

- It is often hard to express in a relational language the constraints that the user expects the answer to the query to satisfy. In Example 2.1, we would like the database to return at least five reviewers.

- The user may be willing to allow selective modification of the relational query to meet his expectations. In Example 2.2, we relaxed the original query to include ships from the countries which are near the Persian Gulf. One can't designate and take advantage of flexible conditions in the relational framework.

- The integrity constraints and other meta-knowledge in the schema could be used for query modification. In Example 2.1, we could use the subset relationship between the set of experts in CAD and the set of experts in engineering to derive the modified query. Unfortunately, in all existing systems, the task of refining the query is left to the user completely.

In the next section, we present our framework which addresses these shortcomings. Below we review the terminologies in relational databases that we will use in the rest of the paper.

A database schema consists of relations and a set of integrity constraints Σ. Any *valid database* must be a model of its integrity constraints Σ.

A *query* is an open formula, denoted $Q(\overline{x})$, where $\overline{x}$ is the set of *free variables*. The relation represented by $Q(\overline{x})$ in a database D is denoted Q^D. We will often abbreviate $Q(\overline{x})$ by Q when either the list of variables is not important or is obvious.

Example 2.3: $Knows(x,y)$ means that person x is knowledgeable in subject y. The set of integrity constraints below state that a person knowledgeable in a subarea (e.g., object-oriented database) is also knowledgeable in the area (e.g., database).
$\Sigma = \{\forall x(Knows(x, \text{oodb}) \rightarrow Knows(x, \text{database})),$
$\forall x(Knows(x, \text{cad}) \rightarrow Knows(x, \text{engineering}))\}$

The following query asks for people knowledgeable in CAD as well as object oriented databases.
$Q(x) \equiv Knows(x, \text{cad}) \wedge Knows(x, \text{database})$

A query $Q(\overline{x})$ is a *conjunctive query* iff it is of the syntactic form[2]: $Q(\overline{x}) \equiv \exists \overline{z} \bigwedge_i P_i(\overline{x_i})$ where $\overline{x_i} \subseteq (\overline{x} \cup \overline{z})$, and P_i-s are relation symbols. The variables in $\overline{z}$ are called the *existential variables*. For the simplicity of exposition, we will restrict ourselves to only conjunctive queries. $Q(x)$, in Example 2.3 above, is a conjunctive query.

3   Framework

We now propose our framework for query modification. This framework extends the relational query model. The key aspects of our framework are as follows:

- A proposal for an *extended query* that expresses the user's expectations of the answer set in addition to the relational query.

- Query Modification Operators to *modify* an extended query.

- Strategy for applying the modification operators.

Extended Queries: We represent the additional constraints by defining an *extended query* to have two components. The first is a relational query, denoted by Q. We will restrict Q to be a conjunctive query for simplicity. The second component is a boolean function (predicate) $\mathcal{P}$ that takes as input the answer generated by executing the query Q over a database D. Thus, $\mathcal{P}$ is a function from 2^S to $\{True, False\}$, where S is the domain to which every answer tuple must belong. We will refer to the second component as the *acceptance test*. This component reflects the user's expectations of the answer set.

We say that an extended query is *acceptable* for a database D iff $\mathcal{P}(Q^D) = True$. The *answer* to an extended query is an empty set if the query is not acceptable and is Q^D otherwise. An extended query model reduces to a relational query when $\mathcal{P}$ is always true. For simplicity, we assume that Q is entirely flexible and $\mathcal{P}$ is invariant. Under the above assumptions, we can model an extended query by a doublet: $\langle Q, \mathcal{P} \rangle$, where the only invariant part is the acceptance test $\mathcal{P}$. The framework easily extends to the case where some of the conjuncts in Q are also invariant.

Modification Operators: The goal of the query modification operators is to transform a query $\langle Q, \mathcal{P} \rangle$ to a query $\langle Q', \mathcal{P} \rangle$ so that the latter is acceptable. It is therefore necessary to establish a correspondence between the acceptance tests and the query modification operators.

---

[2] $\bigwedge_i P_i$ is a shorthand notation for $P_1 \wedge P_2 \wedge \ldots \wedge P_i \wedge \ldots \wedge P_n$

We will assume that the system designer specifies a set of *primitive* acceptance tests. For each primitive acceptance test, there will be one or more query modification operators and vice-versa. Thus, there will be a many to many association between the set of primitive acceptance tests and the set of query modification operators. Such associations may be provided by the designer of the database schema or they could be inferred from properties of the query modification operators and the primitive acceptance tests. For example, we know that the sum of values of an attribute for the answer relation will increase if the number of answer tuples increases. Therefore, an operator that increases the number of tuples in the answer can be used for increasing the sum. Examples 3.1 to 3.3 informally describe a set of acceptance tests and query modification operators. We will assume that every acceptance test is a set of primitive acceptance tests. The acceptance test returns true iff all the component primitive acceptance tests return true.

Next, we address the issue of how a query modification operator is specified. A natural representation of an operator is in terms of a *set of rewrite rules*. These rewrite rules will reflect the semantics of the query modification operator and the integrity constraints in the database. We assume the following structure for the rewrite rules:

$$< expr > \Longrightarrow < expr' >$$

where $< expr >$ (and $< expr' >$) is a conjunctive query. An application of the above rule will transform a set of conjuncts in the query that unifies with the instance of $expr$ to the the corresponding instance of $expr'$.

Example 3.1 (Generalization): Consider Example 2.1. The extended query, as formulated by user, has the following components:

$$Q(x) \equiv Knows(x, \text{cad}) \wedge Knows(x, \text{oodb})$$

$$\mathcal{P}(Q^D) \equiv Count(Q^D) \geq 5$$

Let's assume that *generalization* is the operator associated with $\mathcal{P}$. One of the rewrite rules for generalization is [3] $Knows(x, \text{cad}) \Longrightarrow Knows(x, \text{engineering})$
The rule is based on the integrity constraint given above. We can generalize the above query to:

$$Q(x) \equiv Knows(x, \text{engineering}) \wedge Knows(x, \text{oodb})$$

Example 3.2 (Aggregate Tuning): Our relational query is to retrieve employees who earn more than 50k. The acceptance test is that the sample size must have mean age less than 30. If the answer set of the original

---

[3] A simplied version of constant generalization (section 5.3).

query has a higher mean age, we can use the meta-knowledge $salary \leq a * age + b$ to change 50k to a more appropriate value.

Example 3.3 (Prototype Exclusion) Assume that we want to find the accident number and the type of the aircraft for each air accident that occurred in 1988. Let $accident(accidentid, type, 1988)$ be the relational query that we ask. However, on examining a sample of the answer relation, we realize that the answer contains military aircrafts too. One such unwanted tuple is $(A0090, F16, 1988)$. We realize that our query is too general and indicate to the system that the aircraft $A0090$ ought not to be in the answer. The above now serves as the acceptance test of the extended query. The rewrite-rule for prototype exclusion excludes the most specific relation to which the unwanted answer belongs. Let that relation for the above tuple be *military_accident*. The system modifies the query by adding the negated conjunct $\neg military\_accident(id, type, date)$ to the initial query.

Strategy for Operator Application: An operator becomes eligible to apply when a primitive acceptance test with which it is associated fails. If there are multiple eligible operators, then, we need schemes for conflict resolution. A complete discussion of the strategy selection would require defining the set of primitive acceptance tests and studying the combinatorial properties of the rewrite system. Example of such properties are Church-Rosser, confluent and canonical [Hin 72]. Some of these properties help us identify equivalence classes of modifications.

The strategy for operator applications must capture the intuition that the flexible constraints in an extended query should be modified as little as possible. Formally, we want our rewrite system to guarantee *minimal* modification:

Definition (Minimality): An acceptable modification Q' of Q is *minimal* with respect to a set $\mathcal{R}$ of rewrite rules if there is no other acceptable modification Q'' of Q such that the transformation to Q'' uses only a subset of the set of applications of rewrite rules needed for modifying Q to Q'.

A complete exposition of the language of extended queries, rewrite rules and strategy selection to guarantee minimality is beyond the scope of this paper. However, we will discuss these issues in the context of *generalization*, a useful query modification operator. We will define the generalization transformation and will identify the acceptance tests for which the operator can be applied. We will show how integrity constraints may be used to derive the rewrite rules for generalization. Finally, we present an algorithm that enables us to pick a minimal generalization. Our goal is to run through

all the steps necessary to design a query modification operator.

4 Generalization

Generalization is a query modification operator that weakens the query. Therefore, the set of answers generated by the transformed query is a superset of the set of answers produced by the original query. This operator can be applied to satisfy the acceptance tests that require a minimum cardinality of the answer set (or variants of such acceptance tests). Examples 2.1 and 2.2 illustrate such acceptance tests.

One way to define the generalization operator is in terms of the relationship between the original query and the transformed ("generalized") one:

Definition (Generalization): An extended query $\langle Q', \mathcal{P} \rangle$ *generalizes* $\langle Q, \mathcal{P} \rangle$, if over every database D for the given schema,

$$Q^D \subseteq Q'^D$$

We can now say that the effect of the generalization operator is to produce a generalized query. The following facts are immediate consequences of the above definition. We assume that integrity constraints are the only additional knowledge used for generalization.

Fact 4.1: If Q' generalizes Q and Σ is the set of integrity constraints, then[4], $\Sigma \models \forall \overline{x}(Q \rightarrow Q')$ where $\overline{x}$ is the set of free variables in Q (and Q').

Fact 4.2: (Transitivity) If Q' generalizes Q and Q'' generalizes Q', then Q'' generalizes Q.

We now characterize the set of acceptance tests for which generalization may be used as a query modification operator. We observe that if the acceptance test is such that generalization of an acceptable query is also acceptable, then generalization may be used as an operator. We call this set of acceptance tests *monotonic*:

- An acceptance test $\mathcal{P}$ is *monotonic* if $\mathcal{P}(D)$ is true, then for all $D' \subseteq D$, $\mathcal{P}(D')$ is true.

The constraint on the minimum value of accumulated manpower in example 2.3 is a monotonic acceptance test. For the primitive acceptance tests, we explicitly store information about whether or not it is monotonic.

The definition of generalization does not immediately suggest an algorithm to construct the rewrite rules necessary to specify the generalization operator. Indeed, depending on Σ and Q, there could be many candidate generalizations. However, construction of all possible generalizations may be prohibitively expensive and hard to explain to the user. In the next section, we address this issue of defining the rules of generalization.

---

[4] $\Sigma \models \sigma$ means that σ is a logical consequence of Σ.

5 Rules of Generalization

A *Rule of Generalization* is a rewrite rule such that its application to a query results in a generalization transformation. However, to be meaningful, the rules should have the following properties:

(1) Each application of a rule of generalization must have a simple explanation. Also, changes in the syntax of the query should be few. This makes it easy to explain the generalization easily. Such a consideration argues against indiscriminatingly constructing generalizations based on deductive closures.

(2) In order to perform generalization, the rules of generalization may depend on integrity constraints for the database. The nature of the constraints must be such that they are likely to be specified for a database and are easy to explain. For example, constraints derived from the taxonomic hierarchies are likely to be useful.

(3) The computational overhead in constructing a generalization must be low.

In the remainder of this section, we discuss three classes of generalization all of which meet the constraints mentioned above.

5.1 Conjunct Removal

The removal of one or more of the conjuncts in Q is the simplest approach to generalizing a conjunctive query. This method is computationally inexpensive and satisfies the admissibility condition.

There are many ways in which the transformation of conjunct removal may be accomplished. Some examples of the transformations are given below:

1. *Predicate Removal:* By dropping a conjunct explicitly: $Q_1 \wedge Q_2 \Longrightarrow Q_1$

2. *Variable introduction:*

   (a) By replacing a constant in the query (i.e., a selection) by a new existential variable:
   $P_i(\overline{x_i}, c) \Longrightarrow P_i(\overline{x_i}, y')$

   (b) If the same variable occurs in two conjuncts, we may replace one of the variables by a new existential variable. Thus, an equijoin is replaced by a cartesian product:
   $Q_1(\overline{x}, y) \wedge Q_2(y, \overline{z}) \Longrightarrow Q_1(\overline{x}, y) \wedge Q_2(y', \overline{z})$

The last two rules introduce variables in the query. The variables so introduced (y' in the examples above) must not occur anywhere else in the original query and should be existentially quantified.

Example 5.1: Consider the query $Q(x)$ from example 3.1. We now may construct a generalized query $Q''(x)$

75

by this method of *conjunct removal*

$$Q''(x) \equiv Knows(x, \text{oodb})$$

If we consider conjuncts as filters, conjunct removal corresponds to removal of a filter altogether, instead of decreasing the selectivity of a filter. The following example illustrates how relying on only conjunct removal may lead us to an undesirable generalization.

Example 5.2: Consider Q, Q' (example 3.1) and Q'' (example 5.1). $\forall x(Q' \rightarrow Q'')$ and $\forall x(Q \rightarrow Q')$. If we restrict ourselves to conjunct removal only, we can not discover Q'. Clearly, we would prefer Q' if it is acceptable. In such a case, generalization to Q'' weakens the original query unnecessarily. Intuitively, Q'' is not satisfying because we are not checking for the orthogonal interest area of **engineering**.

We now present examples of other rules of generalizations that do not suffer from this drawback of removing a filter altogether. Unlike conjunct removal, the following transformations are derived from the integrity constraints of the database.

5.2 Predicate Generalization

Predicate generalization modifies a query by replacing one or more positive occurrences of a predicate symbol in the query by another predicate symbol.

The integrity constraints that can be used for this kind of generalization is defined by the following axiom schema:

$$\forall \bar{z}(R(\bar{z}) \rightarrow S(\bar{z})) \qquad (1)$$

Any taxonomic hierarchy of types results in such integrity constraints. Predicate generalization transforms a query posed on a type to a query on its supertype.

Lemma 5.1: If (1) holds, then, the rewrite rule

$$R(\bar{z}) \Longrightarrow S(\bar{z})$$

is a generalization transformation.
□

Lemma 5.1 summarizes the applicability condition and the transformation needed for predicate generalization. This generalization replaces one (or more) positive occurrences of R in the query by S. By transposing axiom (1), we can generalize a negative occurrence of S by a negative occurrence of R.

Example 5.3: The integrity constraint, given below, states that anyone who has published in an area is knowledgeable in that area.

$$\forall x \forall y(Published\_in(x,y) \rightarrow Knows(x,y))$$

In Example 2.1, we want ideally to select those reviewers who have published papers on the topic of the submitted article. Then, the initial query would be:
$$Q(x) \equiv Published\_in(x, \text{cad}) \wedge Published\_in(x, \text{oodb})$$
By using the method of predicate generalization, we may pose the query:

$$Q'(x) \equiv Knows(x, \text{cad}) \wedge Published\_in(x, \text{oodb})$$

Example 5.4: It is possible that a generalization transformation may result in a trivial generalization (i.e., for all databases D, $Q^D = Q'^D$ where Q' generalizes Q). We now present an example of a trivial generalization: Assume that a person must be either a student or a member of staff. A student has additional attributes such as the set of courses he or she takes. Staff do not enroll in courses. We can conceptualize this database in terms of predicates $student(x)$, $staff(x)$, $person(x)$, $courses(x,y)$. Let us consider the query: $Q(x,y) \equiv student(x) \wedge courses(x,y)$. If the occurrence of $student$ is replaced by $person$, then a trivial generalization results because staff do not enroll in courses.

5.3 Constant generalization

Constant generalization replaces an occurrence of a constant in a query by another constant. Let $P(\bar{z}, a)$ be a formula where a constant a occurs. Let $P(\bar{z}, b)$ be the formula where b replaces a in the occurrences of a in $P(\bar{z}, a)$. The axiom schema for the integrity constraints necessary for *constant generalization* is as follows:

$$\forall \bar{z}(P(\bar{z}, a) \rightarrow P(\bar{z}, b)) \qquad (2)$$

The applicability condition and transformation for constant generalization is similar to Lemma 5.1. Given (2), the rewrite rule is:

$$P(\bar{z}, a) \Longrightarrow P(\bar{z}, b)$$

The integrity constraints for constant generalization are likely to be induced by the natural hierarchies or partial orders among domain values. If $\leq$ is a partial order, then the axiom schema (2) may be derived from (3) for all pairs $a \leq b$:

$$\forall \bar{x} \forall y \forall z((y \leq z) \rightarrow (P(\bar{x}, y) \rightarrow P(\bar{x}, z))) \qquad (3)$$

Example 5.5: Examples 2.1 (and 3.1) illustrate the application of constant generalization.

Example 5.6: Consider the problem of selecting reviewers. We have used constant generalization on *Knows* in example 2.1. However, the source of this constant generalization is based on a hierarchy of keywords. Such a hierarchy is used by *ACM* for classification of research articles [Acm 87] for computing survey. We denote the keyword hierarchy relationship by

76

$\leq$. If cad $\leq$ engineering and $Knows$ satisfies (3), then, The following rule of constant generalization is a consequence of the above:

$$\forall x(Knows(x, cad) \rightarrow Knows(x, engineering))$$

If the relation $\leq$ denotes a relationship over a domain such as integers or reals, it may be more effective to generalize in *steps* for computational efficiency. A modification of the axiom schema (3) to use steps is given below:

$$\forall \overline{x}\forall y\forall z((z = y + s) \rightarrow (P(\overline{x}, y) \rightarrow P(\overline{x}, z)))$$

5.4   Neighborhood Generalization

Neighborhood generalization has the effect of replacing every value in an argument position of a predicate by its *neighborhood* (i.e., a set of "nearby" values). For example, the query to find all people of age 40 may be generalized to finding all people of age between 35 and 45. In order to ensure that the transformed query is a generalization, it is necessary that the neighborhood of a value includes itself. We now formalize these ideas.

To use neighborhood generalization for an argument of a predicate, we have to define a neighborhood relation for it. Let N_{P_j} denote the neighborhood relation for the jth argument position of a predicate P. N_{P_j} may be defined either intensionally or extensionally. For domains of type integer or real, the neighborhood relation may be defined intensionally. For example, we could define a neighborhood relation to be an interval of length 10. Such intensional definitions depend on the semantics of the domain. For example, the neighborhood for the dimension of a microchip will be much smaller than that of the population size.

We have pointed out that in order to use the neighborhood relation for generalization, we must ensure that the neighborhood of a value includes itself:

Reflexivity Condition: Let P_j denote the unary relation that is the projection of P onto the jth attribute. Then, N_{P_j} satisfies the reflexivity condition if the following holds:

$$\forall x(P_j(x) \rightarrow N_{P_j}(x, x)) \qquad (4)$$

We observe that the reflexivity condition (4) is weaker than requiring that N_{P_j} be reflexive. The following lemma is a consequence of the reflexivity condition:

Lemma 5.2: If N_{P_j} satisfies (4), then the rewrite rule

$$P(\overline{x_1}, \delta_j, \overline{x_2}) \Longrightarrow P(\overline{x_1}, t, \overline{x_2}) \wedge N_{P,j}(t, \delta_j) \qquad (5)$$

where δ_j occurs in the jth argument position of P, and t is any existential variable that does not occur in the query, is a generalization transformation.

Proof: Since t does not occur in the left hand side of (5), any formula obtained by a specific substitution for t is generalized by the formula on the right hand side of (5). In particular, if we substitute $t = \delta_j$ and then use (4), the formula reduces to the left hand side of (5). Therefore, (5) is a rule of generalization. $\square$

The basic idea is to replace each tuple in a relation by a set of tuples each of which differ from another in the set only in the value of the argument position which is used for neighborhood generalization.

Example 5.6: Let the predicate $Ship(x, y)$ denote that ship x operates in country y. Then, the initial query is to retrieve all ships that operate in India.

$$Q(x) \equiv Ship(x, India)$$

The query Q retrieves the set of all ships which are registered in India. Assume that $Near(x, y)$ is the neighborhood relation for the domain country and for the predicate $Ship$. For the purpose of shipping, the neighboring countries may include Pakistan, China, Burma and Sri Lanka. Then, Q may be generalized to Q' as follows:

$$Q'(x) \equiv Ship(x, y) \wedge Near(y, India)$$

Since the neighborhood relation depends on a domain, there may be additional domain-dependent conditions that further characterizes a neighborhood relation.

5.5   Nature of Rules of Generalization

We have presented several techniques of generalization. The predicate generalization changes a relation symbol into another, a constant generalization substitutes a constant by another constant and the neighborhood generalization introduces a join. In this section, we discuss some of their interesting properties.

All the generalizations introduce *local* modifications to the query. This makes it easy to explain the changes to a query to the user.

The rules of generalization that have been proposed here are mutually *independent* in the following sense. It is not possible to subsume the rewrite rules for one kind of generalization by any set of rules of the other two kinds. For example, no successive applications of predicate and neighborhood generalization can transform the original query into a formula that has been derived using constant generalization. However, the three classes of generalization do interact. Applications of one set of rules may invalidate an application of some other rule of generalization.

The three classes of generalization depend on the semantic knowledge to varying degrees. The technique

of conjunct removal is completely domain independent. Predicate generalization and constant generalizations are *derived* from the integrity constraints. Both these techniques are based on the common hierarchical knowledge structures. Finally, neighborhood generalization must be user-defined.

6 Minimal Generalization

The goal of the generalization operator is to produce an acceptable generalization of a given extended query. However, as stated in Section 3, we would like such a generalization to be minimal. We obtain the definition of minimality in terms of generalization when $\mathcal{R}$ in the definition of minimality is restricted to rules of generalization only. For the following discussion, we assume that for each conjunct, the application of each step in generalization requires *all* the preceding steps in generalization for the conjunct. We call this condition the *ordering hypothesis*. Our algorithm relies on the theorem stated below. The theorem essentially says that in order to obtain a minimal generalization, it is necessary and sufficient that generalization of each conjunct be minimal and that the local minimality condition is reached when ungeneralizing[5] a conjunct one more step makes the query unacceptable. The reason local minimality is sufficient is because all the generalization transformations are local in nature. The correctness of the ungeneralization technique to test minimality relies on the ordering hypothesis. The proof of the theorem is omitted.

Theorem 6.1: Assume that $\mathcal{P}$ in $\langle Q, \mathcal{P} \rangle$ is monotonic. Let Q_1 and Q_2 be two generalizations of Q, which differ only on the generalization of predicate P_i in Q.

$$Q_1 \equiv \Lambda(P_i) \wedge G$$

$$Q_2 \equiv \Pi(P_i) \wedge G$$

where Λ and Π denote applications of sequences of generalization. Further, let Q_1 be acceptable and Q_2 be not. Then, if $\Lambda(P_i)$ is an *immediate generalization*[6] of $\Pi(P_i)$, and the ordering hypothesis holds, then, there is a minimal generalization of Q where P_i is generalized to $\Lambda(P_i)$. $\square$

We use the above theorem to arrive at the following algorithm:

Algorithm 6.1

Input: An extended query: $< Q, \mathcal{P} >$.

---

[5] Assume we generalized P_i to $M(N(P_i))$ where M is a single application of generalization and N represents an arbitrary composition of generalizations. Then, the effect of ungeneralizing $M(N(P_i))$ with respect to P_i is $N(P_i)$. Ungeneralizing P_i with respect to itself returns P_i.

[6] i.e., there is no other generalization Q' of $\Pi(P_i)$ that can be generalized to $\Lambda(P_i)$.

A rule *gen*, when applied, chooses a conjunct and a rewrite rule to generalize Q.
A rule *ungen*, which given a conjunct in the initial query and its current generalization, ungeneralizes it.
Output: A minimal generalization $< Q', \mathcal{P} >$

1. Apply *gen* repeatedly at least until the generalized query is acceptable. Call this query Q_G.

2. Define an order P on the set of conjuncts in Q. Observe that for each conjunct p_i in P, we can identify a unique set of conjuncts S_i in Q_G that corresponds to its generalization.

3. For each $p_i \in P$ do:

   (a) $S_i' := S_i$

   (b) Do

      i. $S_i := S_i'$

      ii. $S_i' := ungen(p_i, S_i)$

      Until $(S_i = p_i)$ or (S_i' is unacceptable)

4. $Q' \equiv \bigwedge_i S_i$

The algorithm above allows us to first overgeneralize a query and to later ungeneralize the query to turn it into a minimal generalization. This flexibility could be important in searching the space of generalization (e.g., in the domain of reals and integers). Also, one could start with a generalization which is easy to evaluate.

The algorithm could be improved in several ways. First, we can exploit special properties of the integrity constraints (e.g., the set of predicate generalizations forms a tree). Next, the properties of the relational query may be used to avoid trivial generalizations (as in Example 5.4). Finally, heuristics based on the meta-data of the specific database (e.g., selectivity, cardinality) may be used to converge on an acceptable query quickly.

7 Related Work

Previous work on query modification has been in the area of semantic query optimization, where the modified query returns the same answer set over all databases [Hamm 80] [Chak 85] [King 81] . The additional constraints that modify the query in semantic query optimization stems from the efficiency requirements. On the contrary, our goal in query modification is to satisfy the acceptance test.

The idea of generalizing a query brings up the issue of *ranking* the output of the extended query. For example, the tuples contributed by the original query may be considered to be of "better quality". There has been

significant amount of work in *information retrieval* that address this issue of ranking the output which may be adapted in our framework [Sal 83] [Smi 79]. The notion of *preference* has been used by Lacroix [Lac 87] to prune the set of answers generated to a relational query. The preference conditions are user specified and the semantic relationships are not utilized.

Motro [Mot 86] has defined a notion of generalization in the context of an object-oriented data model to avoid null answers to queries. We have provided a more general framework that can accommodate other modification operators. We also treat additional aspects of generalization such as monotonicity. The later work by Motro [Mot 88] addresses the issue of intensionally defining the neighborhood relation.

Generalization of queries shares some similarity to *concept acquisition*, whose goal is to induce general descriptions of concepts from specific instances of the concepts [Mic 83]. The rules of generalization are similar. However, the acceptance test in concept acquisition is to generalize the class description so that it includes all the training events. The restricted form of acceptance makes it possible to utilize efficient algorithms.

8   Conclusion

We have proposed extended query as a mechanism to capture the iterative model of query modification. Every extended query has a flexible query and an acceptance test. The flexible component of the query is modified by using a set of query modification operators so that the acceptance test is satisfied. The concept of minimality is used to keep the modification as tight as possible.

Generalization is an example of a useful query modification operator. We provided rewrite rules for generalization. The rules of generalization are based on the knowledge structures that are commonly available and are therefore useful in practice. We have outlined an algorithm to pick a minimal generalization.

Current Status and Future Work: We are experimenting with the techniques of query generalization for developing an application for selecting reviewers for a technical conference. We have utilized the ACM keyword hierarchy [Acm 87] and a large bibliographic database maintained at Stanford University.

The next goal is to experiment with other generic query modification operators and to study the strategy for applying different operators in our framework. We also need to support efficient execution of the extended queries The useful rewrite rules may be compiled at the time of schema definition for efficiency. Opportunities to reuse the answers already generated should

be exploited. The technique of materialization and indexing of views [Rou 81] and common subexpression analysis [Fin 82] can be profitably used. Providing explanation and user-friendly interfaces is another consideration in system design.

References

[Acm 87] ACM; "The Full Computing Reviews Classification System", ACM, 1987.

[Chak 85] Chakraborty, U.S; "Semantic Query Optimization in Deductive Databases", PhD thesis, University of Maryland, July 85.

[Fin 82] Finkelstein, S.; "Common Expression Analysis in Database Applications"; Proceedings of ACM SIGMOD 1982.

[Hamm 80] Hammer, M., Zdonik, S.; "Knowledge-based query processing", Proceedings of VLDB, Oct 80.

[Hin 72] Hindley J.R. et.al.; "Introduction to Combinatory Logic", Cambridge University Press, 1972.

[King 81] King, J.; "Query Optimization by Semantic Reasoning", PhD thesis, Stanford University, May 81.

[Lac 87] Lacroix, M., Lavency, P.; "Preferences: Putting More Knowledge into Queries", Proceedings of VLDB, 1987.

[Mic 83] Michalski R.S., Carbonell J., Mitchell T.; "Machine Learning: An Artificial Intelligence Approach", Tioga Publishing Company, 1983.

[Mot 86] Motro, A.; "Query Generalization: A Method for Interpreting Null Answers", in Expert Database Systems, Edited by Larry Kerschberg, 1986.

[Mot 88] Motro, A.; "Vague: A User Interface to Relational Databases that Permit Vague Queries", ACM transactions on Office Information Systems, July 1988, Vol 16, No. 3.

[Rou 81] Roussopoulos N.; "View Indexing in Relational Databases", Technical Report TR-1046, University of Maryland, April 1981.

[Sal 83] Salton G. et.al.; "Extended Boolean Information Retrieval", CACM, Dec 1983.

[Smi 79] Smith L.; "Selected Artificial Intelligence Techniques in Information Retrieval Systems Research", Ph.D Thesis, Syracuse University, 1979.

Resolving Database Incompatibility:

An Approach to Performing Relational Operations over Mismatched Domains

by

Linda G. DeMichiel

Abstract

We present a solution to the problem of supporting relational database operations despite domain mismatch. Mismatched domains occur when information must be obtained from databases that were developed independently. We resolve domain differences by mapping conflicting attributes to common domains by means of a mechanism of virtual attributes and then apply a set of extended relational operations to the resulting values. When one–one mappings cannot be established between domains, the values that result from attribute mappings may be partial. We define a set of extended relational operators that formalize operations over partial values and thus manipulate the incomplete information that results from resolving domain mismatch.

Index Terms

Databases, domain mismatch, extended relational algebra, federated databases, incomplete information, virtual attributes.

1. Introduction

The increased need for communication among independent computer systems makes it necessary to derive information from multiple database sources. These sources may range from relations and relation fragments within a single distributed database system [4] to relations from several independently developed databases or multidatabase systems [12]. If databases have been established independently, however, they are likely to differ in their representation and encoding of similar information.

It is thus often necessary to perform operations over nonhomogeneous, or disparate, domains. In particular, we want to perform operations over domains that are seemingly incompatible (because of type conflicts, for example) but that are semantically similar.

Type conflicts may result from different physical representations of data; more important, however, although two types may be semantically related and possibly also use the same physical representation, they may differ in their precise semantics.

While the need for operations over mismatched domains is clearly most pronounced in a multidatabase environment, this requirement can also arise within a single database, such as when its conceptual design is no longer adequate to the demands placed upon it for the generation of new information. Furthermore, it may be desirable to use a database for a purpose that was not foreseen in its original design, such as to gather historical data. For example, current encodings for disease categories in medical databases conflict with the encodings used 20 years ago. By using knowledge about the domain of the database, it is often possible to overcome limitations of the original structure and to exploit the data to extract new information.

2. Database Integration

Work in the area of schema integration in heterogeneous databases has recognized the importance of solving the problem of domain mismatch [7] but has provided only limited solutions.

While prior work on the related topic of database integration has considered the issues of schema mismatch in heterogeneous databases, it makes some rather strong assumptions about the domains of the data involved. Most important, these efforts deal only with the situations in which it is possible to establish one–one mappings between domains or many–one mappings in which an element of the source domain can be mapped to a unique element of the target domain. In practice, however, this is often not so. Consider, for example, the problem of mapping between zip codes and town names. Many towns have multiple zip codes, while some zip codes cover multiple towns.

Solutions for the cases in which straightforward data type conversions suffice or in which an element of the source domain can be mapped to a unique element of the target domain have been demonstrateu   Ozejdo, Embley, and Rusinkiewicz [6], by Breitbart, Olson, and Thompson [3], and by   empleton et al. [15].

Litwin and Vigier [13] pres it a system that allows the user to define *dynamic attributes* in order to execute queries over mismatched domains. A dynamic attribute may be explicitly defined in the query in which it occurs, or its definition may be stored as an executable program and invoked from within the query. Litwin and Vigier use dynamic attributes to define one–one and many–one mappings between domains. After dynamic

attribute mappings have been applied to a relation, the standard relational operations can be invoked. Our notion of virtual attributes expands upon Litwin and Vigier's notion of dynamic attributes. We generalize the concept to address the problems of semantic domain mismatch in cases where one–one and many–one mappings are not possible. The removal of the restriction that dynamic attributes be associated with complete information and the extension to the concept to include partial values then lead us to the introduction of an extended relational algebra to handle the resulting incomplete information in a uniform way.

3. An Approach Using Virtual Attributes and Partial Values

In this paper we present a more general solution to the problem of performing operations over mismatched domains. Our solution is designed to handle operations over mismatched domains in cases where it is not possible to map a value in one domain to a definite and unique value in another.

The approach we take makes use of the mechanisms of *domain mappings* and *virtual attributes*. A virtual attribute, like a real attribute, denotes the property of some entity and is associated with a particular domain. It is derivable from other attributes in the database or from other information associated with the database. By mapping real attributes to virtual attributes of the appropriate domain type, we can place relations in union-compatible form or in a form suitable for the correct execution of the desired queries.

The domain mapping definitions are registered and stored within the database. They exist as a layer above the individual database schemas to allow these schemas to be integrated with the schemas of other databases. The use of virtual attribute mappings is thus fully compatible with an environment of autonomous databases. The choice of the domain mapping mechanism is motivated by the desire to obtain data integration while remaining within the relational model and while requiring no modification to the structure or schema of any foreign database.

When a real attribute value in a domain cannot be mapped to a single definite value, a *partial value* may result; that is, it may be possible to characterize the result as a set of values of which *exactly one* must be correct. When operations over partial values are performed, they in turn may produce *maybe results* [5]. A maybe result tuple, or *maybe tuple* [2], is a tuple that cannot be excluded from the result of a query but that is not known with certainty to belong to it.

By extending the operations of the relational algebra to partial values and maybe tuples, we provide a uniform mechanism for performing operations over mismatched domains.

Performing operations over domains corresponding to virtual attributes is a special case of the general problem of performing operations over indefinite values [11]. Extensions of the relational algebra to allow operations over indefinite values have been proposed by Codd [5], Grant [9], Biskup [2], Maier [14], and others. By defining our operators to handle the particular demands of virtual attribute mappings, we are able to obtain more precise results in this area.

In the following sections of this paper, we briefly describe the steps involved in our mechanism, consider an example that illustrates the application of the extended operations, and present the definitions of a number of the extended operators.

4. Overview of the Process

The following steps are involved in performing operations over mismatched domains by means of domain mappings and virtual attributes. These steps will be illustrated by the example of the following section and then will be described in greater detail.

1. Convert attributes.
   Real attributes and values are mapped to virtual attributes in order to resolve domain differences and to place the relations in a form suitable for the execution of the desired queries. The values that result from these mappings may be partial or total.

2. Perform extended relational operations, producing partial results.
   The extended relational operators are applied to the resulting relations, which may contain partial values. The application of the extended relational operators may produce results that are also partial. Furthermore, application of the selection and join operators to tuples containing partial values may generate maybe tuples.

3. Continue further computation, considering the partial results and maybe tuples.
   Further computation may involve applying the extended relational operators to relations containing maybe tuples as well as partial values.

4. Present the final results.

5. A Motivating Example

In the following example, we show how the manipulation of virtual attribute mappings and partial values can be used to synthesize and extract information that is not available by direct means.

Suppose we have two different relations with information on restaurants [12]. These relations may or may not reside in different databases. One gives the location of the restaurants by city but is otherwise not specific about location. It is, however, very specific about the type of food available. The other contains information about restaurants in San Francisco. It is more precise about location but is otherwise very general in its categorization. The schemas of these two relations are Chinese-Restaurants(restaurant, city, type) and SF-Restaurants(restaurant, address, category) respectively. Some of the data contained in these relations is given below.

We would like to eat Hunan food in North Beach.

Chinese-Restaurants		
restaurant: name	city: city	type: cuisine
Brandy Ho	SF	Hunan
Hunan	SF	Hunan
Mandarin	SF	Mandarin
China West	SF	Cantonese

SF-Restaurants		
restaurant: name	address: address	category: category
Brandy Ho	217 Columbus	Chinese
Empress of China	838 Grant	Chinese
Five Happiness	309 Clement	Chinese
China West	2332 Clement	Chinese
Asia Garden	772 Pacific	Chinese

We have the following information about the relationship between street address and

locations:

Locations	
address: street	location: area
Grant	Chinatown
Columbus	NorthBeach
Clement	Richmond
Pacific	Chinatown

Additionally, if a restaurant is Chinese, we believe that it specializes in either Cantonese, Hunan, Mandarin, or Szechwan cuisine. That is, we define a domain mapping from the domain category to the domain cuisine in which the value Chinese results in the partial value [Cantonese, Hunan, Mandarin, Szechwan].

Given this information, we can now derive from the original relations the following two union-compatible relations, Chinese-2 and SF-2:

Chinese-2		
restaurant: name	location: area	type: cuisine
Brandy Ho	[]	Hunan
Hunan	[]	Hunan
Mandarin	[]	Mandarin
China West	[]	Cantonese

SF-2		
restaurant: name	location: area	type: cuisine
Brandy Ho	NorthBeach	[Cantonese,Hunan, Mandarin,Szechwan]
Empress of China	Chinatown	[Cantonese,Hunan, Mandarin,Szechwan]
Five Happiness	Richmond	[Cantonese,Hunan, Mandarin,Szechwan]
China West	Richmond	[Cantonese,Hunan, Mandarin,Szechwan]
Asia Garden	Chinatown	[Cantonese,Hunan, Mandarin,Szechwan]

The relation Chinese-2 is obtained from the relation Chinese-Restaurants by applying a domain mapping to the attribute city to generate the virtual attribute location,

whose domain is area. Since we have no means of deriving the location of a restaurant given the information in Chinese-Restaurants, the values in the location column are fully unspecified partial values (or null values). They can correspond to any element in the domain area. We use the empty brackets ([]) as a notational shorthand to denote these fully unspecified partial values. We have projected out the city attribute.

The relation SF-2 is obtained from the relation SF-Restaurants by joining SF-Restaurants and Locations on the attribute address (we assume for now that we have additional mappings that reconcile the domain types address and street), by performing a mapping from the real attribute category to the virtual attribute type, whose domain is cuisine, and by projecting out the attributes address and category. The mapping from category to type results in the partial value [Cantonese, Hunan, Mandarin, Szechwan].

The result of the query $\sigma'_{type=Hunan \wedge location=NorthBeach}$Chinese-2 is given below. Note that the *status* column does not correspond to an attribute. It is used to distinguish between true and maybe tuples.

restaurant: name	location: area	type: cuisine	status
Brandy Ho	NorthBeach	Hunan	maybe
Hunan	NorthBeach	Hunan	maybe

The result of the query $\sigma'_{type=Hunan \wedge location=NorthBeach}$SF-2 is

restaurant: name	location: area	type: cuisine	status
Brandy Ho	NorthBeach	Hunan	maybe

Thus, when we query these two relations separately, we obtain from the first that Brandy Ho and Hunan *may* be the answers to our search, if indeed they are in North Beach. We know that they are both Hunan restaurants, but we are not certain about their locations. From the second relation we obtain that Brandy Ho *may* be an answer because it is in North Beach. However, we do not have definite information about the type of food that if offers.

If, however, applying our *generalized* union operator, we first take the union of the two relations Chinese-2 and SF-2, we obtain the following result, SF-Chinese:

SF-Chinese		
restaurant: name	location: area	type: cuisine
Brandy Ho	NorthBeach	Hunan
Empress of China	Chinatown	[Cantonese,Hunan, Mandarin,Szechwan]
Five Happiness	Richmond	[Cantonese,Hunan, Mandarin,Szechwan]
China West	Richmond	Cantonese
Asia Garden	Chinatown	[Cantonese,Hunan, Mandarin,Szechwan]
Hunan	[]	Hunan
Mandarin	[]	Mandarin

If we now pose our query, it is clear that Brandy Ho is a definite answer to our search, and that the Hunan *may* be a possibility, if in fact it is in North Beach. Thus, we have been able to extract more information from our data.

The result of the query $\sigma'_{type=Hunan \wedge location=NorthBeach}$SF-Chinese is

restaurant: name	location: area	type: cuisine	status
Brandy Ho	NorthBeach	Hunan	true
Hunan	NorthBeach	Hunan	maybe

In the next sections we present an extended relational algebra that formalizes these operations.

6. Extending the Relational Algebra

In order to perform relational operations over virtual attributes, we need to extend the definition of the relational operators to handle the partial information that arises from virtual attribute mappings. The virtual attribute mappings that we consider map a single real attribute value to a finite set of virtual attribute values. We will assume here definite source databases; thus, the indefinite values that occur arise only as a result of virtual attribute mappings.

6.1 *Definite Values and Partial Values*

We say that a tuple is *definite*, or *fully determined*, if it contains no incomplete information on any attribute. We say that a tuple whose value on a particular attribute is not definite is *indefinite* on that attribute. An indefinite value can mean that no information at all is known about the value or that no value is appropriate for that attribute. In such cases, indefinite values are generally represented by *nulls* [1]. The notion of *set null* has been introduced to denote a null whose value can be constrained to be one of a finite set of values [10].

To distinguish the particular semantics that we attach to the indefinite virtual attribute values that arise from domain mappings and from the operations of our extended relational algebra, we will refer to them as *partial values*. A partial value corresponds to a finite set of possible values such that the "true" or "real" value of the partial value is exactly one of the values in that set.

A *total relation* is a relation in which each tuple is definite and unique. A *partial relation* is a relation containing zero or more tuples with partial values. An *original relation* is any source relation to which virtual attribute mappings have not yet been applied.

In operations over partial values, we need to be able to distinguish when two partial values correspond to the same possible values, and when they actually correspond to the same unique real value. If the partial values correspond to sets of distinguishable values, where the value-bearing elements are accessible in some way, we can do so as follows.

If η is a partial value and ν denotes the function from a partial value to the finite set of values to which it corresponds, we say that a value v is an *element* of partial value η, or $v \in \eta$, if $v \in \nu(\eta)$.

We say that two partial values η_1 and η_2 are *equal* (=) if they must necessarily correspond to the same true or real value. Thus $\eta_1 = \eta_2$ if $\nu(\eta_1) = \nu(\eta_2)$ and $|\nu(\eta_1)| = |\nu(\eta_2)| = 1$; that is, two partial values are regarded as equal if they are each of cardinality 1 and correspond to the same identical real value.

We extend the equality comparison operator = to the comparison of definite values and partial values as follows. We say that a definite value d and a partial value η are equal (=) if $(|\nu(\eta)| = 1) \wedge (i \in \nu(\eta) \Rightarrow i = d)$. That is, we regard a definite value and a partial value of cardinality 1 as equal when they correspond to the same value. We assume that conversions can be freely made between partial values of cardinality 1 and definite values

as required. In cases where it is necessary to distinguish a partial value whose cardinality exceeds 1, we will refer to such a partial value as a *proper partial value*.

In many cases, two partial values will correspond to the same set of *possible* values, although they may not be equal (=). That is, although $\nu(\eta_1) = \nu(\eta_2)$, it is not necessarily the case that $\eta_1 = \eta_2$.

If η is a partial value and $\nu(\eta) = \{\phi_1, \phi_2, \ldots, \phi_n\}$, then we also use the notation $[\phi_1, \phi_2, \ldots, \phi_n]$ to refer to the partial value. If $\{v_1, v_2, \ldots, v_n\}$ are all the elements in a domain D, then in our examples we will use the shorthand notation [] to denote the partial value $[v_1, v_2, \ldots, v_n]$ on an attribute whose domain is D—that is, a partial value whose true or real value can correspond to any element of the given domain.

We define $\eta_1 \cap \eta_2$ as η_3, such that $\nu(\eta_3) = \nu(\eta_1) \cap \nu(\eta_2)$. We define $\eta_1 \cup \eta_2$ as η_3, such that $\nu(\eta_3) = \nu(\eta_1) \cup \nu(\eta_2)$.

We extend the $\cap$ operator to operate over definite values and partial values as follows. Where one of ξ_1, ξ_2 is definite and the other is a partial value, we define $\xi_1 \cap \xi_2 = \xi_1 = \xi_2$ iff $\xi_1 = \xi_2$; $\xi_1 \cap \xi_2 = \xi_2$ if $(|\xi_2| = 1 \wedge \xi_2 \in \xi_1)$; and $\xi_1 \cap \xi_2 = \xi_1$ if $(|\xi_1| = 1 \wedge \xi_1 \in \xi_2)$.

Similarly, we extend the $\cup$ operator to operate over definite values and partial values as follows. Where one of ξ_1, ξ_2 is definite and the other is a partial value, we define $\xi_1 \cup \xi_2 = \xi_1 = \xi_2$ iff $\xi_1 = \xi_2$; $\xi_1 \cup \xi_2 = \xi_1$ if $(|\xi_2| = 1 \wedge \xi_2 \in \xi_1)$; and $\xi_1 \cup \xi_2 = \xi_2$ if $(|\xi_1| = 1 \wedge \xi_1 \in \xi_2)$.

Finally, we define an empty partial value, $\emptyset$, such that $\eta = \emptyset$ iff $|\nu(\eta)| = 0$. Our extended operations, however, never result in the production of empty partial values. The attempted generation of an empty partial value is an indication of an error condition or an inconsistency among relations.

6.2 *True and Maybe Results*

When our extended relational operators are applied to partial relations, such as relations that result from the application of domain mappings, they will normally produce results that are also partial. Following Codd [5], we partition the results of these operations into two classes: *true* results and *maybe* results. The distinction between true and maybe results is independent of the distinction between definite and partial values. The true result of an operation consists of all those tuples that must be contained in the result. The maybe result of an operation consists of all those tuples that cannot be excluded from the result but that are not known with certainty to be contained in the result. The true

result and the maybe result of an operation are thus defined so that their intersection is empty. Depending on the operation itself, incomplete information may occur in the true result, the maybe result, or both. Note that the standard relational operators, which are defined only over total relations, produce results that consist exclusively of true tuples.

7. Definition of the Extended Operators over Partial Values

Our presentation makes use of the two relations $r(R)$ and $s(R)$. The relations r and s are assumed to be union-compatible. Let X, $X \subset R$ denote the designated key in both r and s, and let Z denote the set of attributes $R - X$. We assume that all attributes contained in the designated key are definite. Let C denote some attribute in Z. This attribute may be either real or virtual. If C is a virtual attribute, we assume that C may be indefinite.

The relations are assumed to be in third normal form. They are assumed to be in chased form with regard to the virtual attributes.

We say that a relation r over schema R (denoted as $r(R)$) is in *chased form* [16] if $X \subset R$ denotes the set of attributes in the designated key, $Z \subset R$ denotes the set of attributes $R - X$, and there are no two tuples $t \in r$, $u \in r$ such that $t = u$ or such that $t.X = u.X$ and $t.Z \neq u.Z$.

We define our relational operators in such a way that, with few exceptions, the result relations are in chased form whenever the source relations are in chased form, and thus it is not necessary to perform a separate chase operation. When this is not the case, it will be explicitly noted.

We say that two chased and union-compatible relations $r(R)$ and $s(R)$ are *consistent* if $X \subset R$ denotes the set of attributes of the designated key of both r and s, Z denotes the set of attributes $R - X$, and $(\forall u)(\forall v)((u \in r \land v \in s \land u.X = v.X) \Rightarrow (\forall C)(C \in Z \Rightarrow u.C \cap v.C \neq \emptyset))$.

The extended operators are defined only for union-compatible and consistent relations. If, in a given implementation, relations are found to be inconsistent as a result of attempting to apply an extended operator, the action taken should depend on that implementation. For example, in some implementations it might be reasonable to signal an error and to wait for user intervention, whereas in others it might be appropriate to modify the source data.

We show the extended definitions for the union, select, project, and natural join operations. Definitions for the full set of relational operations are given in [8].

7.1 *Union*

Let the source relations be $r(XZ)$ and $s(XZ)$, where X is the key and Z the set of non-key attributes.

The true result of the extended operation $r \cup' s$ is

$$\{t \mid (\quad (t \in r \wedge (\not\exists u)(u \in s \wedge (u.X = t.X)))$$
$$\vee (t \in s \wedge (\not\exists u)(u \in r \wedge (u.X = t.X)))$$
$$\vee (\exists u)(\exists v)(u \in r \wedge v \in s \wedge (t.X = u.X = v.X)$$
$$\wedge (\forall C)(C \in Z \Rightarrow (t.C = u.C \cap v.C)))) \}$$

The maybe result is $\emptyset$.

The relations r and s are inconsistent if

$$(\exists u)(\exists v)(u \in r \wedge v \in s \wedge (u.X = v.X) \wedge (\exists C)(C \in Z \wedge (u.C \cap v.C = \emptyset)))$$

The extended union operator requires that if two union-compatible relations each contain an entry with a given key, all non-key fields must be consistent. If two tuples with the same key each contain a partial value on a given field, we assume that if the source information is consistent, it is correct, and thus the value of the result tuple on that field must be given by the intersection of those partial values.

For example, if r and s are the relations given below, where X is the key and C a non-key attribute,

r	
X	**C**
1	[a,b]
2	[g,h]
3	[a,b,c,d]

s	
X	**C**
1	[b,c]
3	[b,c,d,e]

then the result $q = r \cup' s$ is

q	
X	C
1	b
2	[g,h]
3	[b,c,d]

Depending on database policies and the degree of trust a system places on its source data, these results may or may not be used to refine the data contained in the source relations r and s.

7.2 *Selection over an Attribute, C*

Let the source relation be $r(XC)$, where X is the key and C is a single attribute.

The true result of the extended operation $\sigma'_{C=z} r$, where z is a constant, possibly a partial value, is

$$\{t \mid t \in r \wedge (t.C = z)\}$$

If the constant z is a proper partial value, the true result is thus empty.

The maybe result is

$$\{t \mid (\exists u)(u \in r \wedge (u.C \cap z \neq \emptyset) \wedge (t.X = u.X) \wedge (t.C = u.C \cap z))\}$$
$$- \{t \mid t \in r \wedge (t.C = z)\}$$

Note that the maybe result excludes tuples contained in the true result.

For the special case where z is a definite value, the maybe result is

$$\{t \mid (\exists u)(u \in r \wedge (z \in u.C) \wedge (t.X = u.X) \wedge (t.C = z))\}$$
$$- \{t \mid t \in r \wedge (t.C = z)\}$$

7.3 *Projection over an Attribute, C*

Let the source relation be $r(XC)$, where X is the key and C is a single attribute.

The true result of the extended operation $\pi'_C r$ is

$$\{t \mid (\exists u)(u \in r \wedge (t = u.C))\}$$

The maybe result is $\emptyset$.

If C corresponds to an attribute that contains partial values, and duplicates are eliminated from the resulting relation, information may be lost. The source relation may contain two or more tuples that have partial values that correspond to the same *possible* values on attribute C. Were the information in that relation to be complete, these partial values might or might not correspond to different definite values. For this reason, it is important to retain the key when a series of extended operations is to be performed.

7.4 *Natural Join*

Let the source relations be $r(XC)$ and $s(YC)$, where X and Y are keys and C is a single attribute.

The true result of the extended operation $r \bowtie' s$ is

$$\{t \mid (\exists u)(\exists v)(u \in r \wedge v \in s \wedge (t.X = u.X) \wedge (t.Y = v.Y)$$
$$\wedge (t.C = u.C = v.C))\}$$

The maybe result is

$$\{t \mid (\exists u)(\exists v)(u \in r \wedge v \in s \wedge (t.X = u.X) \wedge (t.Y = v.Y)$$
$$\wedge (t.C = u.C \cap v.C \neq \emptyset))\}$$
$$- \{t \mid (\exists u)(\exists v)(u \in r \wedge v \in s \wedge (t.X = u.X) \wedge (t.Y = v.Y)$$
$$\wedge (t.C = u.C = v.C))\}$$

7.5 *Properties of the Extended Relational Operators over Partial Values*

All four of the extended operators defined above—union, selection on a definite value, projection, and natural join—are faithful [14] to the corresponding standard operators. That is, the extended relational operators listed above are defined to be identical to the

corresponding standard relational operators on all union-compatible total relations containing only definite tuples.

When the extended relational operators are applied to indefinite relations, information loss in the form of maybe tuples can result from the extended selection and extended natural join operations when partial values occur in the selection and join domains respectively.

8. Extending the Relational Operators to Maybe Tuples

In this section we present an extension to the relational operators shown above to include operations over maybe tuples.

We will first present a definition of the extended operators over partial values and maybe tuples and then will explain some of our motivations.

As before, we assume the relations $r(R)$ and $s(R)$ are union-compatible. Let X, $X \subset R$ denote the designated key in both r and s; let Z denote the set of attributes $R - X$. We assume that tuples resulting from extended relational operations over partial values carry a status designation $\{true, maybe\}$, and that $status(t)$ gives access to the status designation for a given tuple t. This status designation is not part of the value of the tuple.

8.1 *Union*

The true result of the extended operation $r \cup' s$ is

$$
\begin{aligned}
\{\, t \mid (\quad & (t \in r \wedge (status(t) = true) \\
& \wedge (\not\exists u)(u \in s \wedge (status(u) = true) \wedge (u.X = t.X))) \\
\vee \, & (t \in s \wedge (status(t) = true) \\
& \wedge (\not\exists u)(u \in r \wedge (status(u) = true) \wedge (u.X = t.X))) \\
\vee \, & (\exists u)(\exists v)(u \in r \wedge v \in s \\
& \wedge (status(u) = true) \wedge (status(v) = true) \\
& \wedge (t.X = u.X = v.X) \\
& \wedge (\forall C)(C \in Z \Rightarrow (t.C = u.C \cap v.C)))))\}
\end{aligned}
$$

Since if true tuples are consistent they are assumed to be correct, we combine them. Thus true tuples refine the incomplete information of other true tuples.

The maybe result is

$$\{ t \mid (\quad (t \in r \wedge (status(t) = maybe)$$
$$\wedge (\not\exists u)(u \in s \wedge (t.X = u.X)))$$
$$\vee (t \in s \wedge (status(t) = maybe)$$
$$\wedge (\not\exists u)(u \in r \wedge (t.X = u.X)))$$
$$\vee (\exists u)(\exists v)(u \in r \wedge v \in s \wedge (u.X = v.X)$$
$$\wedge (status(u) = maybe) \wedge (status(v) = maybe)$$
$$\wedge (t.X = u.X)$$
$$\wedge (\forall C)(C \in Z \Rightarrow (t.C = u.C \cup v.C)))) \}$$

Maybe tuples do not refine the incomplete information of any other tuples.

Note that the result may contain fewer maybe tuples than the two source relations combined. Every maybe tuple in the result corresponds to a maybe tuple in one or both of the source relations.

The relations r and s are inconsistent if

$$(\exists u)(\exists v)(u \in r \wedge v \in s \wedge (u.X = v.X)$$
$$\wedge (status(u) = true) \wedge (status(v) = true)$$
$$\wedge (\exists C)(C \in Z \wedge (u.C \cap v.C = \emptyset)))$$

8.2 *Selection over an Attribute, C*

The true result of the extended operation $\sigma'_{C=z}r$, where z is a constant, possibly a partial value, is

$$\{ t \mid t \in r \wedge (t.C = z) \wedge (status(t) = true) \}$$

The maybe result is

$$\{ t \mid (\exists u)(u \in r \wedge (t.X = u.X) \wedge (u.C \cap z \neq \emptyset) \wedge (t.C = (u.C \cap z))) \}$$
$$- \{ t \mid t \in r \wedge (t.C = z) \wedge (status(t) = true) \}$$

8.3 *Projection over an Attribute, C*

The true result of the extended operation $\pi'_C r$ is

$$\{t \mid (\exists u)(u \in r \wedge (status(u) = true) \wedge (t = u.C))\}$$

The maybe result is

$$\{t \mid (\exists u)(u \in r \wedge (status(u) = maybe) \wedge (t = u.C))\}$$

Note that if duplicate tuples are eliminated from the true result and from the maybe result to bring each into chased form, information may be lost.

8.4 *Natural Join*

The true result of the extended operation $r \bowtie' s$ is

$$\{t \mid (\exists u)(\exists v)(u \in r \wedge v \in s$$
$$\wedge (status(u) = true) \wedge (status(v) = true)$$
$$\wedge (t.X = u.X) \wedge (t.Y = v.Y)$$
$$\wedge (t.C = u.C = v.C))\}$$

The maybe result is

$$\{t \mid (\exists u)(\exists v)(u \in r \wedge v \in s$$
$$\wedge (t.X = u.X) \wedge (t.Y = v.Y)$$
$$\wedge (t.C = u.C \cap v.C \neq \emptyset))\}$$
$$- \{t \mid (\exists u)(\exists v)(u \in r \wedge v \in s$$
$$\wedge (status(u) = true) \wedge (status(v) = true)$$
$$\wedge (t.X = u.X) \wedge (t.Y = v.Y)$$
$$\wedge (t.C = u.C = v.C))\}$$

9. The Meaning of Maybe Tuples

As we noted above, a maybe tuple represents information that *may* not be true. Thus, it cannot be used to refine the information contained in true tuples. A more subtle point, however, is that it also cannot be used to refine the information contained in other maybe tuples or to detect inconsistencies among the maybe tuples of different relations.

Consider the following example. As before we assume that the relations $r(R)$ and $s(R)$ are union-compatible and that X, $X \subset R$, denotes the designated key in both r and s.

r		
X	C	status
1	[a,b,c]	true
2	[b,c,d]	true

s		
X	C	status
1	[a,b,c]	true
2	[b,c,d]	true
3	[c,d,e]	true

By our definitions of the extended operators, the result of $\sigma'_{C=c}s$ is

X	C	status
1	c	maybe
2	c	maybe
3	c	maybe

Consider now $q = r \cup' (\sigma'_{C=c}s)$. It seems obvious that the first two maybe tuples in the result of $\sigma'_{C=c}s$ should not override the information contained in the true tuples of r. They thus do not affect the result. The result is

q		
X	C	status
1	[a,b,c]	true
2	[b,c,d]	true
3	c	maybe

However, consider $(\sigma'_{C=b}r) \cup' (\sigma'_{C=c}s)$. The result of $\sigma'_{C=b}r$ is

X	C	status
1	b	maybe
2	b	maybe

Because a maybe tuple represents a result that *may* be true, all maybe tuple values need to be reflected in the result of an operation over maybe tuples. Thus, the query $(\sigma'_{C=b}r) \cup' (\sigma'_{C=c}s)$ results in the following relation q, even though the tuples whose X values are 1 and 2 are seemingly inconsistent. The relation q is shown in unchased form:

q		
X	C	status
1	b	maybe
2	b	maybe
1	c	maybe
2	c	maybe
3	c	maybe

If we define the union operator over maybe tuples as given in section **8.1**, the result of $(\sigma'_{C=b}r) \cup' (\sigma'_{C=c}s)$ is

q		
X	C	status
1	[b,c]	maybe
2	[b,c]	maybe
3	c	maybe

This result is in chased form and is semantically equivalent to q.

Note further that when we execute the query $(\sigma'_{C=b}r) \cup' (\sigma'_{C=c}r)$, the result w is

w		
X	C	status
1	[b,c]	maybe
2	[b,c]	maybe

which is the same as the result of $\sigma'_{C=[b,c]}r$ and thus corresponds to our intuitions.

Let us now consider another example, this time involving the set difference operation. We make use of the following relations r and s:

r		
X	C	status
1	[a,b,c]	true
2	[a,b,c]	true

s		
X	C	status
1	[a,b,c]	true
2	[a,b,c]	true

Since X is the key and since we believe the two relations to be consistent, it should follow that $q = r -' s = \emptyset$.

Consider $\sigma'_{C=b}s$. The result of $\sigma'_{C=b}s$ is

X	C	status
1	b	maybe
2	b	maybe

Consider now $r -' \sigma'_{C=b}s$. If the C values of the tuples in s are in fact b, the C values of the tuples in r must be b also, or the two relations would be inconsistent. Thus, it follows that if C values are b, the result of $r -' (\sigma'_{C=b}s)$ must be $\emptyset$. If, however, the C values of these tuples are not b, the result of $r -' (\sigma'_{C=b}s)$ must be

X	C
1	[a,c]
2	[a,c]

In other words, $r -' (\sigma'_{C=b}s) = q$, where q is the following relation:

q		
X	C	status
1	[a,c]	maybe
2	[a,c]	maybe

The following definition of the maybe result of the set difference operator on maybe tuples reflects this semantics:

$$
\begin{aligned}
\{t \mid (\ & (\exists u)(u \in r \wedge (status(u) = maybe) \\
& \wedge (\not\exists v)(v \in s \wedge u.X = v.X) \\
& \wedge (t = u)) \\
\vee\ & (\exists u)(u \in r \wedge (status(u) = maybe) \\
& \wedge (\exists v)(v \in s \wedge (status(v) = maybe) \\
& \qquad \wedge (t.X = u.X = v.X) \\
& \qquad \wedge (\forall C)(C \in Z \Rightarrow (t.C = u.C - v.C \neq \emptyset)))) \\
\vee\ & (\exists u)(u \in r \wedge (status(u) = true) \\
& \wedge (\exists v)(v \in s \wedge (status(v) = maybe) \\
& \qquad \wedge (\forall C)(C \in Z \Rightarrow (u.C \cap v.C \neq \emptyset)) \\
& \qquad \wedge (t.X = u.X = v.X) \\
& \qquad \wedge (\forall C)(C \in Z \Rightarrow (t.C = u.C - v.C \neq \emptyset)))))\}
\end{aligned}
$$

10. Preserving Additional Information

By maintaining information about the origins of maybe tuples, it is possible to characterize the conditions whose satisfaction guarantees that the maybe tuples are in fact part of the true result of a given operation. It is thus possible to preserve additional information when performing a series of operations involving partial values.

Sometimes it is useful to distinguish among individual partial values. Partial values that are *marked* are distinguished from all other partial values that do not bear an identical mark [14]. If two partial values bear the same identical mark, they are identical and therefore denote the same identical real value. The use of marked partial values may allow more information to be retained when the relational operators are applied to partial relations. They can, however, significantly increase the complexity of these operations without necessarily increasing the information content of the result.

Mechanisms for preserving information in operations involving partial values and maybe tuples, the use of marked partial values, and the properties of extended relational operations with regard to the preservation of information are described in [8].

11. Implementation

We have implemented the described concepts in connection with work on the KSYS project [17] at Stanford University. This system, DomainMatch, uses domain mappings, virtual attributes, and a full set of extended relational operations over unmarked partial values and maybe tuples to perform relational query operations over data obtained from mismatched domains. DomainMatch is written in Common Lisp. It is currently running as a front end to the VAX Rdb/VMS database system on Naxos, the KSYS project VAX 11/750.

12. Conclusion

We have presented a solution to the problem of performing operations over mismatched domains. Operations over mismatched domains can be performed through first resolving the domain differences by mapping the conflicting attributes to a common domain and then performing the desired operations over identical domains. The application of a domain mapping to a real attribute results in a virtual attribute. The virtual attribute values may be total or partial.

We have formalized operations across the partial values that result from virtual attribute mappings by defining an extended relational algebra. We have described an extension of the relational algebra that models operations across relations containing both partial values and maybe tuples and have presented definitions of a number of the extended operations of this algebra. As we have demonstrated in examples, our operations allow us to extract more information from our data than would be available by a straightforward application of the standard relational operators.

This approach is particularly suitable for use in a distributed environment where individual databases are autonomous and all updating control resides with local databases, since it can be used with no modification of local schemas.

13. Acknowledgments

14. References

[1] ANSI/X3/SPARC Study Group on Data Base Management Systems, *Interim Report 75-02-08, SIGMOD FDT Bulletin*, Vol. 7, No. 2, 1975.

[2] Joachim Biskup, "A Foundation of Codd's Relational Maybe-Operations," *ACM Transactions on Database Systems*, Vol. 8, No. 4, December 1983, pp. 608-636.

[3] Yuri Breitbart, Peter L. Olson, Glenn R. Thompson, "Database Integration in a Distributed Heterogeneous Database System," *Proceedings of the International Conference on Data Engineering*, 1986, pp. 301-310.

[4] Stefano Ceri and Giuseppe Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, 1984.

[5] E. F. Codd, "Extending the Database Relational Model to Capture More Meaning." *ACM Transactions on Database Systems*, Vol. 4, No. 4, 1979, pp. 397-434.

[6] Bogdan Czejdo, David W. Embley, Marek Rusinkiewicz, "An Approach to Schema Integration and Query Formulation in Federated Database Systems," *Proceedings of the Third International Conference on Data Engineering*, 1987, pp. 477-484.

[7] Umeshwar Dayal and Hai-Yann Hwang, "View Definition and Generalization for Database Integration in a Multidatabase System," *IEEE Transactions on Software Engineering*, Vol 10, No. 6, Nov. 1984, pp. 628-645.

[8] Linda G. DeMichiel, *Performing Database Operations over Mismatched Domains*, Ph.D. Dissertation, Stanford University, 1989.

[9] John Grant, "Incomplete Information in a Relational Database," *Fundamenta Informaticae*, Vol.3, No. 3, 1980, pp. 363-378.

[10] Arthur M. Keller and Marianne Winslett Wilkins, "On the Use of an Extended Relational Model to Handle Changing Incomplete Information," *Transactions on Software Engineering*, Vol. 11, No.7, July 1985.

[11] Witold Lipski, Jr., "On Semantic Issues Connected with Incomplete Information Databases," *ACM Transactions on Database Systems*, Vol. 4, No. 3, Sept. 1979, 262-296.

[12] Witold Litwin, "MALPHA: A Relational Multidatabase Manipulation Language," *Proceedings of the International Conference on Data Engineering*, 1984, pp.86–93.

[13] Witold Litwin and Philippe Vigier, "Dynamic Attributes in the Multidatabase System MRDSM," *Proceedings of the International Conference on Data Engineering*, 1986, pp. 103–110.

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

[15] Marjorie Templeton, David Brill, Son K. Dao, Eric Lund, Patricia Ward, Arbee L. P. Chen, and Robert MacGregor, "Mermaid—A Front End to Distributed Heterogeneous Databases," *Proceedings of the IEEE*, Vol. 75, No. 5, May 1987, pp. 695–708.

[16] Jeffrey D. Ullman, *Principles of Database Systems*, 2nd ed., Computer Science Press, Rockville, Maryland, 1982.

[17] Gio C. M. Wiederhold, Michael G. Walker, Waqar Hasan, Surajit Chaudhuri, Arun Swami, Sang K. Cha, Xiaolei Qian, Marianne Winslett, Linda DeMichiel, and Peter K. Rathmann, "KSYS: An Architecture for Integrating Databases and Knowledge Bases," in Amar Gupta and Stuart Madnick, eds., *Technical Opinions Regarding Knowledge Based Integrated Information Systems Engineering*, MIT, 1987.

Tuning the Reactivity of Database Monitors

Tore Risch *

February 20, 1990

Abstract

Database Monitors allow application programs to asynchronously monitor result changes of database access queries by associating *tracking procedures* with the queries. Whenever some application process U commits updates that the DBMS suspects will change significantly the result of a query monitored by a monitoring process M, the DBMS will invoke the associated tracking procedure of M. The invocation is asynchronous so that the updating process U is autonomous from the monitoring process M.

This paper first gives a formalization of the concept of database monitors. Then we define the *Reactivity* as a measure of how often a given Tracking Procedure will be invoked. Some *Tuning Parameters* are introduced that give the programmer a means to adjust the reactivity. The settings of these parameters adapt the behavior and the performance of database monitors to the needs of particular applications. High reactivity will allow fine grain tracking but it will also decrease the performance of the application, the DBMS, and the communication network. By lowering the reactivity we gain efficiency at the expense of loosing information. The use of tuning parameters is exemplified for two implemented prototype applications.

Key Words and Phrases:

Active Databases, Data Monitoring, Triggers, Real Time Databases, Knowledge Based Systems Support, Object-Oriented Databases.

---

1 Introduction

We have proposed and implemented a *Database Monitor* feature [18] within the Object-Oriented DBMS Iris [10]. The main ideas behind database monitors are the following:

Assume that we have a multi-user DBMS, such as the Object-Oriented DBMS Iris [10], and we also have query language, such as Iris' OSQL[2], with operators to declaratively query the database. Given that the database content is continuously and concurrently updated using atomic transactions, our system allows application processes to be informed when these updates cause the results of given database access queries to change. In particular the technique can support knowledge based modules, or *mediators* [26], where inference methods are triggered to adapt to the observed changes.

We assume that two transactions cannot finish at exactly the same time. At any point in time t_j the database has a global state $S(t_j)$. We denote by U_j the transaction that finishes at time t_j. The database is in state $S(t_j)$ when U_j completes.

Derived data can be specified as views by declarative access queries. An access query Q can be regarded as a mapping from the global database state $S(t_j)$ into the derived result domain of the query, $Q(S(t_j))$. Updates of the global database state $S(t_j)$ over time will also effect $Q(S(t_j))$. We are interested in observing result changes to $Q(S(t_j))$ for each given access query Q, i.e. situations where
$Q(S(t_i)) \neq Q(S(t_j)), i < j$.
A process that is interested in such changes of a query result is said to be *monitoring* the query. For example a process may be continuously monitoring a query deriving the highest paid employee of a company.

We denote with $Q_{(P)}$ a parameterized query where P is a set of actual parameters. Parameterized queries are called *derived functions* in Iris. They are access path optimized for arbitrary parameters similar to canned queries in relational DBMSs [7]. In Iris, object attributes are modeled by derived functions where the first parameter typically is bound to the object identifier. For example, we may define a derived function that retrieves the highest paid employee for any given department. We denote the result of $Q_{(P)}$ at time t_j as $Q_{(P)}(S(t_j))$. Q can be regarded as a special case of $Q_{(P)}$ with no parameters, i.e. $Q(S(t_j)) \equiv Q_{()}(S(t_j))$. Thus we can generalize by monitoring a parameterized query for a given set of parameters, i.e. situations where
$Q_{(P)}(S(t_i)) \neq Q_{(P)}(S(t_j)), i < j$.

We need some mechanism to inform the monitoring process that a result change has occurred for each if its monitored queries whenever $Q_{(P)}(S(t_i)) \neq Q_{(P)}(S(t_j))$. Therefore, the programmer does not only specify $Q_{(P)}$, but also a *tracking procedure* or *tracker*, denoted by τ. A tracker is an application program procedure that is called by the DBMS when a result change has been detected. Thus the tracker is part of the application program and is written in programming language of the application program (e.g. C). The DBMS does *not* send any data to the tracker when a result change has occurred. However, the tracker can freely access the database to retrieve the current result of the monitored query. Thus, we separate change detection (invoking the tracker) from retrieving the new result.

Committed database updates by one process U will cause a tracker of another process M to be invoked asynchronously if the updates of U changed the result of a query monitored by M. Monitoring processes are autonomous from the updating processes, so that committing transactions need not wait for tracking procedures to finish. In this paper we only deal with changes in $Q_{(P)}(S(t_i))$ as seen by monitoring processes other than those causing the result change to occur. Thus we only look at committed data.

A database monitor can be compiled once for a given query [18]. By *activating* a monitor in an application process we inform the DBMS that it shall from now on invoke the tracker whenever significant result changes are detected for a given query and parameters, $Q_{(P)}$. A monitor is *deactivated* either explicitly or when the process is terminated.

Formally, a monitor *activation* A_M by a particular monitoring process M is denoted by a tuple

$$A_M =< Q_{(P)}, \tau, \Delta v, \Delta t, Si, Nc >$$

where $Q_{(P)}$ is a parameterized database query (canned query) monitored for change, P are the actual parameters, and τ is the tracker; $\Delta v, \Delta t, Si, Nc$ are called *Tuning Parameters*.

If the tracker is invoked *exactly every time* it holds that $Q_{(P)}(S(t_{i-1})) \neq Q_{(P)}(S(t_i))$, we say that we have *perfect tracking*. In practice it is often not possible to have perfect tracking. We have identified four important Tuning Parameters with which the programmer can control how often the tracker is to be invoked:
Δv (*Change Significance*),
Δt (*Tracking Delay Time*),
Si (*Synchronous Initiation*),
and Nc (*Nervousness Class*).

In section two we describe the semantics of the four tuning parameters. Section three gives examples of how to use the tuning parameters in two fully implemented prototype applications. In section four we discuss relationships to other distributed modeling concepts such as the *identity connection* introduced in [25] and to database triggers.

2 Tuning Parameters

With the *reactivity* of a monitor activation, we mean the frequency with which its tracker is invoked, defined as $R(A_M)$. The tuning parameters we introduce will influence the reactivity.

The reactivity of an activation depends on five factors:

1. It depends on the structure of the monitored query.

2. It depends on the update frequency for the data over which the monitored query is defined.

3. It depends on the execution time of the tracker. The reactivity will be low if a tracker takes a long time to execute, because the application need to have time to process the notifications.

4. It depends on the overhead for the DBMS to detect change and to transmit the notification to the client.

5. It depends on the tuning parameters below. They allow the programmer to adjust the reactivity.

We say that we have an *underreacting* activation if its tracker is invoked more seldom than with perfect tracking. Thus an underreacting tracker τ is sometimes not invoked at time t_j even though it was previously invoked at time t_i and $Q_{(P)}(S(t_j)) \neq Q_{(P)}(S(t_i))$.

If the data retrieved by $Q_{(P)}$ is intensively updated, the reactivity can become very high, resulting in high overhead and network traffic. In the worst case the monitoring application will spend all its time invoking tracking procedures. For such applications it is desirable to have underreacting monitors.

The asynchronous execution model for trackers makes us have underreacting activations if changes are detected more often than the execution time of the tracker. If one or several changes happen while the tracker is running, the system cannot immediately invoke the tracker for every change. The method used is that the system delays execution of pending trackers until immediately after the first tracker has finished. Therefore, if it takes at least $E(A_M)$ time units to execute the tracker of A_M then

$$R(A_M) \leq \frac{1}{E(A_M)}$$

We now continue by discussing tuning parameters and their influences on reactivity.

2.1   Change Significance

One way to decrease the reactivity of an activation A_M is to make the tracker not react to small changes in monitored data, but only to significant changes. The tuning parameter Δv controls how significant a change to the result of a monitored query need be before the tracker is invoked. If the difference between the old and the new result is within an interval specified by Δv, then there is no significant change.

Δv is relevant only for queries returning numerical results. For queries returning sets of results the significance test is applied to each numeric element in the set, and the tracker will be invoked whenever any result in the set is significantly different from the old result.

The interval can either be specified as a *relative difference*, denoted Δv^R, or as an *absolute difference*, denoted Δv^A.

We denote the previous time the tracker was invoked by t_i. Assuming that $Q_{(P)}$ returns a numeric value, if an absolute difference Δv^A is specified, we invoke the tracker at time t_j if it holds that:

$$|Q_{(P)}(S(t_j)) - Q_{(P)}(S(t_i))| \geq \Delta v^A.$$

By contrast, if a relative difference Δv^R is specified, we invoke the tracker when it holds that:

$$\frac{|Q_{(P)}(S(t_j)) - Q_{(P)}(S(t_i))|}{|Q_{(P)}(S(t_i))|} \geq \Delta v^R$$

Different activations can have different Δv specified.

It is advantageous if the active DBMS can do this kind of *dynamic filtering* before notifying the application in order to decrease the frequency of notification for intensively updated data. The application can dynamically change the difference interval to decrease the reactivity if it is performing some critical task. Dynamic filtering is required by, for example, real time monitoring AI systems where the tracker initiates time consuming reasoning activities [22].

We have initially chosen only to provide the two difference comparison tests above. Other comparisons are also possible, e.g., *fixed thresholding* or *moving averages* [22]. As will be shown in an example, fixed thresholding can be specified by inequality comparisons in the monitored query.

It should be noted that a simple scheme, where the comparison function is well understood, can be more easily optimized than complicated comparison schemes. If the system understands the comparison function, it can do special optimizations for that particular comparison function. Complicated comparison schemes will pose a significant overhead on the system.

2.2 Tracking Delay Time

Another way to decrease the reactivity is to specify that the invocation of a tracker should not be initiated more frequently than once per a given time interval.

For example, if we are tracking the failure rates of produced parts, we may only want to display the failure rate each 10 minutes.

The tuning parameter Δt, the *Tracking Delay Time*, specifies a time interval between two tracker invocations.

With $\Delta t > 0$, the tracker will be invoked if $Q_{(P)}$ has changed *and* at least Δt time units have passed since the last time it was invoked. Thus if the tracker was previously invoked at time t_i it will be invoked at time t_j if

$$Q_{(P)}(S(t_j)) \neq Q_{(P)}(S(t_i)), t_j - t_i = \Delta t.$$

The time when the tracker is invoked is also delayed by the overhead time for the DBMS to detect the change, $Oh(A_M)$. We denote the commit time of the transaction causing the change with t_U, the time where the tracker gets invoked with t_j. It holds that

$$t_U + Oh(A_M) \leq t_j$$

Specifying tracking delay time corresponds to *sampling* changes in the result of $Q_{(P)}$. If $Q_{(P)}$ does not change state too irregularly this is an effective method.

Tracking delay time may be combined with change significance, meaning that we are sampling significant change to monitored data at given time intervals.

2.3 Synchronous Initiation

The tuning parameter Si specifies *synchronous initiation* of the tracker. A tracker is synchronously initiated if its execution is started before the change causing transaction commits. However, we do not wait for the tracker to finish before committing. Thus synchronous initiation makes the updating transaction *semi-autonomous* from the monitoring process. The monitoring process can test if a given tracker has started. The system does synchronous initiation when $Si = True$.

Normally trackers are initiated asynchronously ($Si = False$), i.e., they are invoked as soon as possible after the change causing transaction is committed. For asynchronously initiated trackers, if the transaction causing the change committed at time t_U, and the system overhead to detect the change and to invoke the tracker of the activation A_M is $Oh(A_M)$, then the tracker will be invoked at time

$$t_U + Oh(A_M)$$

Notice that synchronous initiation can be very expensive when monitoring data that is intensively updated. This is because *every* updating transaction need to determine if change in monitored data has occurred, as well as waiting until it receives confirmation that the tracker has been initiated. Thus synchronous initiation delays commits of updating transactions with time $Oh(A_M)$. Synchronous initiation will decrease the reactivity at the expense of autonomy between the updating and monitoring processes. Synchronous initiation should be avoided unless $Oh(A_M)$ can be brought down to a minimum.

Settings of $\Delta t > 0$ are meaningful only when $Si = False$.

2.4 Nervous Monitors

If the trackers are invoked too often, i.e., it is invoked at time t_i and t_j even though $Q_{(P)}(S(t_j)) = Q_{(P)}(S(t_i))$ and no change occurred between t_i and t_j, we say that we have an *overreacting* or *nervous* monitor. For performance reasons, it is sometimes very advantageous to have nervous monitors. The reason is that the test to detect change of a nervous monitor can be designed to have little overhead $Oh(A_M)$. For nervous monitors we only need a simple test that succeeds if the system *suspects* that an update has caused the result of a monitored query to change. We call such a simple test for suspected changes in $Q_{(P)}(S(t_i))$, a *screening test*. The screening test normally analyzes the write set of a transaction, which we denote $Ws(U)$.

These are some examples of screening tests:

1. A trivial test would be to check if the database has been updated by a transaction (i.e. $Ws(U) \neq Null$) then any monitored query eventually has changed. That is an extremely cheap test, but in most cases it would generate too many notifications. It's negation ($Ws(U) = Null$) is a cheap test we use to screen out all read-only transactions.

2. A better test would be that if a certain relation R is updated (i.e. $R \in Ws(U)$) then all nervous monitors referencing R should be notified.

3. The latter test can be refined to test if the value of the column $R.C$ of a relation has been updated (i.e. $R.C \in Ws(U)$). Ifso, all monitored queries referencing that column may have new values. This corresponds to testing if an Iris *function* has been updated.

4. If a monitored query retrieves values from a specific row of a relation only, its result states can change only if that specific row is updated.

When the tuning parameter Nc, the *Nervousness Class*, is set to a non-zero value, it specifies that the system shall use screening tests for change detection, thus activating nervous monitors. The non-zero value of Nc indicates what kind of screening test to use;

the higher value the less careful screening and thus the higher reactivity. We have currently implemented test 3 only.

If we have an application that requires synchronous initiation it can be necessary to use nervous monitors to avoid doing expensive tests inside many updating transactions.

The system uses screening tests also for non-nervous monitors as a quick way to screen out irrelevant updates before doing the full change detection test.

Nervous tracking can be combined with time delays, meaning that the DBMS sometimes invokes the tracker after a time interval Δt even if no result change has happened since the previous notification.

3 Examples

In this section we describe two implemented applications using database monitors. We discuss how the concept is applied to each of them including settings of tuning parameters.

3.1 Tracking Failure Rates

Figure 1 illustrates a manufacturing database used to keep track of products and parts. Figure 2 shows how the schema is defined in Iris. Each product is represented by the entity Product, and each part by the entity Part. The relationship Produces defines which products are manufactured by which department, and ProductOf defines the products using a given part. (Iris supports set valued functions).

We make a simplifying assumption that the same organization both produces and maintains the products. Periodically, when parts are failing, field engineers update the database reporting the current failure rate for failing parts. Such a change in the failure rate for some part will cause the entity Failure to be updated. In addition, the relationships PartFailure and ProductFailure are updated to indicate which part in which product was failing.

When a part is failing there is a service department that is responsible for servicing the

Figure 1: A Manufacturing Database

part, indicated by the relationship ResponsibleDepartment.

We have implemented two applications of database monitors for the manufacturing database:

- A service department wants to monitor when the failure rate of some part for which the department is responsible grows larger than a specified threshold.

- If the failure rate for some part grows very significant, we also notify the producer of its product.

We call these kind of applications *Failure Rate Monitors*. The current implementation is made in directly in C with an X window-based user interface. Future implementation will use knowledge based mediators [26] containing rules that specify adaptations to be made when the observed failure rates change significantly.

Figure 3 shows the OSQL queries to be monitored by a given service department and producing department, respectively. The user enters his/her department name. For service departments the query ServiceReport uses the relationship ResponsibleDepartment

```
/* Entities as Iris types */
create type Department(
        Number                  Charstring,
        Name                    Charstring);
create type Product(
        Number                  Integer,
        Name                    Charstring);
create type Failure(
        FailureRate             Real);
create type Part(
        Number                  Charstring,
        Name                    Charstring);

/* Relationships */
create function ResponsibleDepartment(Part) -> Department;
create function Produces(Department) -> Product;
create function ProductOf(Part) -> Product;
create function ProductFailure(Failure) -> Product;
create function PartFailure(Part) -> Failure;
```

Figure 2: Iris Schema for Parts Database

```
create function ServiceReport(Department d, Integer th) ->
    <Charstring prn, Charstring pan, Integer ta> as
      select prn,pan,ta
          for each Failure f, Part pa, Product pr,
                  Charstring prn, Charstring pan, Integer ta where
          ResponsibleDepartment(pa) = d and
          PartFailure(pa) = f and
          Rate(f) = ta and
          Product(f) = pr and
          prn = Name(pr) and
          pan = Name(pa) and
          th < ta
;
create function ProducerReport(Department d, Integer th) ->
    <Charstring prn, Charstring pan, Integer ta> as
      select prn,pan,ta
          for each Failure f, Part pa, Product pr,
                  Charstring prn, Charstring pan, Integer ta where
          Produces(d) = pr and
          Product(f) = pr and
          PartFailure(pa) = f and
          Rate(f) = ta and
          prn = Name(pr) and
          pan = Name(pa) and
          th < ta
;
```

Figure 3: Monitored Queries

to find the parts serviced by the department. For producing departments the query ProducerReport uses the relationship Produces to find which products produced by the department has failing parts. The user also enters a fixed critical failure rate threshold, so that no part with a failure rate lower than that threshold is reported. Different users use different thresholds, depending on their responsibilities. For example, the producers typically use larger thresholds than service engineers.

With a conventional DBMS the application would have to regularly query the database for every situation that it regards as interesting; with database monitors the DBMS actively notifies the application when interesting situations happen. Tuning parameters are set so that the application is not notified too often and only in response to significant change.

We also developed a conventional database update program to report failure rates, called the *Failure Rate Reporter*. Customers run a program to report failures of parts and store the reports in the database. The failure rate reporter is run at fixed time intervals to calculate the current failure rate as number of reports received for a given part during the time interval. When failure rates grow too high, service engineers are sent out to the failing sites to repair the problems. Hopefully, after the repair, the rates will go down, and not show up as significant failures any more. The implementation of the Failure Rate Reporter is completely independent of the Failure Rate Monitors; it basically counts failure reports regularly and stores the calculated failure rate in the database. Figure 4 illustrates the information flow from two Failure Rate Reporters to Failure Rate Monitors for two service departments and one producing department. Each box represents a process that normally runs on a separate workstation. Failure Rate Monitors run on different machines than the Failure Rate Reporters, and the transmission time between the machines can be significant. We do not want the Failure Rate Reporter to wait for these transmission delays. This is an example of an application where the updating transactions should be autonomous from the monitoring process.

The service department has the following settings of tuning parameters:
$Si = False$ (asynchronous initiation)
$Nc = 0$ (no nervous Tracking)
$\Delta v^R = 0.05$ (report differences larger than 5%)
$\Delta t = 60s$ (notify every minute)

The reactivity of the producing department's Failure Rate Monitor can be set significantly lower:
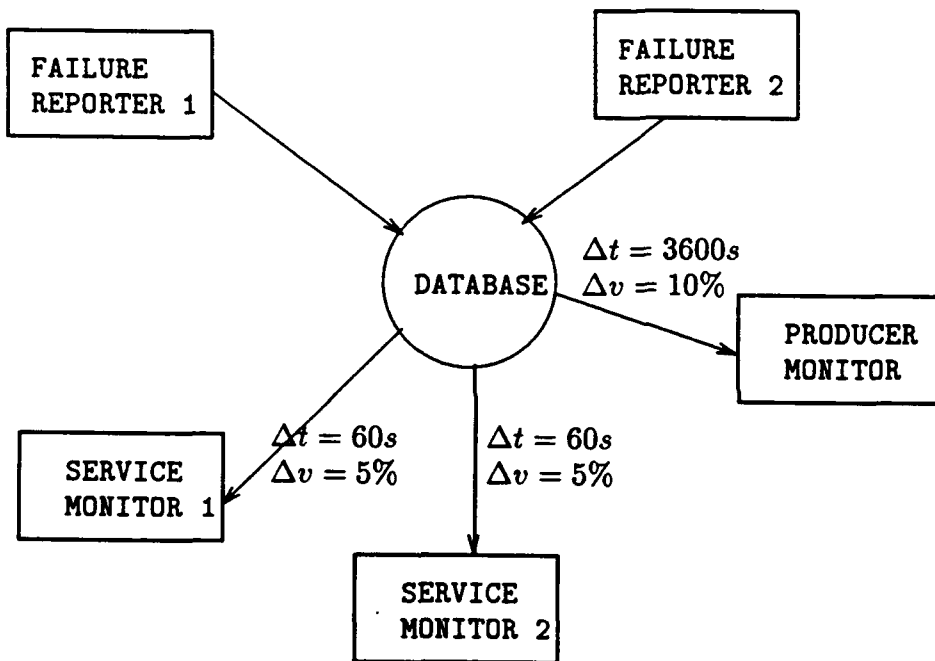$Si = False$ (asynchronous initiation)

Figure 4: Information Flow through Monitors

$Nc = 0$ (no nervous Tracking)
$\Delta v^R = 0.1$ (report differences larger than 10%)
$\Delta t = 3600s$ (notify every hour)

Δt is set high because it is conceivable that the producing department is physically located far from the customer, and the transmission cost can be significant.

3.2   View Object Materialization

Many applications need to manipulate 'objects' stored in persistent databases. Several object-oriented DBMSs [13, 16, 20] are developed for this purpose. An alternative is the *view object* concept [23, 24] where data is stored persistently in a back-end relational DBMS, and objects are materialized in the applications when data is retrieved from the DBMS. Views are used to re-configure data so that only relevant data for the application is materialized in a form adopted to the particular application.

With Object-Oriented DBMSs, the object structure stored in the database is not always optimal for the application. The OODBMS Iris [10] separates the representation of objects from their usage, and allows the programmer to specify *derived functions* to give the right view of objects for a particular application. However, the object structure retrieved by a query may not be optimal for every application, and the view object concept can therefore be applied also to Iris applications.

In particular, we have applied the view object concept to the implementation of database monitors itself. The implementation uses several system tables, some of which are updated very seldom. For example, when a monitor is *defined* (compiled), the system analyzes the monitored query and stores dependency information in system tables. This is done only once for each monitored query. The dependency information is traversed when transactions screen out non-monitored updates. The dependency information is thus updated rarely but accessed intensively and is therefore a very good candidate for materialization as view objects. These view objects should be organized for quick traversal of the screening tests.

In this case, it is critical that the view objects do not reference stale data. Thus, since we use atomic transactions, the view objects need to be flushed at the end of each transaction. This leads to an inefficiency in the implementation, since the dependency table is used by

118

every updating transaction for the screening test. If a session does more than one commit, we would have to re-materialize the dependency table in each of the session's transactions.

We may keep the view objects during the entire session by monitoring the state of the dependency table, and let the tracker invalidate the view-object when the state changes. Such changes would happen only when a monitor is compiled, which would be very seldom. Whenever we need to traverse the dependency table, the system would first check if the view object is invalid, in which case it would be re-materialized. We use synchronous initiation because we can then test in the client whether the cache is invalidated.

For view materialization of the dependency table we used the following settings of tuning parameters:

$Si = True$ (synchronous initiation)
$Nc = 1$ (nervous monitoring)

The nervousness setting indicates that only a screening test is to be used. This is OK because of the expected very low reactivity, and because we know that almost every change to the dependency table will cause significant change.

4 Application to Distributed Models

A concept related to the database monitors is the *identity connection* introduced in [25]. The identity connection defines replicated attributes of relations. For any instance the connected attributes must *eventually* be the same, where *eventually* means that a time limit is specified indicating that the connection is to be updated regularly. An identity connection can also have a derivation formula to transform the connected attribute [24]. One may see view materialization mechanisms [3, 15, 20] as a way to implement such connections.

Database monitors can be seen as a way to specify and implement an identity connection from *persistent data* to an *application process*. With database monitors we specify the connection from a data attribute specified by a query $Q_{(P)}$ to a process M by
$$A_M = < Q_{(P)}, \tau, \Delta v, \Delta t, Si, Nc >.$$

The following relationships hold between identity connections and monitor activations:

- The tracker τ implements an action to be done when the identity is violated. We thus do not maintain the connection, but rather inform the process when it is violated. One may implement replicated attributes using database monitors by a constantly running process whose tracker just retrieves the monitored data and immediately stores it in the replicated attribute.

- The time delay Δt is analogous to the *time limit* of identity connections. The identity connection also allows the specification of explicit time points, which can be implemented by letting the tracker deactivate the monitor.

  By specifying the time delay we get *non-perfect* connection, in the sense that we do not always guarantee that the connection is fully up to date.

- The change significance Δv is an alternative to time delays for specifying non-perfect connections, relying on data differences rather than time differences.

- The synchronous initiation flag Si allows the implementation of synchronous connections between data and process.

- The nervousness class Nc is an implementation tuning parameter controlling the efficiency of maintaining the connection.

- Finally, database monitors implement *temporary* connections because they are valid only while the monitor is activated during the execution of the connected process.

In the identity connection model an 'event' can cause a connection to re-establish. This could be implemented by recording the time for the event and then monitoring the recorded data.

This work is also related to database triggers [1] available in several systems, e.g. Sybase [8] and HiPac [6, 9]. HiPac has a feature called the *coupling mode* between a trigger and a set of actions on the database. The coupling mode controls when a trigger action is executed relative to when a triggering condition has occurred. With database monitors, for a given query there are many actions that eventually cause its result state to change; the monitor compiler analyzes the query to generate a plan for how to deal with these actions when determining changes in query results. We are not interested in the individual

database update actions causing the triggering state change, but rather in query result changes caused by completed transactions.

Because triggers are action oriented, rather than value change oriented, they connect their coupling mode to the updating action. Monitors are value oriented, and thus tie the tuning parameters to a monitor activation in a different process than the updating transactions. Different monitor activations may have different tuning parameters.

The *alerters* proposed in [5] can be seen as a database monitor of a boolean expression, where the user is alerted with a message when the expression becomes true.

5 Summary and Conclusions

We formally defined the semantics of *Database Monitors* as proposed in [18], as monitoring value changes in the result of parameterized database queries as seen by separate monitoring processes. We showed that the concept can be applied to both relational and object-oriented DBMSs, with the basic requirement that the DBMS has a non-procedural query language.

We extended the simple model to include *tuning parameters*, that the programmer can use to adjust the invocation frequency or *reactivity* of trackers.

The tuning parameter *Synchronou. Initiation*, allows the synchronization of the initiation of a tracker with the commits of updating transactions. Synchronous initiation can be slow, but necessary for applications where consistency is required.

The tuning parameters *Change Significance* and *Time Delay* make the connection between data and the monitoring process be non-perfect: These settings make database monitors *underreact* so that the tracker is not invoked even though the connection has been invalidated.

The tuning parameter *Nervousness Class* makes the monitor *overreact* so that the trackers gets invoked even though the connection was not invalidated. Nervous monitors are important for efficiency.

As illustrations we gave an elaborate example of how to set tuning parameters for a database tracking the failure rates of products. We also showed how to use database monitors for caching database objects in the real memory of a client process.

We showed that a database monitor can be regarded as a connection from the database to the monitoring process, whose properties are specified by a tuple A_M. An important part of A_M is the *tracker* τ, which is an application-provided procedure that the DBMS invokes when the connection is invalidated. The tracker implements adaptations to be made when change happens.

An interesting area for further research is to generalize the concept into connections between arbitrary 'active' knowledge based modules, or *active* mediators [26]. Each active mediator would have its internal state recorded in a database, a language to access the internal state, and support for monitoring value changes of externally accessed data, similar to *active objects* in MACE [21] and KNO [11]. Promising techniques for programming such active mediators include rule based techniques [4, 12], and constraint based languages [14, 17, 19].

The concept could be extended by providing a more powerful monitor specification language. For example it should be possible to monitor increases and decreases of values over some time period.

Other future work include improvements to the implementation of database monitors, and tools for performance tuning of monitoring applications.

ACKNOWLEDGEMENTS:

References

[1] M.Astrahan et al: System R: A Relational Approach to Database Management, *Transactions on Database Systems*, 1 (2), June 1976.

[2] D.Beech: A Foundation for Evolution from Relational to Object Databases, *Advances in Database Technology - EDBT '88*, Lecture Notes in Computer Science, Springer-Verlag, 1988, pp251-270.

[3] J.A.Blakeley, P.A.Larson, F.W.Tompa: Efficiently Updating Materialized Views, *Proc. SIGMOD*, Washington D.C., 1986, pp61-71.

[4] Brownston,L., Farell,R., Kant,E., Martin,N.: *Programming Expert Systems in OPS5*, Addison-Wesley, Reading, Mass., 1985.

[5] O.P.Buneman, E.K.Clemons: Efficiently Monitoring Relational Databases, *ACM Transactions on Database Systems* 4, 3 (sept. 1979), pp368-382.

[6] D.R.McCarthy, U.Dayal: The Architecture of an Active Database Management System, *Proc. SIGMOD*, Portland, Oregon, 1989, pp.215-224.

[7] D.D.Chamberlin et. al.: Support for Repetitive Transactions and Ad Hoc Queries in System R, *Transactions On Database Systems*, vol. 6, no. 1, March 1981, pp.70-94.

[8] M.Darnovsky, J.Bowman: *Transact-SQL User's Guide*, Sybase, Inc., 2910 Seventh Street, Berkeley, CA 94710, 1988.

[9] U.Dayal, A.P.Buchmann, D.R.McCarthy: *Rules Are Objects Too: A Knowledge Model For An Active, Object-Oriented Database System*, Advances in Object-Oriented Database Systems, 2nd Intl. Workshop on Object-Oriented Database Systems, Sept. 1988, pp129-143.

[10] D.H.Fishman, J.Annevelink, E.Chow, T.Connors, J.W.Davis, W.Hasan, C.G.Hoch, W.Kent, S.Leichner, P.Lyngbaek, B.Mahbod, M.A.Neimat, T.Risch, M.C.Shan, W.K.Wilkinson: *Overview of the Iris DBMS*, in W.Kim, F.H.Lochovsky: *Object-Oriented Concepts, Databases, and Applications*, ACM Press, 1989.

[11] L.Gasser, C.Braganza, N.Herman: Implementing Distributed AI Systems Using MACE, in A.H.Bond, L.Gasser: *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988, pp.445-450

[12] Hanson,E.N.: An Initial Report on the Design of Ariel, in *SIGMOD RECORD: Special Issue on Rule Management and Processing in Expert Database Systems*, Vol.18, No.3, Sept.1989.

[13] W.Kim, N.Chou, J.Garza: Integrating an Object-Oriented Programming System with a Database System, *Proc. OOPSLA*, sept 1988, pp.142-152.

[14] Wm Leler: *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley Publishing Company, 1988

[15] B.Lindsay, L.Haas, C.Mohan, H.Pirahesh, P.Wilms: A Snapshot Differential Refresh Algorithm, *Proc. SIGMOD*, Washington D.C., 1986, pp53-60.

[16] D.Maier, J.Stein: Development of an Object-Oriented DBMS, *proc. OOPSLA*, Sept 1986, pp.472-482.

[17] M.Morgenstern: Active Databases as a Paradigm for Enhanced Computing Environments, *9th VLDB Conf.*, Florence, 1983, pp34-42.

[18] T.Risch: Monitoring Database Objects, *Proc. 15th Conf. on Very Large Databases* (Amsterdam, Holland), 1989, 445-453.

[19] G.L.Steele Jr., G.J.Sussman: CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions, *Artificial Intelligence* 14, 1980, pp.1-39.

[20] M.Stonebraker: The Design of POSTGRES, *Proc. ACM SIGMOD Conf.*, Washington, D.C., 1986, pp340-355.

[21] D.Tsichritzis, E.Fiume, S.Gibbs, O.Nierstrasz: KNOs: KNowledge Acquisition, Dissemination, and Manipulation Objects, *ACM Transactions on Office Information Systems*, vol 5, no 4, pp.96-112, 1987.

[22] R.Washington, B.Hayes-Roth: Input Data Management in Real-Time AI Systems, *11th Intl. Joint Conf. on Artificial Intelligence*, 1989, pp.250-255.

[23] G.Wiederhold: Views, objects, and databases, *IEEE Computer* 19(12) 1986, pp.37-44.

[24] G.Wiederhold: Connections, in G.Wiederhold, T.Barsalou, S.Chaudhury: *Managing Objects in a Relational Framework*, Stanford Computer Science Report No.STAN-CS-89-1245, 1989.

[25] Wiederhold, G., Qian, X.: "Modeling Asynchrony in Distributed Databases" / 3rd Intl. Conf. on Data Engineering, Los Angeles, CA, Feb. 3-5, 1987, pp.246-250.

[26] G.Wiederhold: *The Architecture of Future Information Systems*. In this research report.

Management of Complex Structural Engineering Objects in a Relational Framework*

Kincho H. Law; Thierry Barsalou; and Gio Wiederhold§

February 12, 1990

ABSTRACT

To structure the development of an integrated building design environment, the global representation of the design data may best be organized in terms of hierarchies of objects. In structural engineering design, we deal with large sets of independent but interrelated objects. These objects are specified by data. For an engineering design database, the system must be able not only to manage effectively the design data, but also to model the objects composing the design. The database management system therefore needs to have some knowledge of the intended use of the data, and must provide an abstraction mechanism to represent and manipulate objects. Much recent research in engineering databases focuses on object management for specific tasks but gives little attention to the sharability of the underlying information. This paper describes an architecture for the management of complex engineering objects in a sharable, relational framework. Potential application of this approach to object management for structural engineering analysis and design is discussed.

1 Introduction

In building design, we deal with large sets of independent but interrelated objects. These objects are specified by data. The data items describe the physical components (for example, columns, beams, slabs) and the topological aspects of the design (for example, member and joint connectivities). The design data need to be stored, retrieved, manipulated, and updated, during all phases of analysis, design, and construction of the project. An efficient data management system becomes an indispensable tool for an effective integrated computer aided analysis and design system.

Using a database to store and describe engineering data offers many benefits [29]. Some of these benefits include:

- Ability to store and access data independent of its use, so that the data can be shared among the participants

- Ability to represent relationships among the data, so that dependencies are documented

- Control of data redundancy, so that consistency is enhanced

- Management of data consistency and integrity, so that multiple users can access information simultaneously

- Enhanced development of application software by separating data management function

- Support of file manipulation and report generation for *ad hoc* inquiry

To maximize these benefits, a database management system needs to have some knowledge of the intended use of the data. That is, the formal structure or model used for organizing the data must be capable of depicting the relationships among the data and must facilitate the maintenance of these relationships. Furthermore, the structure should be sufficiently flexible to allow a variety of design sequences and to aid an engineer to understand the design.

Traditional relational database systems provide many interesting features for managing data; among them are the capabilities of set-oriented access, query optimization and declarative languages. More important, from the user point of view, the relational model is completely independent of how data are physically organized. The relational model presents data items as records (tuples) which are organized in 2-dimensional tables (relations), and provides manipulation languages (relational calculus and algebra) to combine and reorganize the tables or relations for processing. The relational approach is simple and effective, particularly for business data processing. However, a "semantic" gap exists between the relational data model and engineering design applications. The relationships among the data items describing an engineering design are often complex. The lack of a layered abstraction mechanism in the relational model makes it inadequate for defining the semantics of applications and for maintaining the interdependencies of related data items. Furthermore, the traditional

set-oriented relational structure does not support well the engineering views of the data. The engineering users have to supply all the intentional semantics in order to exploit the data.

Object-orientation is an active focus of engineering database research. Object-oriented data models have been proposed to increase the modeling capability, to provide richer expressive concepts and to incorporate some semantics about engineering data. The main objective here is to reduce the semantic gap between complex engineering design process and the data storage supporting the process. In such a process, an engineer often approaches the design in terms of the components (objects) that comprise the design, and the operations (methods) that manipulate the components. A database system that supports the object-oriented nature of the design process can certainly enhance the interactions between the engineers and the system.

It should be noted that an object-oriented data model does not necessarily imply that the object-oriented paradigm need to be explicitly implemented inside the database system. In engineering modeling and design, the information that an object represents is often shared by various applications having different views of the data. Data sharing is therefore as important as object-oriented access. Storing objects (explicitly) in object format is not desirable, particularly if the objects are to be shared [31]. We therefore propose an approach, based on the structural data model, that permits object-oriented access to information stored in a relational database; information which in turn can be shared among different applications [2, 5, 6, 24]. In this paper, we discuss the application of this model for the management of complex structural engineering objects in a relational framework.

This paper is organized as follows: The needs of a structural engineering database system are discussed in Section 2. The structural data model is briefly reviewed in Section 3. In Section 4, we present the principles and motivation of an object-oriented system (PENGUIN), its architecture, and some structural engineering examples. In Section 5, we discuss the use of view-objects for modeling design abstractions. In Section 6, we conclude this paper with a summary of the expected benefits of our approach for engineering design applications.

2   Abstractions in Structural Engineering

Practical engineering tasks have too many relevant facets to be intellectually represented through a single abstraction process. Manageability of an application can be achieved by decomposing the model into several hierarchies of abstractions. In general, an aspect of a building and its design can be described as a collection of objects or concepts organized hierarchically [12, 14, 15, 18, 22, 25, 27]. The description of a design project grows as it evolves. During the design, additional attributes may be added to the description of existing entitites; similarly, aggregated entities can be decomposed into their constituents. That is, during design, information is added to the hierarchy by refinement in a top-down manner or by aggregation in a bottom-up sequence. The concept of abstraction provides a means for defining complex structures as well as the semantic information about the data. Powell and Bhateja have defined

some requirements for an abstraction model in structural engineering application [25]
:

- The model must support several applications.

- The model should be in terms of well-defined entities, relationships and dependencies.

- The model must support the creation of abstractions for real structures; that is, it must allow all relevant features of a structure to be represented. In addition, the concepts used in the model should be familiar to the users.

- The model should allow the level of details to be increased as the design of the structure is progressively refined.

- The model should be able to represent structures of various types.

A structural engineering database system must be capable of supporting such an abstraction model.

Choosing a good data model to represent design data and processes is a major step towards the development of an integrated structural design system. A data model is a collection of well-defined concepts that help the database designer to consider and express the static properties (such as objects, attributes and relationships) and the dynamic properties (such as operations and their relationships) of data intensive applications [10]. In addition to enhancing the database design process, a data model must also provide the integrity rules to ensure consistency among the entities.

A relationship is a logical binding between entities. There are three basic types of relationships that are commonly used: *association*, *aggregation* and *generalization*. *Association* relates two or more independent objects as a merged object, whose function is to represent the many-to-many relationships among the independent objects. Association can be used to describe multiple "member of" relationships between member objects and a merged object. *Aggregation* combines lower level objects into a higher level composite object. In general, aggregation is useful for modeling part-component hierarchies and representing "part of" relationships. *Generalization* relates a class of individual objects of similar types to a higher level generic object. The constituent objects are considered specializations of the generic object. Generalization is useful for modeling alternatives and representing "is a" relationships. These three basic relationship types, in particular aggregation and generalization, are supported by many semantic data models and have been widely used in computer aided building design research [8, 14, 15, 19, 21, 22]. A joint can be represented as an association of several structural elements (such as beams and columns) and connecting plates, and carries some information about the connectors to be used. A staircase is an aggregation of many similar parts. A concept "beams" is a generalization of a variety of members supporting gravity loads.

These three relationships impose certain existential dependency among the object entities. For example, the joint information is only meaningful while the referencing beams and plates are part of the design. As another example, assuming that the

entities "BEAM" and "COLUMN" are specializations of a generic entity "STRUC-TURAL ELEMENT", existence of a "BEAM" or "COLUMN" instance requires that a correponding instance also exists in the generic entity "STRUCTURAL ELEMENT". When an instance is deleted from the generic entity "STRUCTURAL ELEMENT", corrective measure should be taken to remove the corresponding instance in a specialized entity "BEAM" or "COLUMN"; as a result, consistency between the generic and the specialized entities can be maintained in the database. Dependency constraints of these three types of relationships have been examined in details [3, 9, 11, 23, 26].

Besides association, aggregation and generalization, other relationships, such as "connected to", "supported by" (or "supporting"), "influence" and "determinants", have received considerable interests in building design applications [1, 13, 18, 25]. While incorporating these types of relationship into a data model is desirable, dependencies among the entities participating in these relationships have not yet been formally defined. For instance, when a supporting element is removed from a structure, corrective measures should be imposed on the objects that it supports. On the other hand, removing a supported element, from the data management point of view, should have little effect on the corresponding supporting object. It is important for a database management system to ensure consistency among the building design data, and to reflect appropriately the semantic relationships among them.

To be general, a database must support all design activities, in addition to capturing the semantic knowledge of the data. For each activity, an engineer works with specific application abstraction of the building, rather than with a complete physical description. For example, in the analysis of a building structure, an engineer is primarily interested in the building frame in terms of the center line of the members, their physical properties and the stiffness of the connections. Other information, such as room spaces and wall partitions, can be ignored. The database system needs to support various abstract views of the information pertaining to a specific domain. The ability to support multiple views for satisfying the requirements of different applications is also an important aspect of an engineering data model.

3 The Structural Data Model

The goal in selecting a semantic data model is to represent directly in an easily manipulable form as many of the objects and their relationships of interest as possible. The structural data model that we use in this study is an extension of the relational model [16, 30]. Relations are used to capture the data about objects and their parts. The structural data model augments the relational model by capturing the knowledge about the constraints and dependencies among the relations in the database. This section reviews briefly the structural data model. For more detailed description of this data model, the reader is referred to References [3, 5, 16, 30].

The primitives of the structural data model are the *relations* and the *connections* formalizing relationships among the relations. The connection between two relations R_1 and R_2 is defined over a subset of their attributes X_1 and X_2 with common domains. Two tuples, $t_1 \in R_1$ and $t_2 \in R_2$, are connected if and only if the connecting

attributes in t_1 and t_2 match. There are three basic types of connections, namely *ownership*, *reference*, and *subset* connections. These connections are used to define the relationships between the relations and to specify the dependency constraints between them.
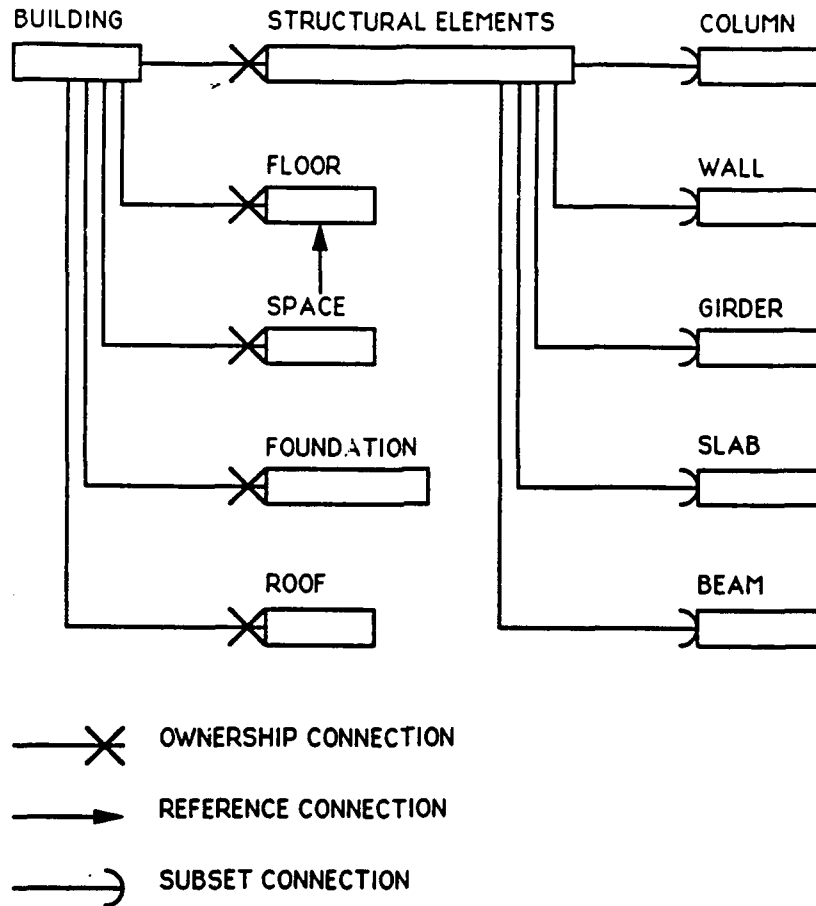

Figure 1: Composition of a Building Structure using the Structural Model

An *ownership* connection between an owner relation R_1 and an owned relation R_2 is useful for representing "part-component" or aggregation type of relationship. As an example, Figure 1 shows the composition of a building structure consisting of a few simple entities. The basic components of a building structure include the descriptions of structural elements, floor, foundation, roof, space, etc.. The components exist if and only if the building exists. This owner-component relationship is best represented using the ownership connection. This connection type specifies the following constraints:

1. Every tuple in R_2 must be connected to an owning tuple in R_1.

2. Deletion of an owning tuple in R_1 requires deletion of all tuples connected to that tuple in R_2.

The ownership connection describes the dependency of multiple owned tuples on a single owner tuple.

A *reference* connection between a primary (referencing) relation R_1 and a foreign (referenced) relation R_2 is useful for representing the notion of abstraction. Referring to the example shown in Figure 1, an architectural space, which may be an office, elevator opening, mechanical room etc., locates on (or references) a floor. The floor cannot be removed without first removing the spaces defined on that floor. This connection type specifies the following constraints:

1. Every tuple in R_1 must either be connected to a referenced tuple in R_2 or have null values for its attributes X_1.

2. Deletion of a tuple in R_2 requires either deletion of its referencing tuples in R_1, assignment of null values to attributes X_1 of all the referencing tuples in R_1, or assignment of new valid values to attributes X_1 of all referencing tuples corresponding to an existing tuple in R_2.

The reference connection describes the dependency of multiple primary tuples on the same foreign tuple. The reference connection can be used to refer to concepts which further describe a set of related entities. It should be noted that association can be modeled with a combination of ownership and reference connections [32]. As an example, in a steel frame structure, a joint connection is associated with the structural elements through one or more connectors (see Figure 3).

A subset connection between a general relation R_1 and a subset relation R_2 is useful for representing alternatives or "is a" type relationship. Generalization (and its inverse, specialization) can be modeled using the subset connection. For the example shown in Figure 1, a structural element can either be a column, a beam, a wall, or a slab. Furthermore, a beam can be generalized to be either a main girder or a joist (secondary beam). Deleting a specific instance in the generic class of structural element must delete the corresponding instance existing in the subclass. The subset connection specifies the following constraints:

1. Every tuple in R_2 must be connected to one tuple in R_1.

2. Deletion of a tuple in R_1 requires deletion of the connected tuple in R_2.

The subset connection links general classes to their subclasses and describes the dependency of a single tuple in a subset on a single general tuple.

Besides supporting the three basic relationships of aggregation, generalization and association, the connections can also be used to define relationships such as "connected-to" and "supported-by", that are useful in engineering application. In the "supported-by" or "supporting" relationship, the supporting entity should not be removed unless all its supported entities no longer exist. For example, when a wall is removed from a design, the openings, such as windows and doors, located inside that wall have to be removed also. As another example, a column should not be removed unless the references from the components such as walls, beams, slabs that
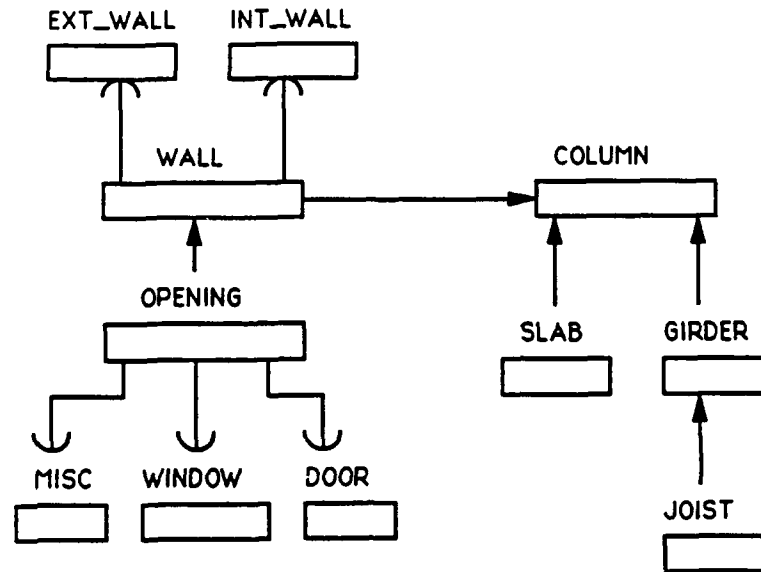
Figure 2: An Example of "Supporting" and "Supported-by" Relationships Implemented Through Structural Connections

the column supports no longer exist. This dependency property can be modeled using the reference connection as shown in Figure 2.

One application of the "connected-to" relationship in structural engineering is for the description of a joint connecting structural elements. As an example, we can represent a joint connection that is described in an interactive modeling system for the design of steel framed structure (Steelcad) [28] : "The connected members and the joint are logically linked in the database:

- If a joint is removed, then the previously connected members are automatically restored, as they were before the connection was defined.

- If a member is removed, then all connections that it has with other members are also removed and the other members reappear accordingly."

As shown in Figure 3, this description of a joint can be modeled using the three basic connection types. We assume that a joint connection may consist of one or more connectors. Each connector references a structural member and a connecting element. That is, deleting a connector does not affect the connecting members. On the other hand, if a structural member or a connecting plate is removed, the connectors are also removed.

4 An Object Management System in a Relational Framework

In the previous section, we have briefly discussed the structural data model and its three formal connection types. We have shown that the model can be used to
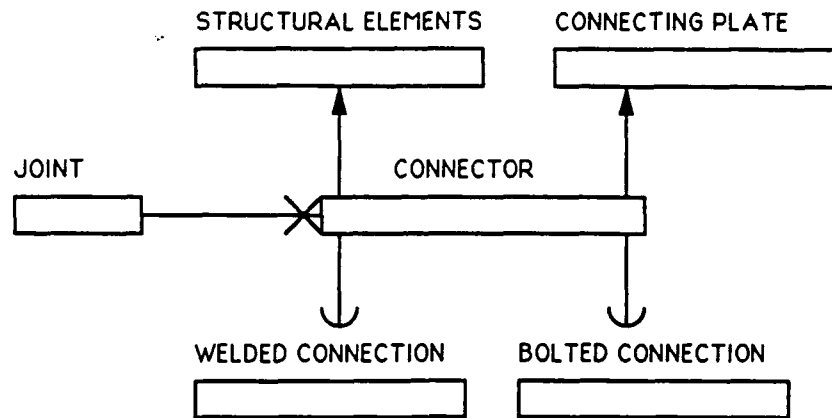
Figure 3: Definition of a Joint Connection

represent various relationships that are useful in structural engineering applications. In this section, we describe an architecture, based on the structural data model, for the management of complex objects in a relational framework and a prototype implementation of that approach in the PENGUIN system. In addition to its ability to capture the semantic relationships among the entities, PENGUIN can handle multiple views of design data and multiple representations of design objects.

4.1 General Principles

The object-oriented paradigm has gained much attention in computer aided design in recent years; it offers many advantages over traditional relational data model. Object-oriented systems help managing related data having complex structure by combining them into objects. The use of objects permits the user to manipulate the data at a higher level of abstraction. However, storing objects poses a problem when these objects are to be shared by multiple engineering design tasks. The amount of information pertaining to an object grow as each design task requires different information about the object. As a design progresses, an object may become too complex to be efficiently managed. The benefits of understandability and naturalness of having objects are lost [31]. Besides the problem of object sharing, object-oriented systems do not provide those indispensable features of DBMSs such as file management structures and concurrency control. Processing of queries involing large and complex sets of data is also not well supported by object-oriented systems.

Another approach is to store the objects explicitly in a relational database. In engineering application, we deal with entities that are more complex than single tuples or sets of homogeneous tuples. Quite frequently, an object is a hierarchical group of tuples comprising of a single root tuple that defines the object, and one or more dependent tuples that further describe the object's properties. Because of normalization theory, these dependent tuples reside in one or more relations distinct from the relation containing the root tuple. Even if such a structure is easily expressed relationally (through joins), it cannot be manipulated as a single entity. We need to

make such structures explicit known to the system. Furthermore, as noted earlier, different users require different views of the information included in an object. Update anomalies and problems of redundancy would arise if the objects corresponding to the different views were to be stored as such lattice. Last but not least, changes to the set of classes and to the inheritance can be made quite frequently at various stages of a design project. If the objects were explicitly stored, the schema would have to be changed accordingly.

Our approach is therefore to define and manipulate complex objects that are constructed from base relations. Our prototype implementation, PENGUIN, keeps a relational database system as its underlying data repository. Indeed, we believe that the relational model should be extended rather than replaced. The relational model has become a *de facto* standard and thus some degree of upward compatibility should be kept between the relational format and any next-generation data model. We augment the relational model with the structural data model. The PENGUIN system uses the structural model together with the traditional data schema to define the object schema. The idea is not to store the objects directly in persistent form but rather to store their description, which are used later to instantiate objects as needed.

4.2   Architecture

Both the concepts of view and object are intended to provide a better level of abstraction, bringing together related elements, which may be of different types, into one unit. For example, to display a building frame, we bring together structural entities, such as beams, columns and connections. The architecture for combining the concepts of views and objects has been initially proposed by Wiederhold [31]. A set of base relations serve as the persistent database and contain all the data needed to create any specific view or object. We then extract the data corresponding to a view-object from a relational database system and assemble the data into object instances based on the definition of the object template specified in terms of the connections of the structural data model [2, 5, 6, 7]. Multiple layers of view-objects can be defined so that a view-object can be expressed in terms of other view-objects and can be shared by other view-objects. Figure 4 summarized this notion of multiple object layers and their interaction with the underlying set of relations.

A prototype system (PENGUIN) for this object-based architecture is being implemented by Barsalou [4]. The schematic diagram of the PENGUIN system architecture is shown in Figure 5. The system architecture consists of three basic components: an *object (template) generator*, an *object instantiator* and an *object decomposer*. Each view-object is defined by an object template. The *object generator* maps relations into object templates where each template can invoke join (combining two relations through shared attributes) and projection (restricting the set of attributes of a relation) operations on the base relations. To define an object template, we first select a pivot relation from a set of base relations. The key of the pivot relation corresponds to the primary object key. Further data is related to this object by following the connections of the structural data model. By organizing an object template around a
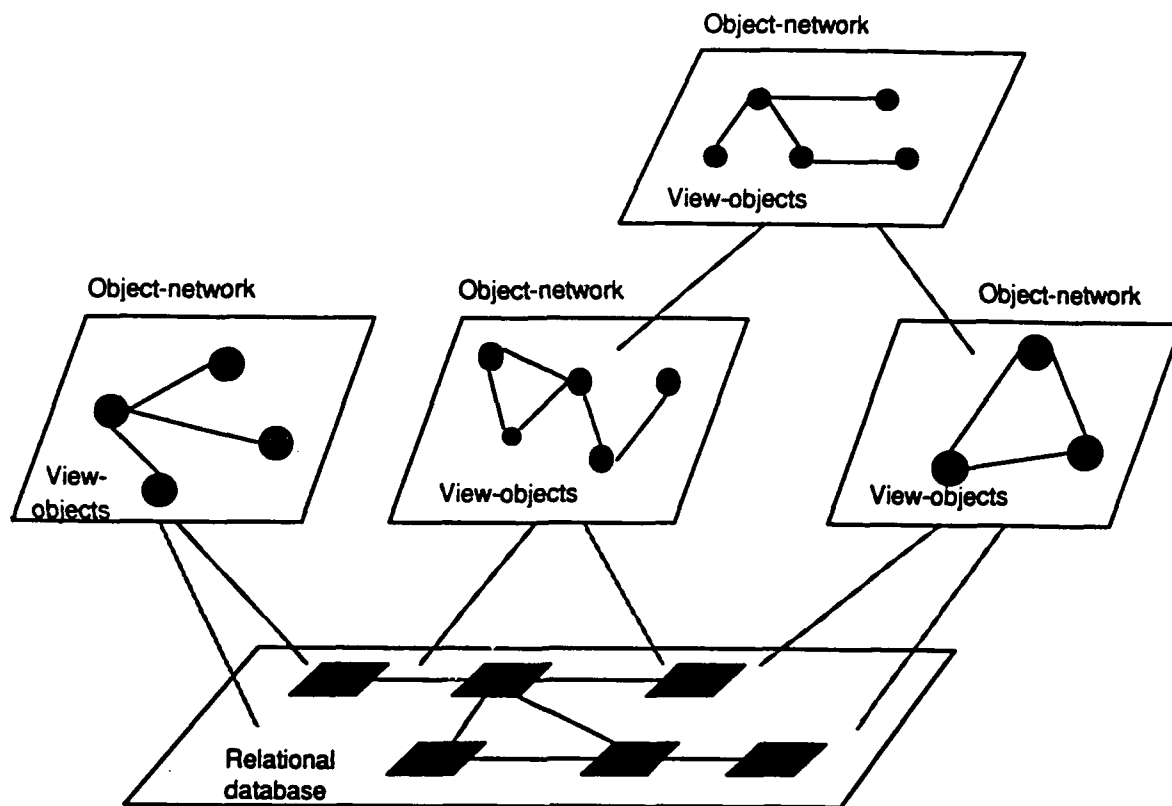
Figure 4: Multiple View-Objects and Object Networks for Sharing Information Stored in a Relational Database

pivot relation, each object instance can be uniquely identified by the value of the key of its pivot relation. The relations that are connected to the pivot relation through structural connections become potential candidate relations to be included in the object template. Once the pivot relation is specified, PENGUIN automatically derives a set of candidate relations from the connections of the structural data model. Secondary relations and their attributes (including those of the pivot relation) can then be selected from the set of candidate relations. Once an object template is defined, data access functions are derived to facilitate the data retrieval process. Related templates can be grouped together to form an object network, identifying a specific object view of the relational database. The whole process is knowledge-driven, using the semantics of the database structure as defined by the connections of the structural data model.

The *object instantiator* provides nonprocedural access to the actual object instances. The instantiator performs all the operations for information retrieval and manipulation that are necessary to instantiate and display an object template. A declarative query specifies the template of interest. Combining the database-access function and the specific selection criteria, the system automatically generates the relational query and transmits it to the relational database system, which in turn transmits back the set of matching relational tuples. The retrieved tuples are assem-
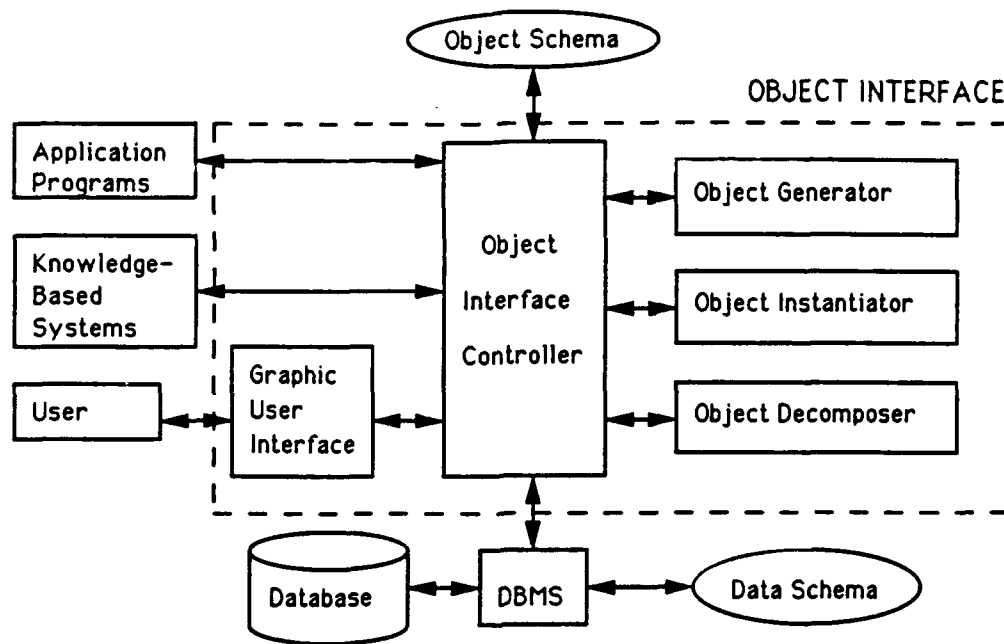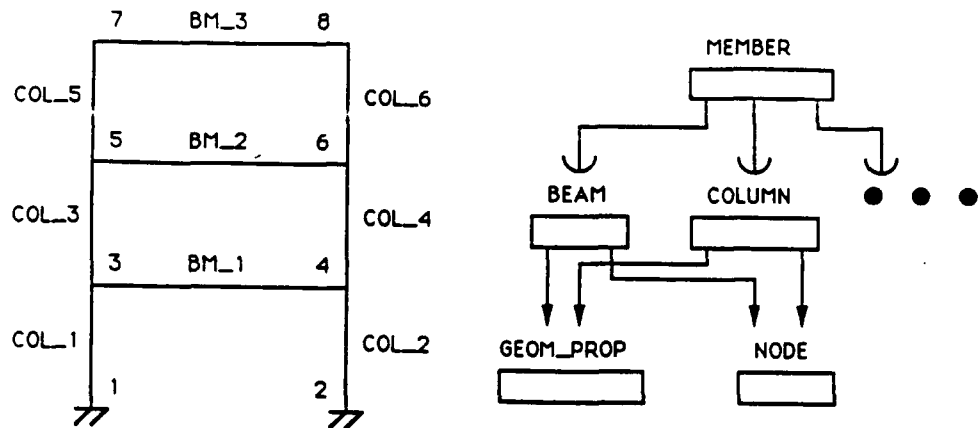
135

Figure 5: PENGUIN's Architecture: An Object Layer on Top of a Relational DBMS

bled into the desired object instances, based on the semantics defined in the object template.

The *object decomposer* maps the object instances back to the base relations. This component is invoked when changes to some object instances need to be made persistent at the database level. An object is generated by collapsing (potentially) many tuples from several relations. Similarly, one update operation on an object may result in a number of update operations on several base relations. Dependency constraints are enforced to ensure the database consistency. These actions are based on the integrity rules imposed by the connections of the structural data model. Since the object templates are defined using join operations on the database relations, we are then facing the well-known problem of updating relational databases through views involving multiple relations [17]. Updating through views is inherently ambiguous, as a change made to the view can translate into different modifications to the underlying database. Keller has shown that, using the structural semantics of the database, one can enumerate such ambiguities, and that one can choose a specific translator at view-definition time [20]. Because of the analogy between relational views and PENGUIN's object templates, Keller's algorithm applies to our approach. When creating a new object template, a simple dialogue, the content of which depends on the structure of the object template, allows the user to select one of the semantically valid translators. The chosen translator is then stored as part of the template definition. When an object instance is modified, the object decomposer will use this information to resolve any ambiguity completely and to update the database correctly [5].

136

4.3 An Example

Relation Member

Member_ID	Type	...
BM_1	Beam	...
BM_2	Beam	...
BM_3	Beam	...
COL_1	Column	...
COL_2	Column	...
COL_3	Column	...
COL_4	Column	...
COL_5	Column	...
COL_6	Column	...

Relation Beam

Beam_ID	Design	N_ID1	N_ID2	...
BM_1	W24X76	3	4	...
BM_2	W24X68	5	6	...
BM_3	W24X68	7	8	...

Relation Column

Column_ID	Design	N_ID1	N_ID2	...
COL_1	W14X82	1	3	...
COL_2	W14X82	2	4	...
COL_3	W14X61	3	5	...
COL_4	W14X61	4	6	...
COL_5	W14X53	5	7	...
COL_6	W14X53	6	8	...

Relation Node

NODE_ID	X	Y	Z	...
1	0.0	0.0	30.0	...
2	30.0	0.0	30.0	...
3	0.0	12.0	30.0	...
4	30.0	12.0	30.0	...
5	0.0	24.0	30.0	...
6	30.0	24.0	30.0	...
7	0.0	36.0	30.0	...
8	30.0	36.0	30.0	...

Relation Geom_Prop

Design	S	A	I	...
W24X76	176	22.4	2100	...
W24X68	154	20.1	1830	...
W14X53	77.8	15.6	541	...
W14X61	92.2	17.9	640	...
W14X82	12.3	24.1	882	...

Figure 6: A Set of Base Relations for a Simple Frame Structure

We now illustrate the construction of an object network using the PENGUIN system with a simple structural engineering example. Figure 6 shows a set of base relations and their connections that describe a simple frame structure. As shown in Figure 7, the relations and the connections are specified using PENGUIN's graphical interface. To create a complex "MEMBER OBJECT", we define a template organized around the pivot relation "MEMBER" in the underlying database. Each instance of the "MEMBER OBJECT" is thus uniquely identified by the key attribute values of the pivot relation "MEMBER". Once the pivot relation has been specified, a candidate

137

graph containing the valid relations is derived and is converted into a candidate tree where the root is the pivot relation and all other nodes are secondary relations that can be included in the object template. The candidate relations for the "MEMBER OBJECT" template are shown in Figure 8.
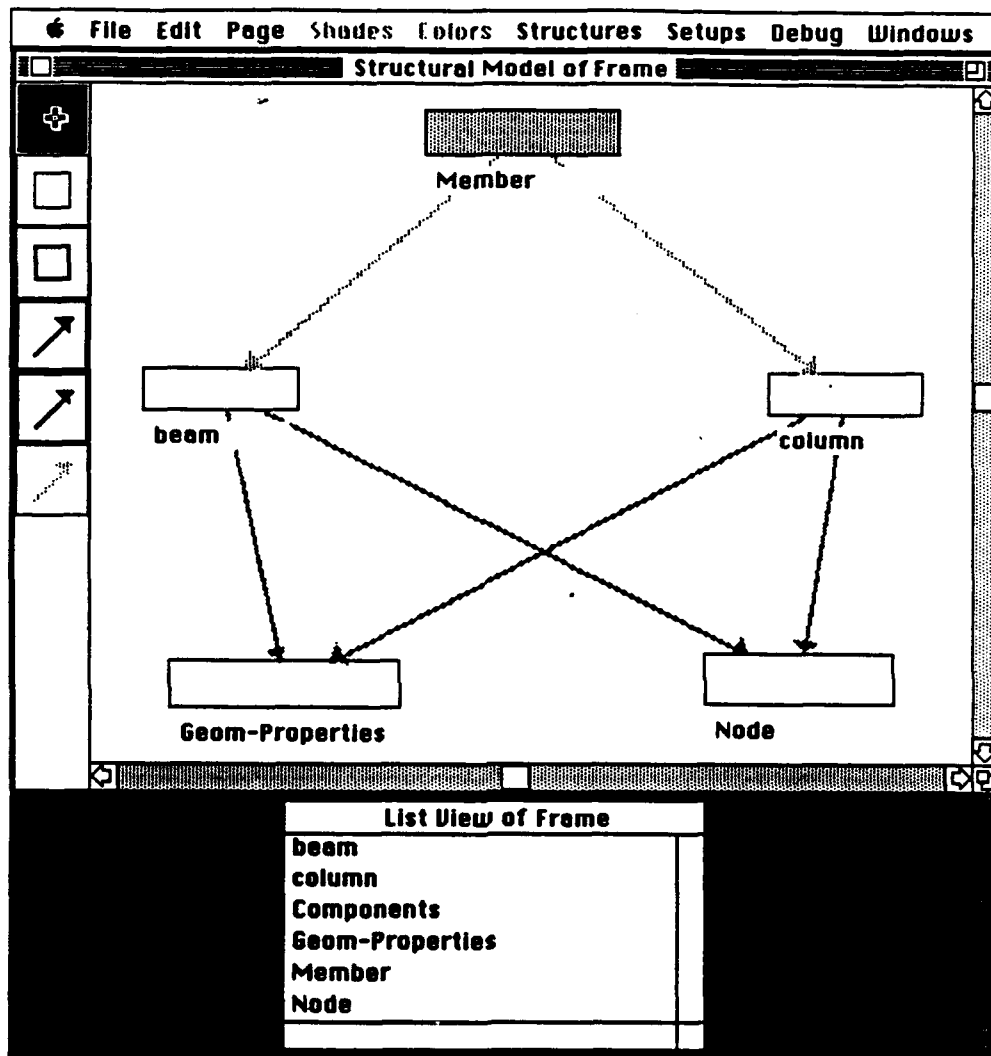

Figure 7: Defining an Object Template Using PENGUIN's Graphical Interface

Once the candidate tree is established, the user can specify any number of secondary relations and their attributes to be included in the object template. For example, we can include the information about beam members as shown in Figure 9 and define it as "BEAM OBJECT". Based on this information, the system automatically derives the database access function, the linkage of the various data elements within the object, and the compulsory attributes that are required for performing join operations on the selected relations. Such a view-object can now be exploited by engineering applications. For example, the query (retrieve BEAM-OBJECT with Member-Id = 'BM-1') would fetch the view object instance displayed in Figure 10.

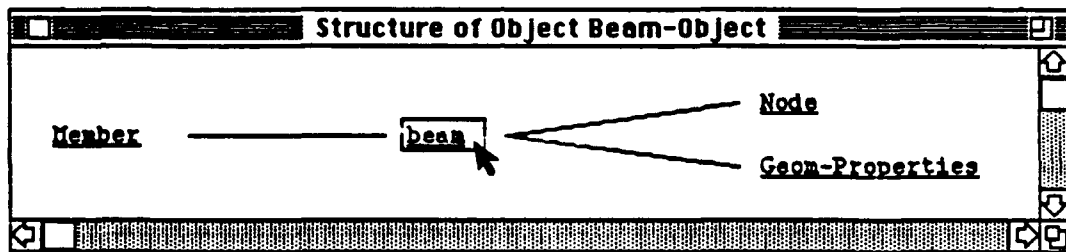Figure 8: The Candidate Tree of an Object Template Generated by PENGUIN

Figure 9: Definition of a View-Object for Beams

Now, let us assume that a substructure-component definition and the corresponding relations have been entered as shown in Figure 11. We can then create an object template "SUBSTRUCTURE OBJECT" with the pivot relation "SUBSTRUCTURE". As shown in Figure 12, this newly created object template can be inserted into an object network by simply connecting it to other object templates previously defined (thereby updating the object schema.) Since the object-network connections are abstracted from the underlying database structural connections, the relationships are explicitly carried over to the object layer and can be used to provide inheritance of attributes among the templates.

5 View-Objects and Design Abstractions

As the design process progresses from conceptualization to design, the way to represent the design is constantly changing. For an integrated design system to be effective, the database system must be able to accomodate the "growth" of the design. As mentioned earlier, the data model must support a wide variety of design representations, sharing the same information in the engineering model. In addition, the data model must allow dynamic changes of the object schema, reflecting the evolutionary process of design, and must be able to minimize database reorganization. The view-object facility described in the previous section allows the engineer to select the object information pertinent to a design task and to ignore the irrelevant details. Furthermore, the separation of the object schema and the database schema can facilitate schema
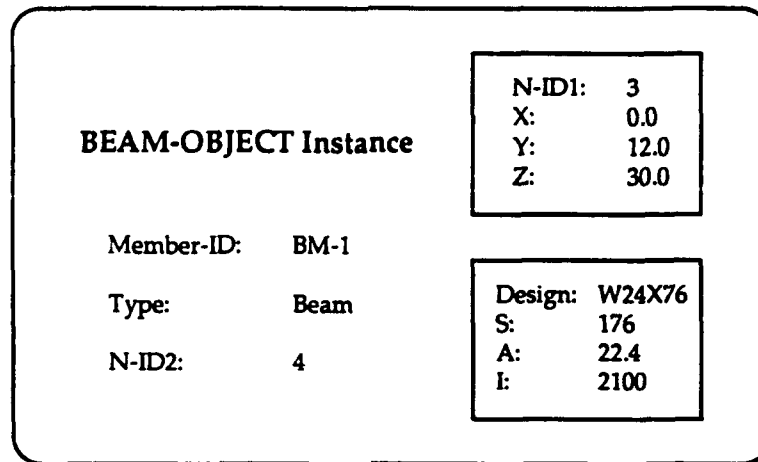
139

Figure 10: An Instance of the View-Object BEAM-OBJECT

evolution during the design process.

An abstraction view of a building and its components is not unique. An engineer abstracts a specific view of design to focus on a particular task. As an example, an hierarchical decomposition of a building structure is shown in Figure 13(a). As shown in Figure 13(b), for design purposes, a floor composed of beams (including girders and joists), slabs, columns can be conveniently treated as objects. For analysis purposes, a building frame composed of girders and columns is defined as shown in Figure 13(c). We see here two multiple design views sharing the same information. For the respective views, the attributes of a shared entity are not the same. For the description of a floor plan, only the location and orientation of the columns are important. For frame analysis, however, the location, the dimension and the properties of the columns are needed.

By allowing the user to select any number of secondary relations for inclusion, an object can be specified to any level of details that is desired. For example, in the earlier analysis stage, a floor may be treated as the basic component entity but may be expanded to include other subcomponents such as beams and slabs at a later stage of the design; an object schema can be changed dynamically as the design evolves. Hence, the complexity of a design can be managed by suppressing the irrelevant details as necessary. Furthermore, the description of an entity can be refined as needed.

In Figure 13(c), the entity "FRAME" is composed of subentities "GIRDER" and "COLUMN". However, removing a "FRAME" instance does not necessarily imply that all its constituent components be deleted as well. As shown in Figure 13(d), an alternative may be to augment the "FRAME" object with the auxiliary relations "FRAME GIRDER" and "FRAME COLUMN", which reference relations "GRIDER" and "COLUMN", respectively, and are owned by relation "FRAME". This new structure provides an associative relationship such that removing a "FRAME" instance does not affect the base relations "COLUMN" and "GIRDER"; however, a column cannot be deleted as long as a structural frame containing that column exists. A similar view can be created for the abstract entity "FLOOR". It should be empha-
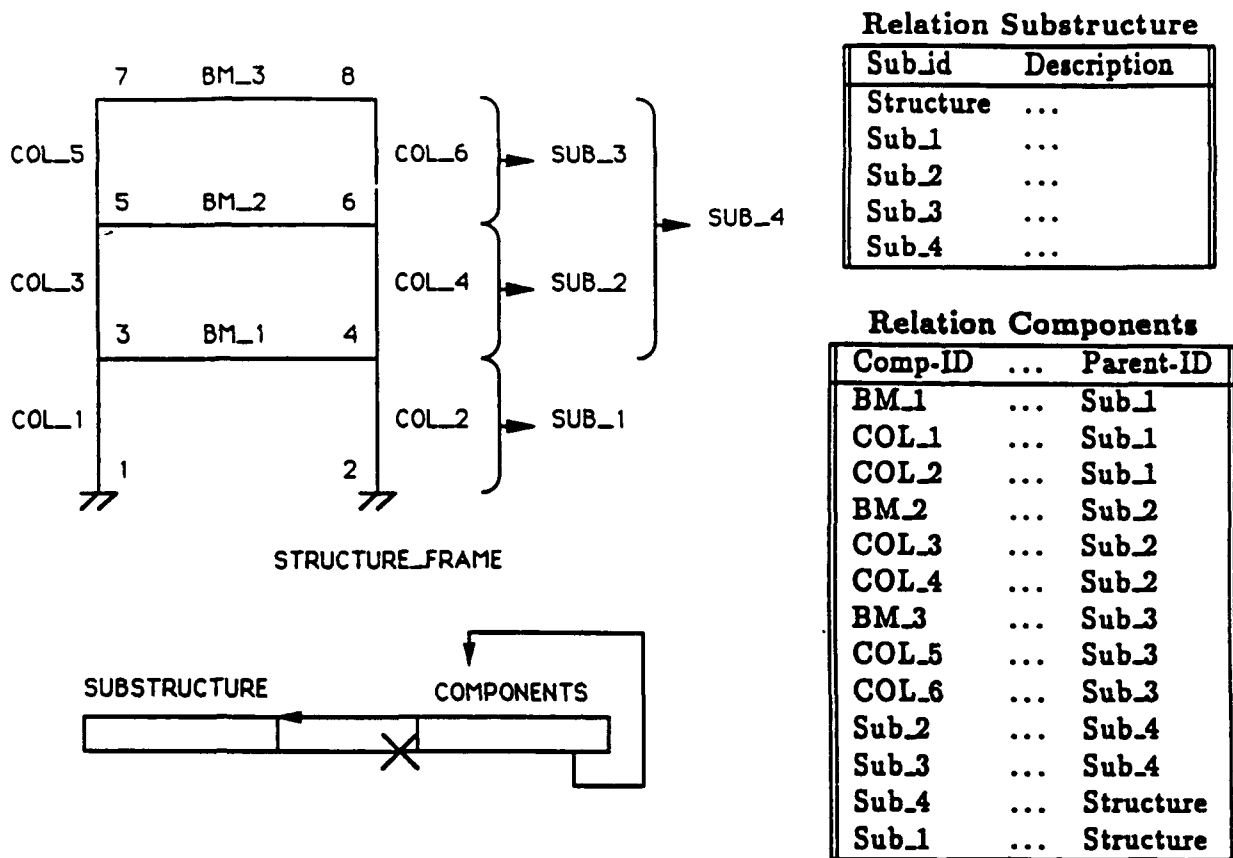
Relation Substructure

Sub_id	Description
Structure	...
Sub_1	...
Sub_2	...
Sub_3	...
Sub_4	...

Relation Components

Comp-ID	...	Parent-ID
BM_1	...	Sub_1
COL_1	...	Sub_1
COL_2	...	Sub_1
BM_2	...	Sub_2
COL_3	...	Sub_2
COL_4	...	Sub_2
BM_3	...	Sub_3
COL_5	...	Sub_3
COL_6	...	Sub_3
Sub_2	...	Sub_4
Sub_3	...	Sub_4
Sub_4	...	Structure
Sub_1	...	Structure

Figure 11: A Set of Base Relations for the Definition of Substructures

sized that modifications made to an individual object template does not necessarily lead to modification of a higher level object. Conversely, modifications of the object network do not affect the definition of the base relations.

6   Summary and Discussion

In this paper, we have examined the potential applications of the structural data model in structural engineering. The connections of the structural data model properly define the various relationships, and their constraints and dependencies that are of interest to researchers in computer aided building design. Besides being an effective database design tool, the structural data model can serve as the basis for the development of an object interface to a relational database system, supporting multiple object views. The architecture of an object management system has also been briefly described. The application of this object model for structural analysis and design has been discussed.

Many advantages of this view-object approach can be identified:

- It provides multiple views of the stored information that is relevant to an engineering model or design
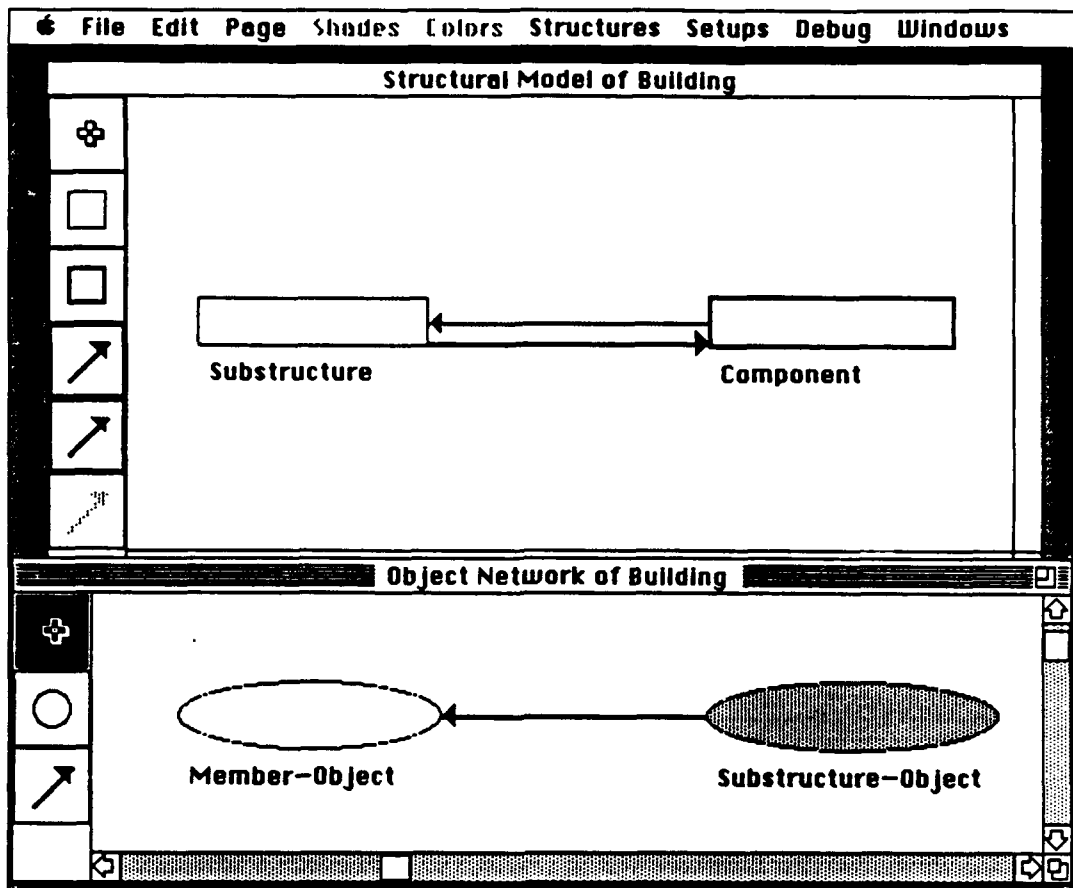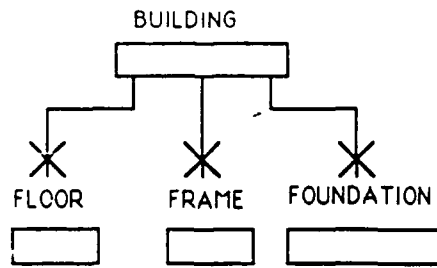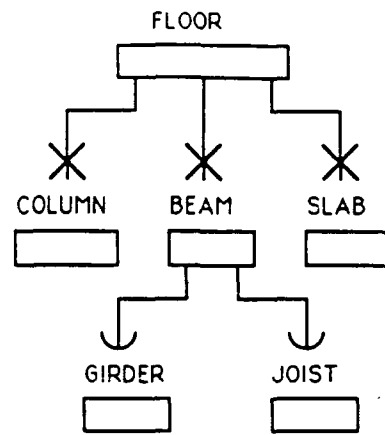
141

Figure 12: Inserting a New Object Template into an Object Network Using PENGUIN

- It provides a mechanism for storing objects, independent of one specific view or application

- It eliminates the data redundancy since the information about an object is shared but not duplicated

- It maintains integrity of the data for a given object based on the structural constraints that are specified by the connections

- It supports growth of a design because of the logical independence between the object definitions and the database schema.

The key benefit of the view-object interface to a relational system, besides information sharing, is that any new attributes and/or relations added to the underlying database do not affect the object definitions. Conversely, changes in the definition of any objects do not affect the schema of the underlying database. In other words, the architecture is sufficiently flexible to allow growth as design evolves.
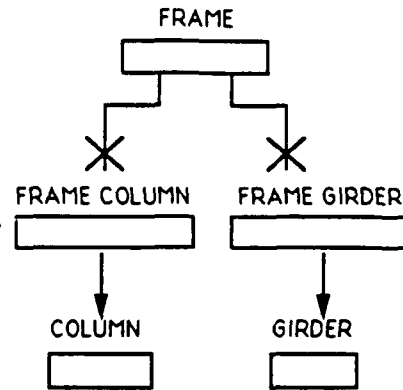
142

(a) A Hierarchical Decomposition of
   a Building

(b) Decomposition of a Floor Entity
    for Design Purpose

(c) Decomposition of a Frame Entity
    for Analysis Purpose

(d) Alternative Definition of a
    Frame Entity

Figure 13: Multiple Views and Representations of Design Objects

7 Acknowledgment

References

[1] Ackroyd, M.H.; Fenves, S.J.; McGuire W. (1988) Computerized LRFD Specification. National Steel Construction Conference.

143

[2] Barsalou, T. (1987) An Object-Based Interface to a Relational Database System. Technical Report KSL-87-41, Knowledge Systems Laboratory, Stanford University.

[3] Barsalou, T.; Wiederhold, G. (1987) Applying a Semantic Model to an Immunology Database. The Eleventh Symposium on Computer Applications in Medical Care, IEEE Computer Society, pages 871-877.

[4] Barsalou, T. (1988) An Object-Based Architecture for Biomedical Expert Database Systems. The Twelfth Symposium on Computer Applications in Medical Care, IEEE Computer Society, pages 572-578.

[5] Barsalou, T. (1990) Deriving View Objects from Relational Databases. PhD Thesis (under preparation), Medical Information Sciences Program, Stanford University.

[6] Barsalou, T.; Wiederhold, G. (1989) Knowledge-directed mediation between application objects and base data. Proceedings of the Working Conference on Data and Knowledge Base Integration, University of Keele, England.

[7] Barsalou, T.; Wiederhold, G. (1989) Knowledge-based mapping of relations into objects. (Submitted for Publication) Computer Aided Design.

[8] Bjork, B.-C. (1989) Basic Structure of a Proposed Building Product Model. Computer Aided Design 21:71-78.

[9] Blaha, M.R.; Premerlani, W.J.; Rumbaugh, J.E. (1988) Relational Database Design using an Object-Oriented Methodology. Communications of the ACM, 31(4):414-427.

[10] Brodie, M.L. (1982) On the Development of Data Models. In: On Conceptual Modeling (Eds. M.L. Brodie, J. Mylopoulos and J.W. Schmidt). Springer-Verlag, pages 19-48.

[11] Brodie, M.L. (1983) Association: A Database Abstraction for Semantic Modelling. In: Entity-Relationship Approach to Information Modeling and Analysis (Ed. P.P. Chen). Elsevier Science Publishers, pages 577-602.

[12] Bryant, D.A.; Dains, R.B. (1977) Models of Buildings in Computers: Three Useful Abstractions. IF, 8(2):9-14.

[13] Darwiche, A.; Levitt, R.E.; Hayes-Roth, B. (1988) OARPLAN: Generating Project Plans by Reasoning about Objects, Actions and Resources. The Journal of Artificial Intelligence in Engineering Design, Analysis and Manufacturing, 2(3):169-181.

[14] Eastman, C.M. (1978) The Representation of Design Problems and Maintenance of Their Structure. IFIPS Working Conference on Application of AI and PR to CAD, IFIP, pages 1-23.

[15] Eastman, C.M. (1988) Automatic Composition in Design. Design Theory '88 (Eds. Newsome, S.L., W.R. Spillers and S. Finger). 1988 NSF Grantee Workshop on Design Theory and Methodology.

[16] ElMasri, R.; Wiederhold, G. (1979) Database Model Integration Using the Structural Model. Proceedings of the ACM-SIGMOD Conference, pages 191-198.

[17] Furtado, A.L.; Casanova, M.A. (1985) Updating Relational Views In: Query Processing in Database Systems (Eds. W. Kim, D.S. Reiner and D.S. Batory). Springer-Verlag, New York, NY.

[18] Garrett Jr., J.H.; Breslin, J.; Basten, J. (1988) An Object-Oriented Model for Building Design and Construction. Extended Abstract, Department of Civil Engineering, University of Illinois, Urbana, Illinois.

[19] Garrett Jr., J.H.; Basten, J.; Breslin, J.; Andersen, T. (1989) An Object-Oriented Model for Building Design and Construction. Computer Utilization in Structural Engineering, Structures Congress, ASCE, pages 332-341.

[20] Keller, A.M. (1986) The Role of Semantics in Translating View Updates. IEEE Computer, 19(1):63-73.

[21] Lavakare, A.; Howard, H.C. (1989) Structural Steel Framing Data Model. Technical Report 12, Center for Integrated Facility Engineering, Stanford University.

[22] Law, K.H.; Jouaneh, M.K. (1986) Data Modeling for Building Design. Fourth Conference on Computing in Civil Engineering, ASCE, pages 21-36.

[23] Law, K.H.; Jouaneh, M.K.; Spooner, D.L. (1987) Abstraction Database Concept for Engineering Modeling. Engineering with Computers, 2:79-94.

[24] Law, K.H.; Barsalou, T. (1989) Applying a Semantic Structural Model for Engineering Design. ASME International Conference on Computers in Engineering, Anaheim, CA, pages 61-66.

[25] Powell, G.H.; Bhateja, R. (1988) Database Design for Computer Integrated Structural Engineering. Engineering with Computers, 4(3):135-144.

[26] Smith, J.M.; Smith, D.C.P. (1977) Database Abstraction: Aggregation and Generalization. ACM TODS, 2:105-133.

[27] Spillers, W.R.; Newsome, S. (1988) Design Theory: A Model for Conceptual Design. Design Theory '88 (Eds. Newsome, S.L., W.R. Spillers and S. Finger) 1988 NSF Grantee Workshop on Design Theory and Methodology.

[28] STEELCAD (1989), Cadex Ltd.

[29] Vernadat, F.B. (1984) A Commented and Indexed Bibliography on Data Structuring and Data Management in CAD/CAM : 1970 to Mid-1983. National Research Council of Canada, Technical Report 23373.

[30] Wiederhold, G.; ElMasri, R. (1980) The Structural Model for Database Design. Entity-relationship Approach to System Analysis and Design. North-Holland, pages 237-257.

[31] Wiederhold, G. (1986) Views, Objects, and Databases. IEEE Computer, 19(12):37-44.

[32] Wiederhold, G. (1989) Connections. In: Managing Objects in a Relational Framework. Technical Report STAN-CS-89-1245, Department of Computer Science, Stanford University.

Prescribing Inner/Outer Joins for Instantiating Objects from Relational Databases through Views

Byung Suk Lee
Electrical Engineering
Stanford University

Gio Wiederhold
Computer Science
Stanford University

Abstract

When objects are instantiated from normalized relational databases through views, outer joins may be needed to prevent information loss. This paper develops a mechanism for prescribing inner/outer joins from the semantics of a system model. A view is defined by a complex query, and has additional features for resolving the mismatch between program objects and database relations. Some object attributes allow null values to prevent loss of instances due to nonmatching tuples of join operations. This mandates outer joins for some of the join operations in the complex query. Using a structural data model, we can minimize the number of outer joins by exploiting the cardinality constraints of connections, so that the query can be processed more efficiently. To present our ideas, we first describe the system model in terms of object type model, data model, and query model and secondly, describe the development of an algorithm for classifying edges into inner join edges and outer join edges.

1 Introduction

One of the major research issues these days is to integrate object-oriented programs with databases so that applications working in object-oriented environment can have shared, concurrent accesses to persistent storage. Roughly there have been two alternative approaches. One is to use object-oriented model uniformly for applications and persistent storage [1, 2, 3]. The other is to use object-oriented model for applications and relational model for storage [4, 5]. In this approach, the system is configured as a front-end system with an object-oriented model and conventional relational databases as back-end storage. Objects are instantiated by evaluating complex queries to databases. Our approach belongs to this category but has a bit different perspective, *view-object* [6].

View-object concept was first proposed by Wiederhold [6] as an effective tool for merging the concepts of database views and program objects. Views interface between two different paradigms - objects and relations - by retrieving object instances in the form of nested tuples from a set of relations. Subsequently Cohen [7] implemented a slightly modified concept of Wiederhold's view-object in Prolog domain using hierarchical data model. Barsalou et al. [8] implemented a view-object generator in Lisp domain using a relational database. Our view-object concept has the advantage of being able to use existing databases[1] with effective sharing and concurrency control. However, an open question is the performance of a mixed paradigm system.

When objects are instantiated from a set of normalized relations by evaluating complex queries, outer joins[2] [9] may be needed to prevent information loss. The typical case is when we instantiate

---

[1] We cannot throw away relational databases in a decade. Remember that the IMS hierarchical data model is still prevalent now when we call the relational model 'conventional'.

[2] Precisely speaking, these are *left outer joins*.

objects which allow *null* values for some of their attributes. For example, an object of type *Employee* may allow null values for its *department*, *salary*, and *children* attributes. Instantiating objects generally involves joining over multiple normalized relations. Therefore, allowing null values for object attributes mandates retrieving the tuples of a relation that do not have matching tuples on the joined relations. To achieve this, we need to execute *outer joins*. However, outer joins are more costly than inner joins and should be avoided if possible.

The current state-of-art is that query optimization with outer joins has been neglected since outer joins are not directly available in relational languages. Even if outer joins are available, they must be specified manually by programmers. Our principle is to derive most semantics for deciding on outer joins from the system model to minimize the overhead on a human programmer. We thus identify exploitable semantics from the application and database model such as object type, database schema, and query structure. It is the objective of this paper to present a mechanism for deciding whether to perform the joins of a complex query as inner joins or outer joins.

We assume the programmer defining object types declares an attribute as deliberately allowing a null value by attaching a *null-allowed* option to the attribute. This is equivalent to specifying the constraint of 'minimum cardinality = 0' on the attribute[3]. Attributes without null-allowed options are prohibited from having null values. That is, attributes have *null-forbidden* options by default. These are mapped to so called *−nodes* of a query graph. A −node is a relation occurrence restricted by a selection condition '$A \neq$ null' where A is a set of relation attributes mapped from an object attribute. On the other hand, the null-allowed option is mapped to so called *+edges* of a query graph. A +edge is a 'prescription' for evaluating the edge by an outer join, unless it is overridden by the decision on an inner join from the semantics of data model.

It turns out a *join cardinality constraint* (JCC) is important for this purpose. A JCC is a concept generalized from the connection cardinality constraint of a structural model [11, 15], extended to incorporate non-connection joins. A structural model is essentially a relational model, augmented with connections to incorporate interrelational integrity constraints into the data model. We assume the information about these connections is stored in a system table called a *connection catalog*. In fact, this table provides almost all the JCC values for a complex query. A query has additional structure for resolving the mismatch between program objects and database relations. We restrict queries to select-project-join expressions d do not yet support recursive queries. A query is translated into a query graph before being processed. Our query graph model is similar to that used by Finkelstein [10], but peculiar in that ours distinguishes joins into *connection joins*, equijoins, and general joins.

Following this introduction, we first describe the system model in the order of object type model, data model, and query model in Sections 2, 3, and 4, respectively. Our object type model is similar to that of O2 [14] and supports a subset of the attribute type constructors available in O2. The data model is based on the structural model with extended connection cardinalities [15]. In the query model, we extend the concept of *pivots* from Barsalou et al.'s work [8] to incorporate so called *abstract pivots*. Then in Section 5, we develop a mechanism for prescribing inner/outer joins from the semantics available in these models. Specifically we first develop a mechanism for mapping null-allowed options on object attributes to +edges and null-forbidden options to −nodes of the query graph. Secondly, we generalize the cardinality constraint available from the structural model to the join cardinality constraint, and finally we develop the algorithm for classifying edges into inner join edges and outer join edges based on these results. It is followed by a conclusion in Section 6.

---

[3]Many commercial tools for building object-oriented system applications, KEE for example, actually have this option.

2 Object Type Model

An object type is defined as a tuple of attributes, i.e., Type $O[A_1, A_2, \cdots, X_1, X_2, \cdots]$ where O is the type name, A_i is a simple attribute, and X_i is a complex attribute.

An attribute is described in Backus-Naur Form as follows. ({ } denotes a set and [ ] denotes a tuple.)

        attribute == simple attribute | complex attribute
        simple attribute == internal attribute | reference attribute
        complex attribute == [attribute, attribute, $\cdots$] | {[attribute, attribute, $\cdots$]}.

A *simple* (or *atomic*) attribute defines a subobject by itself and has no subobject of itself. The subobject defined by a simple attribute is either internal or external to the object. An *internal* attribute has a primitive data type such as string, integer, etc., while an *external* (or *reference*) attribute has another object type name as its data type. We assume the value of a reference attribute is retrieved as the identifier (id) of the referenced object.

A *complex* attribute defines a subobject by embedding its type definition internally as part of the object type. The type of a complex attribute is a tuple or a set of tuples. A *tuple* defines a collection of attributes with different data types while a *set* defines a collection of attributes with identical data type. For each object and its subobjects defined by complex attributes, we associate value-oriented object id's that are retrieved from databases.

Given an object type O and its attribute s_0, it is not clear whether we should allow a null value for s_0. To resolve this ambiguity, we require a programmer to declare null-allowed options on object attributes that should allow null values. For example, the following type is defined to retrieve programmers even if they have no managers or no assigned projects. Note, without null-allowed options, it is not clear whether we want 'a programmer *and* a manager' or a 'programmer *with* a manager'.

Example 1 Type Programmer
```
[ name: string,  dept: Department, salary: integer null-allowed,
  manager: Employee null-allowed,
  project: { [ title: string, sponsor: string null-allowed,
              leader: string null-allowed, dept: Department,
              task: string null-allowed ] } null-allowed
]
□
```

Null-allowed options on object attributes lead to retrieving null values from databases. There are two sources of null values from a database: from null values in a tuple, or from non-matching tuples. Inner joins can have the first source only, while outer joins can have both. In our model, we support both sources of null values and, therefore, use outer joins for null-allowed options.

Here we introduce two concepts derived from the object type: Oset and Ochain, which are important to facilitate mapping objects to relations.

Definition 1 (Oset) Given an object type O, Oset(O) is defined as the set whose elements are the object defined by O and all of its subobjects. The subobjects are recursively defined by nested complex attributes.

For example, Oset(Programmer) = {Programmer, Project}.

Definition 2 (Ochain) Given an object type $O[A_1, A_2, \cdots, X_1, X_2, \cdots]$, the chain of object-subobject relations from O to an attribute s_0, denoted by $\text{Ochain}(O, s_0)$ is defined as

$$\text{Ochain}(O, s_0) \equiv O_0.O_1.\cdots.O_n.s_0. \tag{1}$$

Here, O_0 is an object of type O; $O_1, \cdots, O_n$ are the subobjects of O_0 recursively defined at each level of nesting, i.e., O_i is a subobject of O_{i-1}, and s_0 is a subobject (O_{n+1}) of O_n if s_0 is a complex attribute or a simple attribute of O_n if s_0 is a simple attribute. (We assume there is no ambiguity of attribute names.)

For example, Ochain(Programmer, title) = Programmer.project.title and Ochain(Programmer, project) = Programmer.project.

3 Data Model

In this section, we describe how the structural data model, in particular the cardinality constraints of connections are used in our model. Connections in a structural model make it possible to describe entities in a more object-oriented manner than when we have relations only [12, 13].

Figure 1 shows the structural diagram of our sample database. Each connection is described by (From-att)Name(To-Att). Note some connections have non-default cardinality constraints declared by a database designer. For example, the [1,200] attached to Emp(works-for)Dept expresses the constraint 'For every department, there must exist at least one employee and can exist at most 200 employees.'

A structural model consists of seven relation types (entity relations, foreign entity relations, nest relations, associative relations, lexicons, subrelations, derived relations) and four connection types (ownership connection, reference connection, subset connection, identity connection). Relations are those of a relational model and take on roles depending on the connections to other relations.

In our model, we use the first three types of connections, i.e., ownership, reference, subset connections. (The identity connection describes derived and distributed data.) Given two relations R_1 and R_2, a connection from $A_1 \subseteq R_1$ to $A_2 \subseteq R_2$ satisfies the following condition.

- Ownership connection: $R_1.A_1 = \text{Key}(R_1) \wedge R_2.A_2 \subseteq \text{Key}(R_2) \wedge R_1.A_1 = R_2.A_2$

- Reference connection: $R_2.A_2 = \text{Key}(R_2) \wedge (R_1.A_1 = \text{null} \vee R_1.A_1 = R_2.A_2)$

- Subset connecction: $R_1.A_1 = \text{Key}(R_1) \wedge R_2.A_2 = \text{Key}(R_2) \wedge R_1.A_1 = R_2.A_2$

We augment these three types of connections with a *general* type of connection. It is defined for non-connection joins to retain cardinality constraints but no update constraints. We make cardinality constraints explicit by attaching a pair of minimum and maximum cardinalities to them. This was suggested by El-Masri et al. in [15] and can be regarded as a mechanism for describing domain knowledge. These extended cardinalities may be explicitly specified by a database designer, or use default values.

The symbols and default cardinality constraints (DCC) of these four types of connections are as shown below.

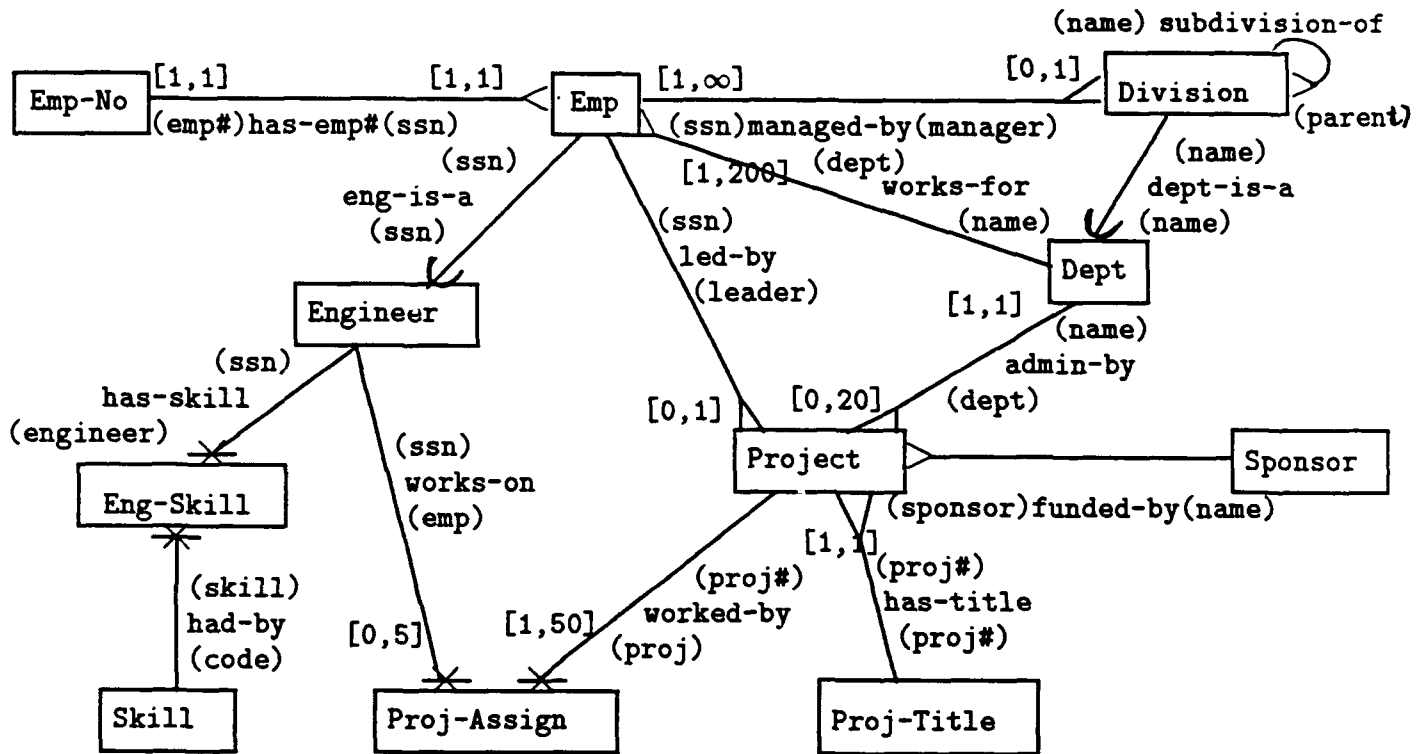| Type	Symbol	DCC	
		from	to
Ownership	—*	[1,1]	[0,∞]
Reference	>—	[0,∞]	[0,1]
Subset	—Э	[1,1]	[0,1]
General	—	[0,∞]	[0,∞]

Figure 1: The structural diagram of a sample database

For our purpose, we interpret the extended cardinality constraints as *join cardinality constraints*, that is, as the minumum and maximum number of matching tuples in the joined relation given a tuple of one relation[4]. A more rigorous discussion will appear in Section 5.2.

We assume the system keeps a so called *connection catalog* as part of its data dictionary.

Definition 3 (Connection catalog) A connection catalog (CC) is a table containing the description of connections, i.e., a connection catalog is a relation whose attributes are connection name, connection type, from-relation, from-attributes, to-relation, to-attribitues, minimum cardinality, maximum cardinality, inverse minimum cardinality, and inverse maximum cardinality.

Note a connection catalog may contain the cardinality constraints of equijoins or general joins under the type 'General' in addition to those of connection joins.

Example 2 (Connection catalog) The connection catalog of a sample database shown in Figure 1 contains the following entries. We show only the portion needed in this paper.

---

[4]Their semantics of constraining updates for maintaining integrity is not important here.

Name	Type	From-Rel	From-Att	To-Rel	To-Att	m_{12}	M_{12}	m_{21}	M_{21}
eng-is-a	subset	Engineer	ssn	Emp	ssn	1	1	0	1
works-on	ownership	Engineer	ssn	Proj-Assign	emp	0	5*	1	1
worked-by	ownership	Project	proj#	Proj-Assign	proj	1*	50*	1	1
has-title	reference	Project	proj#	Proj-Title	proj#	0	1	1*	1*
funded-by	reference	Project	sponsor	Sponsor	name	0	1	0	∞
led-by	reference	Project	leader	Emp	ssn	0	1	0	1*
admin-by	reference	Project	dept	Dept	name	1*	1	0	20*
works-for	reference	Emp	dept	Dept	name	0	1	1*	200*
dept-is-a	subset	Dept	name	Division	name	1	1	0	1
can-be-a	general	Division	name	Sponsor	name	0	1*	0	1*

where m_{12}, M_{12} are the minimum, maximum cardinalities, and m_{21}, M_{21} are the inverse of them. The *-ed values are non-default values explicitly declared by a designer. □

4 Query Model

This section describes the models of query and query graph, and discusses eliminating cycles from a query graph without affecting the result.

4.1 Query

Our query model supports *impedance matching*[5] between objects and normalized relations, as well as retrieving objects from a set of relations. Impedance matching in turn has two aspects: *semantic matching* and *structural matching*. To achieve semantic matching, we introduce the concept of an *abstract pivot*. An abstract pivot defines a *virtual relation occurrence* whose semantics directly matches that of an object type. For example, we want to instantiate type *ProjectLeader* from the sample database shown in Figure 1. We cannot find any relation that provides the instances of *ProjectLeader* directly. However, an inner join between *Emp* and *Project* over the connection *led-by* materializes "an employee who is leading a project" and thus matches the semantics of the type *ProjectLeader*. In this case the join expression defines an abstract pivot of a virtual *ProjectLeader*.

Structural matching maps object attributes to relation attributes. We restrict the queries to select-project-join expressions. A join expression is a conjunction of join predicates. Each join (predicate) is either a connection join, an equijoin, or a general join. A connection join is an equijoin but includes semantics such as update/cardinality constraints. We limit equijoins to those that are *not* defined over connections. A general join is neither a connection join nor an equijoin.

We evaluate outer joins as *left* outer joins. They are not symmetric in general so that the order of join operands is important. To emphasize this, we say a join is performed *from* one operand *to* another operand whenever necessary.

Figure 2 shows the functional structure of a query for mapping between objects and relations.

Definition 4 (Query) A query for instantiating an object type O is a triplet (JS, AMF, PD) where

1. JS (join set) is a set of joins between two *restricted relation occurrences* where

---
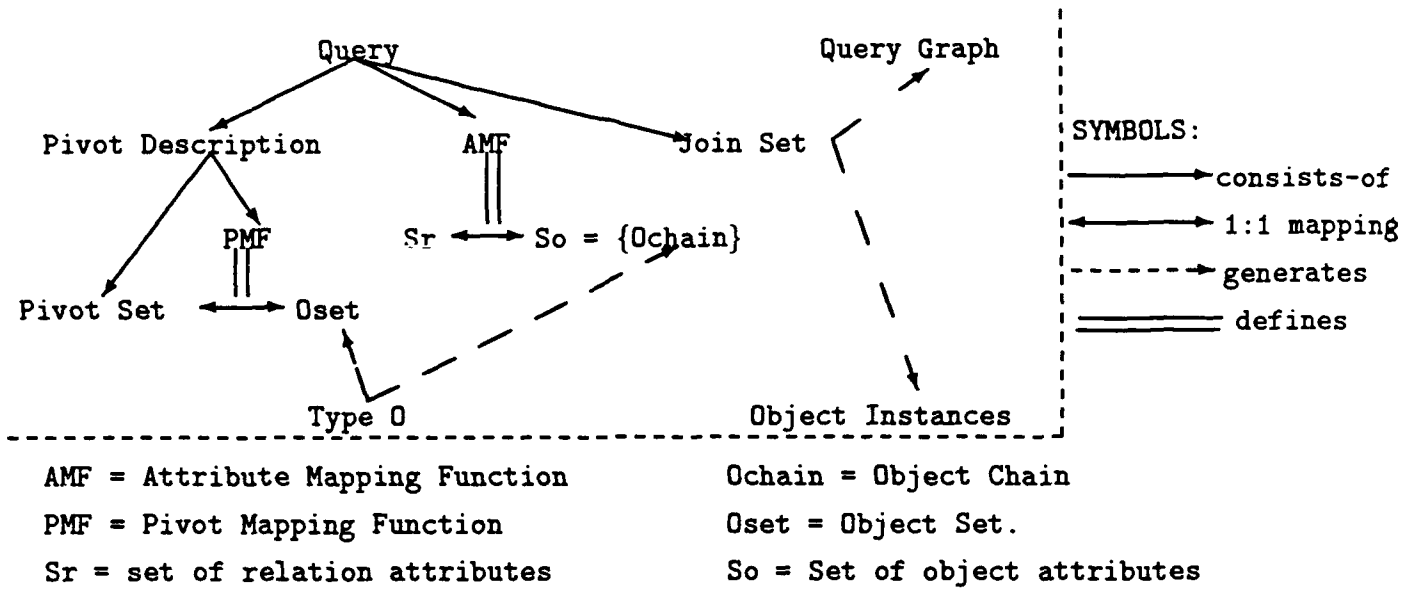[5]A counter term of *impedance mismatching* used by Maier and Bancilhon [16, 17].

Figure 2: Mapping between objects and relations

- a restricted relation occurrence is a relation name subscripted with an integer and restricted by a selection expression. The subscript distinguishes different occurrences of the same relation in the query.

- a join is a connection join, an equijoin, or a general join.

That is, $\qquad\qquad JS = \{\sigma_{f_i} R_i \bowtie \sigma_{f_j} S_j | \bowtie \in \{ CJ, EJ, GJ \}\}$
where R_i and S_j denote the i-th and j-th occurrence of a relation R and S respectively, f_i and f_j are selection condition expressions such that $\text{Attr}(f_i) \subseteq R$ and $\text{Attr}(f_j) \subseteq S$, respectively[6], and CJ, EJ, GJ denote a connection join, equijoin, and general join, respectively.

2. AMF (attribute mapping function) is a one-to-one mapping between object attributes and relation attributes. That is,
$$\text{AMF: } S_o \xleftrightarrow{1:1} S_r$$
where $S_o = \{\text{Ochain}(O, s_0) | s_0 \in \text{Attr}(O)\}$ and $S_r = \{R_i.A | R \text{ is a relation name} \wedge A \subseteq \text{Attr}(R)\}$. $\text{Ochain}(O, s_0)$ was defined in Definition 2.

3. PD (pivot description) is a pair (PMF, PS) where

    (a) PS (pivot set) is the set of pivots. A pivot is either a *base pivot* or an *abstract pivot*. A base pivot is a relation occurrence whose key is ma... d to the id of an element of Oset (see Definition 1). An abstract pivot is defined by an ordered pair $\langle r_b, PJS \rangle$ where r_b is a base pivot and PJS (pivot join set) is a subset of join set needed to define a virtual relation occurrence. A PJS must satisfy the following properties.

        - PJS $\subseteq$ JS.
        - All relation occurrences in PJS are connected by inner joins.

---

[6] $\text{Attr}(\Phi)$ denotes the set of attributes appearing in Φ where Φ can be an expression, relation schema, object type, etc.

153

- A PJS contains one and only one base pivot (r_b) whose key is used as the key of the virtual relation occurrence.

(b) PMF (pivot mapping function) is a one-to-one mapping between PS and Oset. For every element of Oset, there exists one and only one pivot whose key is mapped to the id of the element.

As mentioned in Section 2, we associate value-oriented object id's with an object and its complex subobjects. These id's are invisible in the type definition and their mappings to relation attributes are not explicitly specified in the AMF. Rather these mappings are deduced from the information stored in PD by the following algorithm.

Algorithm 1 (Mapping object id's to pivots)
Input: Object type (O), Pivot description (PD)
Output: Mappings between object id's and relation attributes
Procedure:
    For each $p \in$ PS begin
        If p is a base pivot
            then add Ochain(O, PMF(p)).id $\leftrightarrow$ p.Key(p) to AMF.
        else /* p is an abstract pivot */ begin
            Find the base pivot r_b of p.
            Add Ochain(O, PMF(p)).id $\leftrightarrow$ r_b.Key(r_b) to AMF.
        end.
    end.

As a special case of defining AMF, if an object attribute s_0 is mapped to a relation attribute A which defines a *connection join* from $\sigma_{f_i} R_i$ to $\sigma_{f_j} S_j$, then s_0 can be mapped to either $R_i.A$ or $S_j.A$. Our mechanism retrieves the same set of object instances in either case. We choose $R_i.A$ in the following example.

Example 3 The query for instantiating the *Programmer* type shown in Example 1 is as follows.

1. JS ={ Engineer$_1$ (works-on) $\sigma_{job = ``*programming*"}$ Proj-Assign$_1$, Engineer$_1$ (eng-is-a) Emp$_1$, Emp$_1$ (works-for) Dept$_1$, Dept$_1$ (dept-is-a) Division$_1$, Proj-Assign$_1$ (worked-by) Project$_1$, Project$_1$ (has-title) Proj-Title$_1$, Project$_1$ (funded-by) Sponsor$_1$, Project$_1$ (led-by) Emp$_2$ }

2. AMF ={ Programmer.name $\leftrightarrow$ Emp$_1$.name, Programmer.dept $\leftrightarrow$ Emp$_1$.dept, Programmer.salary $\leftrightarrow$ Emp$_1$.salary, Programmer.manager $\leftrightarrow$ Division$_1$.manager, Programmer.Project.title $\leftrightarrow$ Proj-Title$_1$.title, Programmer.Project.sponsor $\leftrightarrow$ Sponsor$_1$.name, Programmer.Project.leader $\leftrightarrow$ Emp$_2$.name, Programmer.Project.dept $\leftrightarrow$ Project$_1$.dept, Programmer.Project.task $\leftrightarrow$ Proj-Assign$_1$.task }

3. PS = { Programmer$_1$, Project$_1$ }
   where *Project$_1$* is a base pivot and *Programmer$_1$* is an abstract pivot defined by
   $\langle$ Engineer$_1$, {Engineer$_1$ (works-on) $\sigma_{job = ``*programming*"}$ Proj-Assign$_1$ } $\rangle$.

4. PMF = { Programmer $\leftrightarrow$ Programmer$_1$, Project $\leftrightarrow$ Project$_1$ }

From PS and PMF, we can deduce, using Algorithm 1,
    { Programmer.id $\leftrightarrow$ Engineer$_1$.ssn, Programmer.project.id $\leftrightarrow$ Project$_1$.proj# }.
These are added to the AMF. □

4.2 Query Graph

A query graph is constructed from the join set (JS) part of a query.

Definition 5 (Query Graph) A query graph (QG) is a directed connected graph (V, E) where

- $V(QG)$ denotes a set of vertices. Each vertex $v \in V(QG)$ is represented by a triplet (π, r, f) and represents a node r labeled with f and π, where r is a relation occurrence, f is a selection condition expression on r, and π is the set of attributes that are projected from r.

- $E(QG)$ denotes a set of directed labeled edges. Each edge $e \in E(QG)$ is represented by an ordered pair $\langle v_i, v_j \rangle$, and represents a set of joins denoted by $\text{Jset}(e)$. Each edge is labeled as one or a conjunction of the followings:

    1. A connection name for a connection join.
    2. A join predicate, i.e., $attr_1 \; \theta \; attr_2$, for an equijon or a general join, where $\theta \in \{=, \neq, >, \geq, <, \leq\}$.

Frequently we say 'a *join* between two *nodes* v_1 and v_2', the precise definition of which is

$$v_1 \bowtie v_2 \equiv \Pi_{\pi_1 \cup \pi_2}(\sigma_{f_1} r_1 \bowtie \sigma_{f_2} r_2)$$

Example 4 The query graphs of type *Programmer* and type *InternalSponsor* are shown in Figure 3. The definition of type *InternalSponsor* is as follows.

```
Type InternalSponsor /* an internal division sponsoring a project */
[ name: string,
  project: {[ title: string, fund: string,
              period: string null-allowed ]} null-allowed ]
```

In Figure 3, nodes surrounded by double lines are base pivots, and those surrounded by dotted line are abstract pivots. The + labels, 'I/O' types on edges and '(attr $\neq$ null)' on nodes are not relevant here but shown for later use. Note Example b includes a *general* type of edge. □

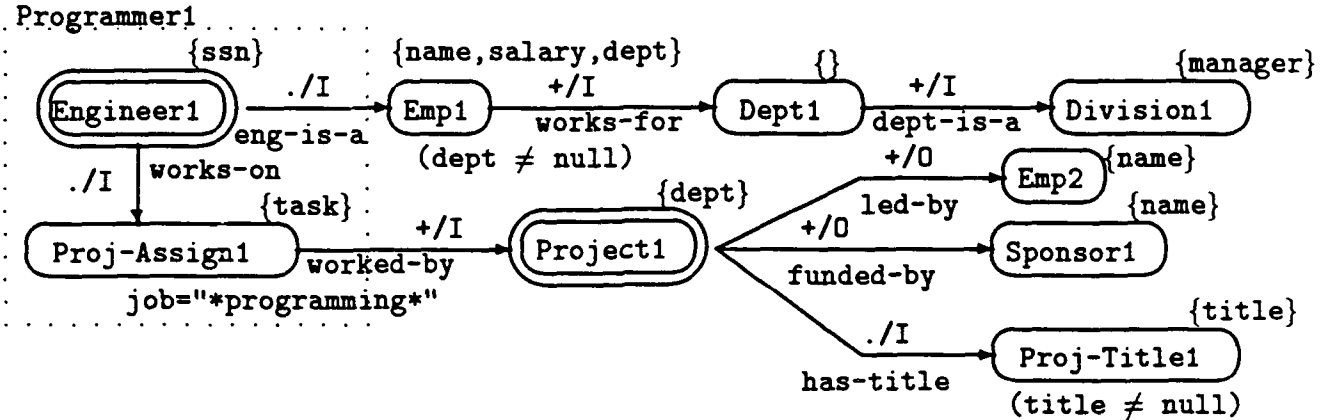## 4.3 Eliminating (Directed) Cycles from a Query Graph

A query graph containing cycles can be transformed to an equivalent one without cycles. We discuss *undirected cycles* first and apply the result to (directed) cycles.

Definition 6 (Undirected cycle) An undirected cycle in a query graph represents a cyclic join expression for retrieving the instances that satisfy *all* join expressions ($\bowtie_{ij}$'s) in $v_1 \bowtie_{12} v_2 \bowtie_{23} \cdots v_{n-1} \bowtie_{n-1,n} v_n \bowtie_{n1} v_1$. That is, it retrieves
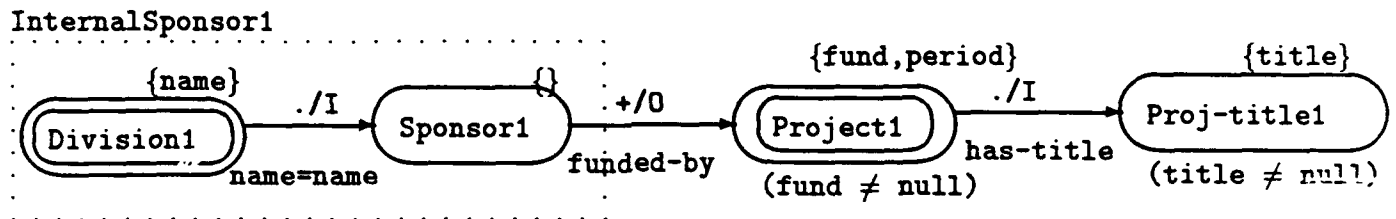
$$\{(t_1.\pi_1, t_2.\pi_2, \cdots, t_n.\pi_n) | \forall i \in [1, 2, \cdots, n-1](t_i \in \sigma_{f_i} r_i \wedge t_i \bowtie_{i,i+1} t_{i+1}) \wedge t_n \bowtie_{n1} t_1\}$$

Note, since we are using relation *occurrences* instead of relations themselves in our query model, 'cycles' are defined at the relation *instance* level, not at the schema level. Therefore, from Definition 6, we can conclude that all the edges of an undirected cycle in a query graph must be evaluated by inner joins.

Example 5 Figure 4 shows a cycle in the query graph of type *DBDProjLeader* with the semantics of 'An exmployee of the database department leading projects that are administered by the database department', whose type definition is as follows.

155

a. Programmer

b. InternalSponsor

Figure 3: Query graphs of type Programmer and InternalSponsor

```
Type DBDProjLeader
[ name: string, salary: integer null-allowed, manager: Employee,
  project: {[ title: string null-allowed ]} ]
```

□

We observed, but have not proved, that an undirected cycle in a query graph defines an abstract pivot. An important property of an abstract pivot node is that all its edges are *inner join* edges. Note, if we were to perform an outer join for Emp_1 (led-by^{-1}) $Project_1$ for example, then we retrieve employees not necessarily leading a project, thus violating the above definition of *DBDProjLeader*.

Sometimes a query graph may have (directed) cycles in it. Since all edges in a cycle are evaluated by inner joins and inner joins are commutative, we can eliminate these cycles by reversing the direction of an edge in a cycle without affecting the query result. Therefore, for further discussions, we assume a query graph is *acyclic* without loss of generality.

5 Prescribing Inner/Outer Joins for Edges

In this section we develop a mechanism for deciding whether to perform inner or outer joins for evaluating the joins in a query graph. First we describe an algorithm for mapping null-allowed and null-forbidden options on object attributes to +edges and −nodes, respectively. Then we discuss the join cardinality constraint rigorously and derive the the decision rule for inner/outer join. Finally, we describe an edge typing algorithm.
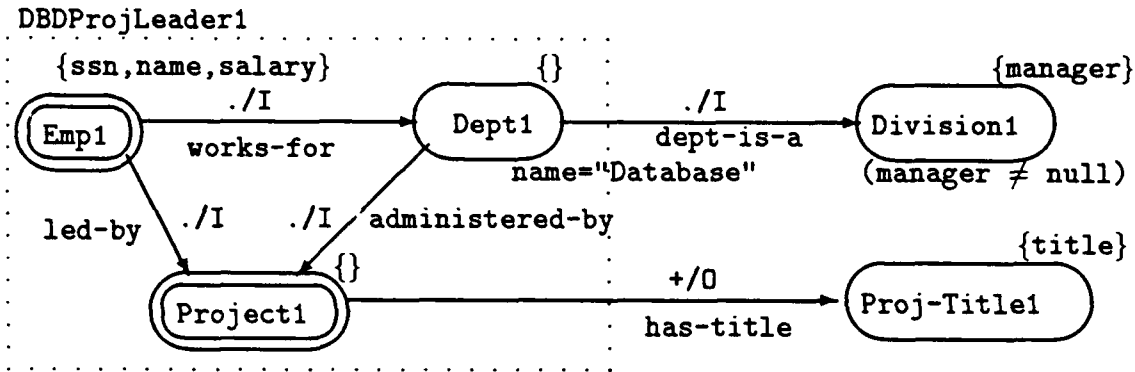
Figure 4: Query graph of DBDProjLeader

Our strategy is to first designate +edges without regard to the join cardinality constraints, and then reduce the number to a minimum by converting as many +edges as possible to inner joins. The rationale for this is that queries with inner joins are more efficient to process than those with outer joins.

5.1 Mapping Null-allowed/forbidden Options to +Edges/−Nodes

In our model, an attribute with a null-allowed option has the following semantics.

Definition 7 (Semantics of null-allowed/forbidden option) If an attribute s_0 of an object type O has a null-allowed/forbidden option then, given the id of O_n in $Ochain(O, s_0) \equiv O_0.O_1. \cdots .O_n.s_0$,

- s_0 is allowed/forbidden to be null if s_0 is a simple attribute.

- s_0.id is allowed/forbidden to be null if s_0 is a complex attribute.

Example 6 Given the *Programmer* type of Example 1, having a null-allowed/forbidden option on *manager* is interpreted as 'given a *Programmer.id*, *manager* is allowed/forbidden to be null' because *manager* is a simple attribute. On the other hand, having a null-allowed/forbidden option on *project* is interpreted as 'given a *Programmer. id*, *project.id* is allowed/forbidden to be null' because *project* is a complex attribute. □

Provided with this semantics, the algorithm for mapping a null-allowed/forbidden option on s_0 to +edges/−nodes is given below. We need the AMF and PD parts of a query here. Note the JS part was used for constructing a query graph.

We define a concept of *no-null relation attribute* first.

Definition 8 (no-null relation attribute) An attribute A of a relation R is called a no-null relation attribute if and only is it satisfies one or more of the following conditions.

- $A \in Key(R)$, i.e., A is a key attribute.

- A is prohibited from having a null value by schema definition.

- A is prohibited from having a null value by structural cardinality constraint, i.e. A is an attribute of a join which has an entry t in the connection catalog and $t.m_{12} > 0$.

That is, A cannot have a null value by these semantic constraints.

Algorithm 2 (Mapping null-allowed/forbidden options to +edges/−nodes)
Input: query graph (QG), query (Q), object type (O)
Output: a query graph (QG+) with +edges and −nodes
Procedure: For each object attribute (s_0) of type O begin

1. Find $\text{Ochain}(O, s_0) \equiv O_0.O_1. \cdots .O_n.s_0 \equiv \Omega_{0,n}.s_0$.

2. $r_p.\text{Key}(r_p) := \text{AMF}(\Omega_{0,n}.id)$.
   If s_0 is a simple attribute then $r_s.A := \text{AMF}(\Omega_{0,n}.s_0)$
   else /* If s_0 is a complex attribute */ $r_s.\text{Key}(r_s) = \text{AMF}(\Omega_{0,n}.s_0.id)$.
   If s_0 is null-allowed then go to 4.

3. If s_0 is a simple attribute and A is not a no-null relation attribute
   then $f_s := f_s \wedge (A \neq \text{null})$. Stop. /* If s_0 is a complex attribute then $A = \text{Key}(r_s)$ which is a no-null relation attribute. */

4. If s_0 is a simple attribute and at least one element of $\{\text{AMF}^{-1}(a_i)|a_i \in \pi_s\}$ does *not* have a null-allowed option then stop.

5. Find all directed paths
$$\Psi \equiv \{p(v_p, v_s)|v_p \equiv (\pi_p, r_p, j_p), v_s \equiv (\pi_s, r_s, f_s), A \subseteq \pi_s\}$$
   where $p(v_p, v_s)$ denotes a path, i.e., the set of nodes, from v_p to v_s.

6. For each $p(v_p, v_s) \in \Psi$ begin

   (a) If $p(v_p, v_s)$ contains only one node /* No join */ then stop.

   (b) Starting from v_s, traverse $p(v_p, v_s)$ in reverse direction until we find the first node v_q whose $\pi_q \neq \{\}$. If not found then $v_q := v_p$.

   (c) Label all edges of $p(v_q, v_s)$ as +edges.

   (d) If $\text{PMF}(O_n)$ is an abstract pivot then begin
       i. Find its $\text{PJS} \equiv \text{Second}(\text{PMF}(O_n))$. /* *Second* returns the second element. */
       ii. If $\neg(\langle v_q, \text{Succ}(v_q)\rangle \in \text{PJS})$ then stop. /* *Succ* returns the successor node. */
       iii. Starting from v_q, traverse $p(v_q, v_s)$ in forward direction until we find the first node v'_q such that $\neg(\langle v'_q, \text{Succ}(v'_q)\rangle \in \text{PJS})$. If not found then $v'_q := v_s$.
       iv. Remove + labels from all edges of $p(v_q, v'_q)$.
       end

   end

end.

That is, first identify the base pivot (r_p) of O_n and the relation occurrrence (r_s, called *target node*) providing the instances of s_0 in Step 1 and Step 2. Then in Step 3, if s_0 has no null-allowed option and it is mapped to a relation attribute (A) which may have null values, then restrict the target node with '$A \neq \text{null}$' and stop. Note Step 3 does not apply to a complex attribute because a complex attribute is always mapped to a key attribute of a relation. A key attribute is a no-null relation attribute by Defintion 8. On the other hand, if s_0 has a null-allowed option, check in Step 4 if it is mapped to the relation attribute of a node (v_i) whose projected attributes (π_i) are *all*

mapped to null-allowed object attributes. If so, find all paths from the base pivot (r_p) to the target node (r_s) in Step 5. Then, in Step 6, find a subpath of each path ($p(v_q, v_s)$) such that all nodes (v_j) between v_q and v_s have empty projection sets (π_j's), and label the edges with +. In case the pivot is an abstract pivot (Step 6-d), all edges in the abstract pivot must be inner join edges by definition and, therefore, their + labels are removed.

Example 7 The +edges of the *Programmer* and *InternalSponsor* types are shown in Figure 3. Illustrating this for the *Programmer* example, attributes *name, dept, project.title, project.dept* do not have null-allowed options while *salary, manager, project, project.sponsor, project.leader, project.task* do (see Example 1). First, pivots are AMF(Programmer.id) = Engineer$_1$.ssn and AMF(Programmer.project.id) = Project$_1$.proj# (See Example 3) in Line 1 and Line 2.

Those without null-allowed options are handled in Line 3. We assume, for this example, *name* is prohibited from having a null value by the database schema and, therfore not add 'name $\neq$ null' to the Emp$_1$ node. For *dept*, we add 'dept $\neq$ null' to Emp$_1$ node. Likewise, we add 'title $\neq$ null' to Proj-Title$_1$ node. *project.dept* cannot have a null value because it defines a connection *admin-by* and its minimum cardinality constraint is greater than 0 (see Example 2). Therefore we do *not* add 'dept $\neq$ null' to Project$_1$ node.

Those with null-allowed options are handled in Lines 4, 5, and 6. For the simple attribute *manager*, AMF(Programmer.manager) = Division$_1$.manager. Since *manager* is the only one attribute projected from the Division$_1$ node, it does *not* stop at Line 4. We find a directed path { Emp$_1$, Dept$_1$, Division$_1$ } in Line 5. The edges on its subpath { Emp$_1$, Dept$_1$, Division$_1$ } are labeled + in Lines 6-b and 6-c. PMF(Programmer) = Programmer$_1$ is an abstract pivot (see Example 3). Since no node on the subpath is in the abstract pivot, no + label is removed in line 6-d. For *salary*, it stops at Line 4 because *salary* is an attribute of Emp$_1$ node and the dept attribute of the node is mapped to an object attribute with no null-allowed option. For the complex attribute *project*, AMF(Programmer.project.id) = Project$_1$.proj# in Lines 1 and 2. Line 4 does not apply. In Line 5, we find a path { Engineer$_1$, Proj-Assign$_1$, Project$_1$ }. In Line 6-b, we find a subpath { Proj-Assign$_1$, Project$_1$ }. Its edge is labeled + and not removed in Line 6-d because the nodes are not in the same abstract pivot Programmer$_1$.

The mapping for other attributes can be verified in the same way. $\square$

5.2 Join Cardinality Constraint

As mentioned, join cardinality constraint (JCC) is a concept generalized from the connection cardinality constraint of a structural model. It turns out the connection catalog (CC) is the main source of information for determining the JCC of a join.

Definition 9 (Join cardinality constraint) Given a join from a node v_1 to another node v_2, the *join cardinality constraint* is defined as the pair [min, max]. The value of min is the minimum possible number of matching tuples of v_2 for each tuple of v_1.

The rule for determining the value of min is as follows.

Rule 1 (The min of a join) Given a join from a node $v_1 \equiv (\pi_1, r_1, f_1)$ to another node $v_2 \equiv (\pi_2, r_2, f_2)$, let R and S be the relation names of r_1 and r_2, respectively, and $X \subseteq$ Attr(R) and $Y \subseteq$ Attr(S) be the sets of join attributes.

1   If $f_2 \neq$ empty then min := 0
2   else if an entry t exists in the CC
3   then if t.From-Rel = R

4   then if t.type = 'reference' and $t.m_{12} = 0$ and
                  f_2 contains a predicate '$X \neq$ [null, null, $\cdots$, null]'
5   then min := 1
6   else min := $t.m_{12}$
7   else /* t.From-Rel = S */ min := $t.m_{21}$
8   else min := 0.

That is, if v_2 has been restricted by a non-empty selection condition, we cannot guarantee the existence of matching tuples whatever the constraints from the CC may be and, therefore, min = 0 (Line 1). Otherwise min is read from the connection catalog (CC) depending on the direction of the join (m_{12} or m_{21}) (Line 2 - 7). As an exception (Line 4 - 5), min becomes 1 if the condition of Line 4 is satisfied. A reference connection from v_1 to v_2 may have null values of join attributes (X) in v_1. This causes the default m_{12} to be 0. But, if these null values are removed by the selection condition $X \neq$ [null, null, $\cdots$, null], then X cannot have a null value and, therefore min := 1. The selection condition may not be a part of the query but have been added by mapping a null-forbidden option to a $-$node.

Example 8 The min of $Dept_1$ (works-for) Emp_1 is 1 because Emp_1 is not restricted and $m_{21} = 1$ is read from the CC shown in Example 2 (Line 7). On the other hand, the min of $Dept_1$ (works-for) $\sigma_{salary > 50,000} EMP_1$ is 0 because EMP_1 has been restricted (Line 1). The min of $Dept_1$ (zip = zip) Emp_1 is 0 because there is no entry decribing this non-connection join in the CC (Line 8). $\Box$

As we have seen from Example 3, a single join composes an edge in most cases. However, to be complete, we have to consider cases in which a single edge represents a set of joins, i.e., a conjunctive join. In these cases, the min's are determined by Rule 1 respectively and combined into a single value (min_C) according to the following theorem.

Theorem 1 (Combined min of an edge) Given an edge $e \equiv \langle v_1, v_2 \rangle$ which represents a set of p joins whose min's are a_i, $i = 1, 2, \cdots, p$, the combined min of e, denoted by min_C, is computed as follows.

$$\text{min}_C = \text{MAX} \left(\sum_{j=1}^{p} a_j - (p-1)k, 0 \right) \tag{2}$$

where k is the cardinality of v_2 and MAX is a function returning the maximum value of its arguments.

The proof of this theorem appears in Appendix A.

Example 9 Given a conjunctive join expression $Dept_1$ (works-for) $\wedge$ (zip = zip) Emp_1, where (works-for) is a connection join and (zip = zip) is an equijoin, assuming $k = 40$,
$$\text{min}_C = \text{MAX}(1 + 0 - (2-1)40, 0) = \text{MAX}(-39, 0) = 0.$$
Note the min of $Dept_1$ (zip = zip) Emp_1 is 0 because CC does not have an entry for it. To show an example of selecting the first term of MAX, let's imagine the minimum cardinality of $Dept_1$ (works-for) Emp_1 is 30 in the CC and the same for $Dept_1$ (zip = zip) Emp_1[7]. Then, $\text{min}_C = \text{MAX}(30 + 30 - (2-1)40, 0) = \text{MAX}(20, 0) = 20.$ $\Box$

From now on, 'min' denotes 'min_C' unless stated otherwise.

---

[7]For example, this can be derived from a domain constraint like "All employees must live in the same zip code area as their departments."

5.3 'I/O' Edge Classification

Provided with Algorithm 2 and Theorem 1 with Rule 1, we are ready to derive the mechanism for classifying edges into type 'I/O' for inner/outer join.

We discussed min for two nodes only ($\langle v_1, v_2 \rangle$) so far, not considering the effect of joins along edges descending from v_2. To consider this effect, we introduce the concepts of an *effectively restricted node* and *effective min*.

Definition 10 (Effectively restricted node) A node $v_i \in V(QG)$ is called an effectively restricted node if and only if there exists at least one edge $\langle v_i, v_j \rangle \in E(QG)$ that is evaluated by an inner join, and $min_{ij} = 0$ or v_j is an effectively restricted node. In particular we say, even if min > 0 according to Rule 1, the *effective min* becomes 0 if v_j is an effectively restricted node.

Note a node is *acutally* restricted if its selection condition expression (f) is not empty. This effect has already been considered in Rule 1 to derive min = 0 and is excluded from the definition of *effective* restriction.

This definition is recursive and implies the *inheritance of effective restriction* over inner join edges as follows.

Theorem 2 (Inheritance of effective restriction) Given an edge $\langle v_i, v_j \rangle \in E(QG)$, if v_j has no child and $min_{ij} = 0$, then v_i and its ancestors are all effectively restricted as long as there is an 'inner join path' from v_i to those ancestors.

Proof 1 Certainly v_i is effectively restricted according to Definition 10. Consider a chain of nodes from one of v_i's ancestors (v_a) to v_i, $\langle v_a, \cdots, v_i \rangle$, such that all edges on the chain are evaluated by inner joins. Then, by applying Definition 10 repeatedly, Pred(v_i), Pred(Pred(v_i)), $\cdots$, Pred(Pred($\cdots (v_i)$))) $\equiv v_a$ all become effectively restricted. Q.E.D.

Before stating the rule of inner/outer join, we introduce the *principle of minimial outer joins*. As mentioned, an outer join is prescribed by a +edge. However, if v_2 is not an effectively restricted node and min > 0, then the effective min > 0 and, therefore, the results of an inner join and an outer join are the same. In this case, the prescription of outer join is overridden and the +edge is evaluated by an inner join.

Rule 2 (Inner/Outer join) Given an edge $e \equiv \langle v_1, v_2 \rangle$ of a query graph QG and its min, the edge is evaluated by either an inner join or an outer join according to the following rule.

> If min > 0 and v_2 is *not* effectively restricted, i.e., effective min > 0,
> then use an inner join
> else if c is a +edge then use an outer join
> > else use an inner join.

Based on these discussions, we give here an algorithm for classifying edges into type 'I' for inner joins and 'O' for outer joins. We use Rule 2 as a subprocedure returning 'I' or 'O'. QG+ is assumed to be an *acyclic* graph.

Algorithm 3 (Edge typing)
Input: Query graph (QG+) with +edges and −nodes, Connection catalog (CC), Object type (O)
Output: Query graph (QG^t) with edges typed into 'I/O' for inner/outer join

Procedure Main:
1a   If PMF(O) is a base pivot then $v := $ PMF(O)

1b   else $v := \mathrm{First}(\mathrm{PMF}(O))$. /* Find the base pivot */
1c   Call Edge-Type(v).

Procedure Edge-Type(v_i: node)
2a $\Psi := \{v_j | \langle v_i, v_j \rangle \in E(\mathrm{QG})\}$. /* Children of v_i */
2b   For each $v_j \in \Psi$ begin
2c   if v_j has no child /* Therefore, not effectively restricted */ then begin
2d $L := \mathrm{Rule}\ 2(v_i, v_j)$. /* Apply Rule 2 to $\langle v_i, v_j \rangle$. */
2e   If $L = $ 'I' and $\min_{ij} = 0$ then designate v_i as 'effectively restricted'.
2f   end else begin
2g   Edge-Type(v_j).
2h $L := \mathrm{Rule}\ 2(v_i, v_j)$.
2i   If $L = $ 'I' and v_j is 'effectively restricted'
2j   then designate v_i as 'effectively restricted'.
2k   end
2l   end.

Example 10 Edges typed with 'I/O' are shown in Figure 3. Starting from the Engineer$_1$ node, Edge-Type is called recursively until we hit the Division$_1$ node which has no child. Then in Lines 2c - 2f, Rule 2(Dept$_1$, Division$_1$) returns 'I' because min $= 1 > 0$ although the edge was labeled $+$ (see Example 7). Dept$_1$ is *not* designated as 'effectively restricted' because min > 0 . Next in Lines 2f - 2k, Rule 2(Emp$_1$, Dept$_1$) returns 'I' because min > 0 (Emp$_1$ has been restricted with 'dept $\neq$ null' in Example 7. Therefore min $= 1$ according to Lines 4 - 5 of Rule 1.) and Dept$_1$ is not effectively restricted although the edge was labeled $+$. Emp$_1$ is *not* designated as 'effectively restricted' because Dept$_1$ is not 'effectively restricted'.

We leave it up to the reader to verify the rest. $\square$

6 Conclusion

In this paper, we developed a mechanism for automatically prescribing inner/outer joins for the joins of a complex query used to instantiate objects from a relational database. The major criteria for deciding on inner join or outer join are null-allowed options on object attributes and minimum join cardinalities derived from the connection cardinality constraints of a structural data model. We began with describing the system model in the order of object type model, data model, and query model. The null-allowed options of object types provide the semantics for outer joins, making it possible to prescribe outer joins for a maximal subset of edges in a query graph. Then the join cardinality constraints provide the semantics of effective min > 0, making it possible to override the prescriptions with inner joins for more efficient processing.

The overall algorithm developed in this paper can be summarized as follows. First, compile the stored O into Oset and $\{\mathrm{Ochain}(O, s_0) | s_0 \in \mathrm{Attr}(O)\}$. Second, construct the query graph (QG) of the query Q according to Definition 5 and map object id's to pivots using Algorithm 1. Third, map null-allowed and null-forbidden options on object attributes of O to +edges and −nodes of the QG, respectively, using Algorithm 2, producing QG+. Fourth, derive the minimum join cardinality constraint for each edge of the QG+. Finally, type the edges of the QG+ into 'I' for inner join and 'O' for outer join using Algorithm 3, producing QG^t.

APPENDIX

A Proof of Theorem 1

Let $\{J_1, J_2, \cdots, J_p\}$ denote the set of p joins of the edge e, and $m_i, i = 1, 2, \cdots, p$ be the combined min of the first i joins. Given a tuple of v_1, let $\phi_1, \phi_2, \cdots, \phi_i$ denote the sets of matching tuples in v_2 for $J_i, J_2, \cdots, J_i$, respectively, and Φ_i be their intersection, i.e., $\Phi_i \equiv \phi_1 \cap \phi_2 \cap \cdots \cap \phi_i$.

1. Base case ($p = 2$):
$$m_2 = \begin{cases} a_1 + a_2 - k & \text{if } (a_1 + a_2) > k \\ 0 & \text{otherwise} \end{cases}$$

   $m_2 = 0$ holds true if and only if $\Phi_2 \equiv \phi_1 \cap \phi_2$ is empty. On the other hand, Φ_2 cannot be empty if $a_1 + a_2 > k$, and it can be easily seen the miminum possible cardinality of Φ_2 in this case occurs when ϕ_1 and ϕ_2 overlaps minimum and the number of minimum-overlapping tuples is $a_2 - (k - a_1) = a_1 + a_2 - k$.

2. Induction step: By the same reasoning as above for $i \geq 2$, having obtained m_{i-1}, m_i is obtained as
$$m_i = \begin{cases} m_{i-1} + a_i - k & \text{if } (m_{i-1} + a_i) > k \\ 0 & \text{otherwise} \end{cases}$$

3. Conclusion: From 1 and 2, we can conclude
$$m_p = \begin{cases} \sum_{j=1}^{p} a_j - (p-1)k & \text{if } \sum_{j=1}^{p} a_j > (p-1)k \\ 0 & \text{otherwise} \end{cases}$$

   Rewriting this,
$$\min_C \equiv m_p = \text{MAX} \left(\sum_{j=1}^{p} a_j - (p-1)k, 0 \right)$$

   Q.E.D.

Acknowledgement

References

[1] Maier, D., Stein, J., "Development of an Object-Oriented DBMS," Proceesgins of OOPSLA, September 1986, pp. 472 - 482.

[2] Ford, S., et al., "Zeitgeist: Database Support for Object-Oriented Programming," International Workshop on Object-Oriented Database Systems, 1988, pp. 23 - 42.

[3] Kim, W., Chou, N., Garza, J., "Integrating an Object-Oriented Programming System with a Database System," Proceesings of OOPSLA, September 1988, pp. 142 - 152.

[4] Fishman, D., et al., "Iris: An Object-Oriented Database Management System," ACM Trans. on Office Information Systems, Vol.5, No.1, January 1987, pp. 48 - 69.

[5] Stonebraker, M., Rowe, L., "The Design of POSTGRES," Proceedings of ACM SIGMOD, 1986, pp. 340 - 354.

[6] Wiederhold, G., "Views, Objects, and Datbases", IEEE Computer, December 1986, pp. 37-44.

[7] Cohen, B., " Views and Objects in OB1: A Prolog-based View-Object-Oriented Database", SRI, TR.PRRL-88-TR-005, March 1988.

[8] Barsalou, T., Wiederhold, G., "Knowledge-based Mapping of Relations into Objects," The International Journal of Artificial Intelligence in Engineering, Computational Mechanics Publ., UK, 1989.

[9] Date, C., "The Outer Join," Proceedings of the Second International Conference on Databases, Cambridge, Britain, September 1983.

[10] Finkelstein, S., "Common Expression Analysis in Database Applications," the Proceedings of ACM SIGMOD, 1982.

[11] Wiederhold, G. "Database Design (2nd ed.)," Chapter 7, Mc-Graw Hill, Inc., 1983.

[12] Hull, R., King, R., "Semantic Data Modelling: Survey, Applications, and Research Issues," ACM Computing Survey, Vol.19, No.3, September 1987, pp. 243.

[13] Wiederhold, G., Barsalou, T., Chaudhuri, S., "Managing Objects in a Relational Framework," Stanford Technical report CS-89-1245, Stanford University, January 1989.

[14] Bancilhon, F., et al., "The Design and Implementation of O_2, an Object-Oriented Database System," in 'Advances in Object-Oriented Database Systems,' Springer-Verlag, September 1988.

[15] El-Masri R., Wiederhold, G., "Properties of Relationships and their Representation," AFIPS Conference Proceedings, Vol. 49, 1980.

[16] Bancilhon, F., "Object-Oriented Database Systems," Invited lecture, 7th ACM SIGART-SIGMOD-SIGACT Symposium on Principles of Database Systems., Austin, Texas, March 1988.

[17] Maier, D., "Why Isn't There an Object-Oriented Data Model?," Proceedings of the IFIP 11th World Computer Congres, San Francisco, California, September 1989.