SRC TR 87-178

High Performance Engineering Information Systems

by

N. Roussopoulos, L. Mark,
T. Sellis and C. Faloutsos

HIGH PERFORMANCE ENGINEERING INFORMATION SYSTEMS[‡]

N. Roussopoulos, L. Mark, T. Sellis, and C. Faloutsos

Department of Computer Science
and Systems Research Center
University of Maryland
College Park, MD 20742

Abstract

We present a framework for developing High Performance Engineering Information Systems (EIS) based on **Incremental Computation Models**. These models utilize results of previous computations stored in **persistent cache structures** by merging them with new results obtained from computations performed on the incremental changes of the database. Because computation is performed on small increments rather than the whole database, incremental computation models achieve performances that are far beyond the reach of ordinary models. High performance EIS will be able to support engineering knowledge bases of significant size, explicit models of interdependent knowledge, optimization techniques for deductive queries, and efficient methods for organizing and accessing large numbers of engineering facts and rules. The persistent cache structures of the incremental models provide an innovative self–improving performance capability for computation intensive EIS.

[‡] This research was sponsored partially by the National Science Foundation under the grant CDR–85–00108 and by UMIACS.

1. Introduction

Traditionally **Database Management Systems (DBMS)** have been used in business applications to efficiently organize and store large amounts of data. The main thrust of database research has focused on designing data structures and algorithms so that operations, common in this environment, can be processed efficiently.

Recently, there has been considerable interest in providing a framework for information sharing and exchange in engineering environments. **Engineering Information Systems (EIS)** are the current focus of several government sponsored research projects [LINN86]. These efforts are trying to consolidate and integrate the engineering support environments that have been and are being developed to support planning, design automation, manufacturing, resource management, etc. Along this direction, there has already been some work in extending existing database management systems to accommodate engineering applications. In particular, relational DBMS's have been used in support of Computer Aided Design (CAD) [KATZ82,LORI83,GUTT84,CAMM84,BATO85], Computer Integrated Manufacturing (CIM) [KIMU82,KOCH84,MELK84,HARH87], and Artificial Intelligence and Expert Systems [KERS84,KERS86]. The main difference between the business applications and the ones mentioned above lies in the kind of information that the two types of applications are using. Business applications are mainly concerned with large volumes of *structured data* with well understood access patterns, while engineering applications usually involve a sophisticated *control mechanism* that handles structured and unstructured data with rather unpredictable access patterns. Therefore, a system of the second type should be able to support explicit representations of control information in addition to factual information.

Using a data manager with full capabilities offers the advantages of better data organization, simple user interface, integrity of data in multi–user environments and recovery from hardware or software crashes. Given these advantages, there have been various attempts to build systems that support non–traditional database applications over large volumes of data. In general, there are three different approaches that can be taken

- One can enhance a specific application system with a *specialized* data manager.

- One can *interface* a specific application to a general purpose DBMS.
- Finally, one can *extend* a general purpose data manager by enhancing it with more sophisticated capabilities.

The first approach suffers from two major disadvantages. First, considerable effort must be put into designing and building several modules that DBMS's already include (data definition and manipulation facilities, query processing algorithms, etc). Second, such specialized data managers are very narrow, in the sense that they cannot be easily modified to support applications other than the ones they were originally written for.

In the second approach there is a clean interface between a specialized application program and a general purpose DBMS. The DBMS acts as a server to the application program by supplying on demand the data that the latter requires. However, the major disadvantage of this approach lies in the difficulty to define exactly where the two systems must be interfaced. [ZANI84,SCIO84] provide good criticisms of this approach.

The most promising approach is the third one. In this one, data manipulation and control functions are integrated into a single system in a *homogeneous* way. The work of [MYLO80,SHIP81,ZANI83] in semantic data models, of [COPE84,DERR86] in the design of systems based on the object oriented data model and of [STON86,STON87] in the extension of INGRES to support expert systems applications are representative of this approach. The basic idea has been to create a simple system that gives the user the capability to *build on top* of a basic set of functions whatever constructs are required by his applications. Moreover, it has been assumed that minimal extensions to the relational model should be attempted. An example of such efforts have been **Engineering Databases** [LORI85,BATO85]. The assumption here is that complex engineering objects can be supported by normalizing them into relational structures connected via the concept of *surrogates*, and engineering rules for change propagation can be handled by a deductive mechanism. Another example is the work on **Deductive Databases** [GALL78] which provide basic support for expert systems applications. In a deductive database system both *rules* and *facts* are stored in the same system. In [IOAN84,DAYA85,ULLM85,ZANI85] several database systems enhanced with inference capabilities are proposed, each being a specific implementation of the above model of rules and facts.

None of above proposals however explores in detail all basic Engineering Information Systems requirements that need be supported by database systems. Moreover, engineering and deductive databases are severely handicapped by their performance. Little work has been done to improve the storing of and efficient access to large and complex engineering objects and rule bases. This specific issue of performance is the focus of our paper.

Section 2 briefly presents the basic Engineering Information Systems requirements. In Section 3 we show how these requirements can be handled by minimally extended relational database systems. In the same section we also discuss the various performance and optimization issues and suggest various solutions. Section 4 presents the implementation status of the project and, finally, we conclude in Section 5 with a summary.

2. Engineering Information Systems Requirements

The primary goal of an **Engineering Information System (EIS)** is to provide integrated support for the management and the engineering processes. The management processes are planning, monitoring, and evaluation. The engineering processes are design, development, testing, fabrication, and maintenance of engineering products for which form, fit and function is controlled. The management of the complex and continuously evolving databases consisting of data objects associated with management and engineering processes is the key concept in this framework and the integration of special purpose management and engineering tools with database technology will be the foundation of it.

An EIS must support **reuse, evolution** and **trace** of information. These are the most common practices in any engineering process and they constitute the foundation of technology evolution.

Reuse of designs and the acquired know–how is currently limited to small groups of people. Technology transfer among these groups is often so time consuming that it cannot catch–up with its own evolution. Dynamic models which can acquire new information and/or trace back to the development of the technology are needed. *The increasing complexity and size of the information that needs be managed by current and future EIS's mandates the use of both expert system and database technology.*

Control of evolving engineering information bases can use the deductive capabilities of expert systems. Systems like INTERNIST–1 [MILL82], R1 [McDE82], PROSPECTOR [DUDA78] and DENDRAL [LIND80] manipulate information in the order of a few hundred facts and rules. Scaling this up to accommodate an EIS with tens of thousands of rules and hundreds of thousands of facts is a non–trivial task. Organizing the information using the well defined principles of database systems is clearly not only an advantage, but a necessity.

However, one must first look in a systematic way at all the basic EIS requirements in order to get a good understanding of the kind of extensions that need be made to current database systems. This section provides such a discussion.

EIS's are different from other information systems in that they manage and control vast amounts of redundant and interdependent information and knowledge. Therefore, they need spe-

cialized management techniques. We have classified them into five main categories: representation of information, management of highly redundant information, management and control of dependent information, design–reuse and control methodologies, and management of interoperable environments.

2.1. Example of an Integrated EIS

We shall first introduce a simple example and use this as a medium for the subsequent discussion. The system is shown in Figure 1.

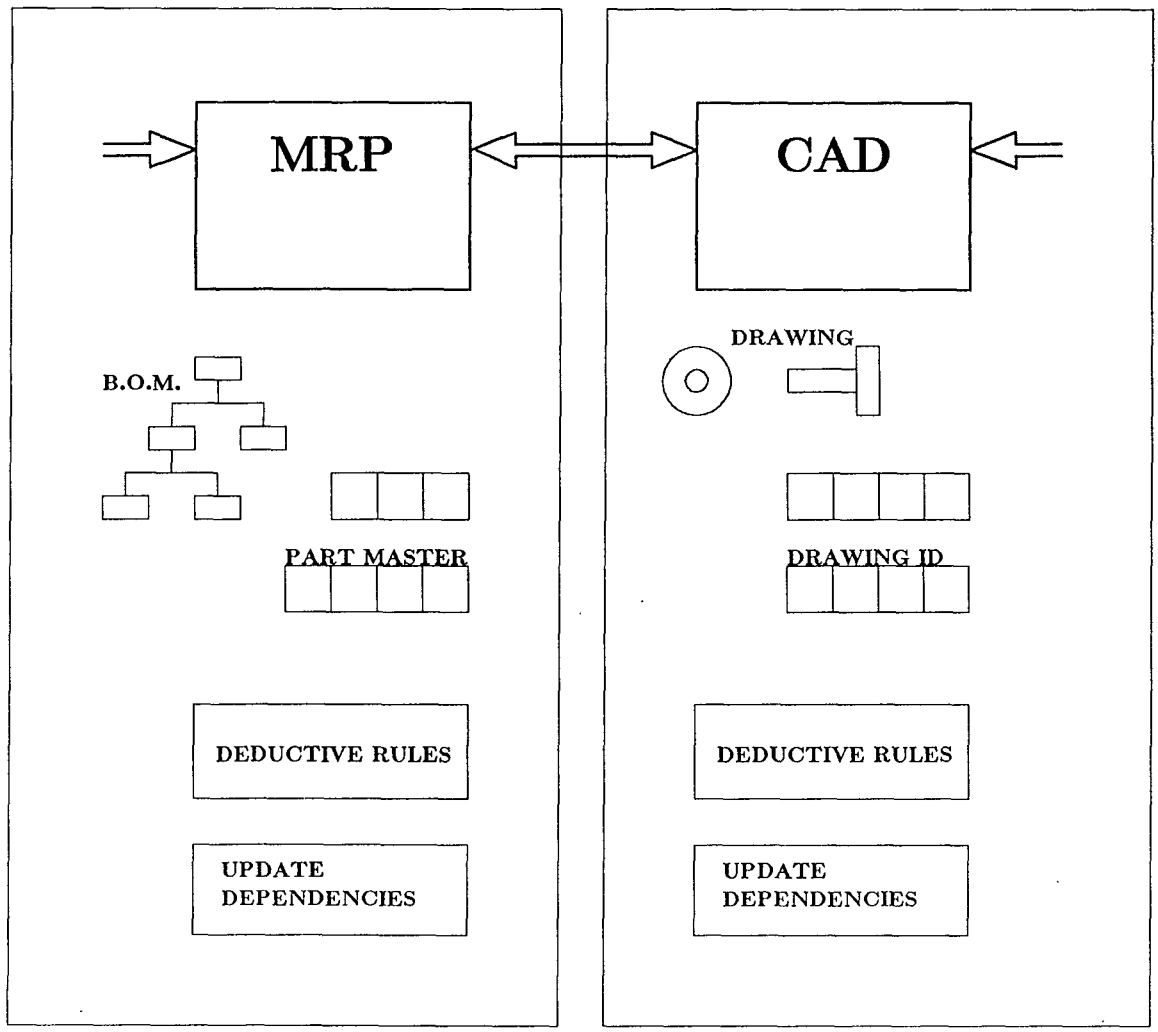

Figure 1: An Example EIS

The example illustrates a simplified model of an information system for Manufacturing and Resource Planning (MRP) integrated with an information system for Computer Aided Design (CAD). The MRP system contains a variety of information, in particular the Bill of Material (B.O.M.) describing part explosion and the Part Master record containing the individual part descriptions and versions. The CAD system contains designs and drawings in various versions. As we shall see later, there is a number of situations where deductive rules are needed for complex data and information retrieval from the two systems. There is a considerable amount of overlap and redundancy between the data stored and used by the two systems. The update dependencies of such redundant data systems require explicit representations to be used by the redundancy controller in maintaining consistency, in supporting data exchange and change propagation between the two systems.

2.2. Representation of Information

The information in an EIS can be classified with respect to the following three orthogonal dimensions:

- management or engineering related,
- process or product related, and
- description or instance related.

Management processes are activities like planning, monitoring, and control. Management products include plans, schedules, resource allocations, and results from the monitoring and control processes. Engineering processes are activities like design, manufacturing, and testing. Engineering products are designs, manufactured products, test results, etc. Knowledge related to both the description and actual executions of processes must be represented in an EIS, e.g. information about the planning strategy used by management must be represented in and enforced by the EIS, and information about each individual planning activity following this strategy must be recorded. Likewise, information related to both the description and the actual instances of products must be represented in the EIS, e.g. both the description of a manufactured product and the number of instances produced are relevant to engineers. There are several relationships between these classes of information. One example of such a relationship is that a specific manufacturing process consumes component products and produces manufactured products.

Another example of a relationship is that the result of a planning process may be used as the description of a manufacturing process. A unifying model of the information needed in an EIS is essential. We believe that such a model can be based on the above classification.

Only part of the information in an EIS can be represented and used by means of traditional database management system. Several management and engineering processes, such as planning, decision making, monitoring, and control are based on summarized and aggregated information deduced from the database. To represent and use Deductive Engineering Rules rules in an EIS, we have to extend database management systems with Expert System capabilities.

Capabilities are needed for

- representation of deductive engineering rules
- efficient retrieval of the rules by an inference engine
- learning based on previously made deductions.

2.3. Management of Highly Redundant Information

A variety of new data types are needed for representing descriptions of management and engineering processes and products. Many of these data types require specialized software for storage, display, and processing. Redundant multiple representations of the same information is necessary to support the multiplicity of tools and users. Another factor that significantly contributes to the redundancy of an EIS is the maintenance of large number of versions. Managing collections of complex objects is different than managing a no–redundant snapshot database. **Meta–information** is needed to describe and control the redundancy.

Maintenance of highly redundant information requires techniques that are based on **differential files**. These are logs of the changes made after a consistent state of the database and maintained separately from the main files. Differential files save storage but, most importantly, make version control and regeneration of previous versions very efficient.

2.4. Management and Control of Dependent Information

Update propagation of dependent information, library like check–in/out control and long transactions are distinct characteristics of engineering databases. Techniques for dealing with long lasting processes are different than those dealing with rapid transactions found in today's

data processing applications. The required concurrency and consistency protocols must account for and control engineering changes. Declarative and executable models are needed to capture **Update Dependencies** of objects. Update dependencies can be thought of as the information and control needed to maintain the integrity and the consistency of an evolving database.

2.5. Design Reuse and Control Methodologies

Unlike data processing environments which only capture a snapshot of the world, EIS databases must capture the whole history of the information from the design of engineering products to maintenance. The database is developed incrementally and is maintained for trace of information. It must support cooperative design based on the fundamental practice of **reuse** of existing designs and previously obtained know–how. To reuse a piece of design, a designer must be made aware of its existence and browse through its associated information.

In order to control new designs and other derived information, the EIS must support methodologies tailored to the application. EIS designers have to provide the EIS users with methodologies and guidelines for interacting with the EIS. Having explicit models for representing information and its interactions, is like having a programming language. One can write correct or erroneous programs. Existing engineering support methodologies must be extended to account for the large complexity of the future systems.

2.6. Interoperability of Environments

The environment of an EIS is naturally distributed among a large number of tools and special purpose workstations, such as schematic analyzers, graphics editors, simulators, database management systems, etc. In a typical EIS environment, most of the interactions will originate from a workstation or a tool tailored to some specific task. It is highly unlikely that all the functionality of special purpose processors and tools that currently exist in the market or those that will be developed in the near future can be provided by a single host. Instead, special purpose processing will continue to be done at tailored tools which will interact with an EIS. This requires an EIS–tool architecture that is partially distributed between workstations and mainframes but tightly controlled by an interface that explicitly manages their interactions: downloading of objects, bindings of downloaded objects, check–in and out, versions, etc.

The distribution of data and processing among workstations allows parallelism. To guarantee data consistency, updates that affect the EIS database must be transmitted to the EIS tool where they are validated for consistency. Special purpose protocols for derived and redundant data are necessary. Furthermore, the interface should keep track of how information from the EIS disseminates to the various tools, how updates done after the dissemination affect processing at the tools, and what updates must be propagated to them. Differential file techniques and incremental updating *must* be used to minimize data transmission and processing time.

3. Performance Issues in Large Scale EIS's

In this section we first suggest schemes whereby relational database systems can be used in the implementation of various engineering information systems features. Then, we turn to discuss a number of performance issues that arise.

3.1. Expert Database Systems

As mentioned in the previous section, engineering information systems usually involve an expert sub–system which is used to intelligently search over vast amounts of data. However, up to now, this is done assuming the fact that main memory is sufficient to hold all the necessary information. We exploit here an alternative that can be used in EIS, namely intelligent database systems or **Expert Database Systems (EDS)**. Although we focus on ideas mainly applicable to production rule based systems, semantic networks and frames could also be supported. The basic features discussed are data, rules, meta–knowledge and the inference mechanism (see [SELL87b] for more details). Throughout the discussion we will illustrate our proposals through the MRP/CAD system of section 2.

Engineering Information can be clearly stored using relations. For example, the realization of a predicate `bom(part,sub-part)` is done through a relation `BOM` (we will use all–capital names to distinguish DBMS entities) with two fields `PART` and `SUBPART` and specific domains for these two fields.

Engineering Rules can also be mapped to database entities. First, consider simple inference rules such as those used in deductive databases. For example, a rule may be used to define a `contains` predicate that describes a part hierarchy, i.e. which part contains another part, as follows

$$\texttt{contains(cover,part)} \leftarrow \texttt{bom(cover,X)} \wedge \texttt{bom(X,part)}$$

The above rule can be expressed using relational views in the following way (we use SQL to describe the view)

```
CREATE VIEW CONTAINS (COVER,PART)
    AS SELECT FIRST.PART, SECOND.SUBPART
       FROM   BOM FIRST, BOM SECOND
       WHERE  FIRST.SUBPART = SECOND.PART
```

The semantics are exactly the same, that is, they both define the `contains` entity in the same

way by examining the bom entity.

A different type of rules are general production rules of the form

IF <CONDITION> THEN <ACTION>

where ACTION is a general operation such as an insertion or deletion, not just a retrieval from the knowledge base as deductive rules assume. Hence, this kind of rules incorporates procedural semantics in EIS's. Notice that this type of rules corresponds to the idea of triggers or alerters [BUNE79] in database systems. They can be used to generally model such activities as well as protection and security mechanisms.

Rules can be modeled in a database system using the idea of **Update Dependencies**. The update dependency formalism was originally suggested in [MARK85] to support a wide variety of applications, such as walk–through guidance control systems, cause–effect systems, statistical information gathering, knowledge acquisition, database integrity enforcement, database view updates, policy enforcement, and production control. An update dependency has the following format

ON <OPERATION> PERFORM <ACTION_1,ACTION_2,...>

where OPERATION is an operation on the database that results to a series of actions ACTION_1, ACTION_2, etc. Using update dependencies one can implement production rules by simply defining the OPERATION and ACTION parts. As an example, we consider again the MRP/CAD system. Suppose there are two databases, the **CAD** and the **MRP** database with the following format respectively

CAD drawing_id

part#	description	u.o.m.	status	revision_no

MRP part_master

part#	description	lead_time	cost	status	revision_no

The following example illustrates a rule that calls for implied operations on the **MRP** database

when a new design is completed in the **CAD** database.

```
ON      [ complete_CAD(drawing_id_CAD(P,D,U,S,R))  ∧  ¬drawing_id_CAD(P,_,_,S,R) ]
PERFORM [ create_MRP(part_master_MRP(P,D,_,_,S,R)),
          assert(drawing_id_CAD(P,D,U,S,R)) ]

ON      [ complete_CAD(drawing_id_CAD(P,D,U,S,R))  ∧  drawing_id_CAD(P,_,_,S,R) ]
PERFORM [ write("Drawing already exists for",P) ]
```

The rule checks if a given drawing already exists in the database, and if not it adds it to the **CAD** database making the corresponding insertions in the **MRP** database as well.

Update dependencies are based on the logic programming formalism [KOWA74]. It can, therefore, be used to implement all known trigger mechanisms. However, because logic programming is found difficult by a large class of programmers, we have extended the formalism to include familiar control structures from ordinary programming languages such as *while*, *case*, and *repeat* statements [MARK86b].

A very significant part of a knowledge base lies in the **Meta–Knowledge**. This is an issue that has received very little attention in previous proposals for the implementation of expert database systems. Most of these proposals assume that meta–knowledge is hidden somehow in the various operations or the search algorithm. In this proposal we look for better and more complete means of describing meta–knowledge. Recent work in Self–Describing Databases and Meta–Data Management [MARK85,MARK86] is applicable to the management of meta–knowledge. A self–describing database system maintains a four level data description model. Each level is the *intension* of the level that it describes below it. It is also the *extension* of (or is being described by) the level above. This intension–extension model provides an active and integrated Data Dictionary System in which the meta–knowledge required to represent, change, and retrieve engineering rule bases is defined. A key issue will be to find an appropriate representation of the meta–knowledge. This issue is discussed in more detail in the following sub–sections.

Finally, we discuss briefly the implementation of an inference engine in an EDS. As illustrated above, both simple deductive rules and more general procedural rules can be supported; hence, the inference engine really maps to the query processing and data manipulation engines of the database system. Backward chaining is handled already through query modification for view

processing. For example, if one asks for the part containing bolt1, the database query will be

```
SELECT COVER
FROM   CONTAINS
WHERE  PART = "bolt1"
```

which is in turn changed using query modification to

```
SELECT FIRST.PART
FROM   BOM FIRST, BOM SECOND
WHERE  FIRST.SUBPART = SECOND.PART
AND    SECOND.SUBPART = "bolt1"
```

The query modification module needs to be extended to handle recursive view definitions [ULLM85,IOAN86] as well as multiple view definitions (to accommodate multiple definitions for the same rule), but conceptually the mechanism exists already in database systems. Forward chaining can be implemented based on the triggering mechanism realized by update dependencies. Hence, appropriate design and implementation of an EDS can take advantage of all existing database technology to support expert systems. However, the major question arising in such an environment will be *performance*. This issue is addressed in the rest of this section.

3.2. Incremental Computation Models

Expert Database Systems and Object Oriented Databases are severely handicapped by the performance of existing DBMS's. The database access and the real time response requirements are well beyond the capabilities of classical query optimization and buffer management techniques. Classical methods were based on static compilation of data processing access patterns. These patterns are very different from the very dynamic and specialized ones necessary to support a general inference mechanism that an EIS may employ. Adaptive access mechanisms for improving performance are necessary.

We propose the idea of *Incremental Learning* as a solution to the high cost of repetitive but unpredictable access patterns observed in Engineering Information Systems. Incremental learning is a concept that enables a system to avoid unnecessary search by remembering some of the computation search it did before. It allows the generation of ever improving systems. The foundation of Incremental Learning is a new class of **Incremental Computation Models (ICM)** [ROUS87b].

Let C be a computation performed at time t_1 in input I_1. An incremental computation model (ICM) performs C at time t_2 by merging the result obtained at time t_1 with the result obtained by performing C on the **differential** of the inputs ($dI=I_2-I_1$). An ICM can be employed in any distributive computation. The advantage of an ICM is clearly in performance because, at any time, the computation is only performed on the increments which are typically very small compared to the whole database, and get emptied after each use.

Previously obtained results are stored in **persistent cache structures** that become part of the database. These structures can be made very compact using the view indexing techniques proposed in [ROUS82a,b]. These structures contain simple pointers to the data records obtained by the ICM. The compactness of the cache structures allows them not only to be loaded in main memory quickly, but also to be maintained sorted for **optimal caching**. An access method based on the incremental approach and its corresponding algorithms are analyzed in [ROUS87a].

The incremental approach can be extended not only to identical computations but to those that are derivable from others. We can perform a derivable computation using the differential between the two computations $dC=C'-C$. Incrementally derivable computations are applicable to any monotonic computation. For example, extracting from the database only the parts including `bolt1`, can be done using the set `CONTAINS` produced before, i.e. the parts containing other parts, appropriately updated to reflect all the new containments introduced due to additions of new parts.

A wide class of computations performed by real–time systems can be accommodated by incremental computation models. Dramatic performance improvement is obtained when the increments are small. Expert systems doing deductive search are the most representative ones because of the tiny increments between the huge number of one–fact–at–a–time access requests. ICM's also facilitate the dynamic establishment of learned search patterns pertinent to the application. Other classes of systems that will be greatly improved by ICM's include surveillance systems, control and command systems, air–traffic control systems, control of manufacturing processes, etc. Such systems receive data arriving in real time from simultaneous sensor and human observations and require rapid and intelligent assimilation of them with the help of factual information stored in the database. Most of these systems have very high input frequency that results in extremely small input increments. Therefore, ICM's are very appropriate for

these systems.

3.3. Rule processing and optimization

Currently, most expert systems and simple solutions to building EDS's, such as interfacing PROLOG with a DBMS, are characterized by *one–fact–at–a–time* access which drives the deductive search. Very often, access to a single fact requires a long database search. Consider the example of the previous section, where the join of the `BOM` relation with itself is performed to determine whether or not `CONTAINS(part1,bolt1)` is true. Imagine a search where a large list of `CONTAINS` pairs had to be checked using an ordinary system. For each pair, the join is to be constructed, accessed, and thrown away, only to be repeated again and again. The solution to this problem is to utilize existing database access paths to avoid such high searching costs. Since the join will be the most used operation in such a system, efficient support for its computation is needed.

We can again exploit the idea of *Incremental Learning* for avoiding the high cost of repeating the same relational operators over and over. This can be done by storing a view as a persistent cache structure after it is requested for the first time. In our part hierarchy example, a system could learn which of the part–subpart pairs are joinable with other part–subpart ones during the processing of the first query on `CONTAINS`, and *short–cut* the joins during subsequent queries involving `CONTAINS`. This incremental learning avoids all the database search needed to reconstruct the join.

Incremental models and their learning capability generalize the utilization of common access paths, and permit a framework for the reuse of the optimization performed by the query optimizer. The optimization techniques will be transformed into inter–query incremental algorithms that will **amortize** their cost over a series of queries.

Another expensive computation is performed in Expert Systems because queries involve rules with more than one definition (more than one rule with the same "head" in deductive definitions). This expands the initial query to a set of queries to be processed by the system. In this case, support for multiple–query processing is needed. It is often advantageous to process a collection of queries differently than at the query–at–a–time manner. The reason is that queries may share data and therefore elimination of redundant accesses may be possible. In

[SELL85,SELL86a] we study this problem and suggest several multiple–query processing algorithms. Performance improvements can be achieved at a very high degree depending on the extent these queries access common data. Since many EIS applications have a lot of rules defining the same entity, support for multiple–query processing will be an important component of the query optimizer.

For production rules that contain database updates, as is the case for update dependencies, we must consider not only processing efficiency, but also concurrency, locking, etc. However, a generalization of the techniques that apply to transaction management in traditional database systems can also be applied to update dependencies. Compile–time analysis of the conditions in update dependencies can help identify the relations that may be updated and therefore need to be locked when an update dependency is instantiated. After instantiation, an evaluation of the conditions may help pre–release some of the locked relations before the actual updates of the remaining relations are performed. As in traditional database applications, updates always consist of retrieval followed by modification of data. The aspect of retrieving data efficiently has already been addressed above and we shall therefore concentrate on modifications. The efficiency of processing update dependencies and propagating changes can be facilitated by a deferred update strategy (see section 3.7). This is combined with the incremental approach which allows very concurrent protocols for interdependent data.

3.4. Indexing large rule bases

Another subject of interest is how to search the intension of the knowledge efficiently. As mentioned in the previous section, resolving a task can be done either with backward or with forward chaining. The problem of quickly finding the rules that need be applied on a given situation constitutes the problem of indexing the rule base. In the case of backward chaining, the index is defined on the head of each rule. Most methods up to now build an index on the predicate name [FUTO78] or use superimposed coding techniques [RAMA86], to take the arguments into account. Thus, all the relevant rules for a query will be retrieved with few disk accesses, avoiding the I/O bottleneck.

In the case of forward chaining, one is interested in finding quickly which rules satisfy a given condition. For example, given a collection of rules

IF <CONDITION> THEN <ACTION>

one has to index these rules based on their CONDITION part so that given a specific situation, all rules that are applicable can be efficiently recovered. The situation is more complicated than before, since conditions involve general expressions (such as selections or joins) instead of just predicate names and attributes. Our proposal here lies in transforming conditions on relations to objects in some high–dimensionality space [SELL86b] where relation attributes are thought of as the coordinates. Then, a given state of the database can be modeled as an object in this space (described by the values of the various attributes) and the problem of detecting applicable rules reduces to an intersection problem. For example, for the **MRP** relation, a condition 30≤cost≤40 and 20≤lead_time≤40 can be represented as a rectangle in a two–dimensional space with coordinates cost and lead_time. The lower left corner and the upper right corner of such a rectangle will have coordinates (30,20) and (40,40) respectively. A scheme based on multi–attribute hashing [FALO87b,FALO87c] or multi–dimensional trees, such as R^+–trees [FALO87a,SELL87a], can then be used to limit the search by rejecting quickly many non–applicable rules, thus improving the search performance and response time.

3.5. Complex objects

Complex objects may be represented using relations. We illustrate what needs to be done for complex objects by showing a database of drawings from the CAD part of the CAD/MRP system. Suppose that we have two relations DRAWING(DID,DESIGNER,DATE,COMPONENTS) and COMPONENTS(CNAME,TYPE,OTHER_INFO), where in the former relation the field COMPONENTS is further decomposed using the fields from the second relation. Figure 2 shows an instance of this schema; this is simply used to show the structure of the objects, not the way they are actually implemented.

There are three rows in the DRAWING relation; the fourth column is itself the COMPONENTS relation that corresponds to a particular drawing. Notice, that component c005 is found in both drawings with id's 001 and 002; therefore, the information regarding type and status is replicated.

Clearly, one way to store complex objects is by simply storing the object hierarchy in a "flattened" way. For example, the most natural way would be to store the hierarchy represent-

DRAWINGS					
DID	DESIGNER	DATE	COMPONENTS		
			CNAME	TYPE	OTHER_INFO
001	Samuels	2/10/87	c001	ALU	working
			c002	Adder	working
			c003	MUX	working
			c004	MUX	testing
			c005	BUS	testing
002	Lever	3/15/87	c006	Adder	working
			c005	BUS	testing
003	Simmons	4/20/87	c007	Encoder	problems
			c008	Decoder	working

Figure 2: An Example Complex Object

ing an object in *pre-order*. That is, store the top-level component, then each one of its children followed by their children, etc. This approach provides high clustering which enables the fast retrieval of all the components of an object. However, it is very inefficient if some partial retrieval is requested. For example, in the case of the DRAWING object, if only the information on TYPE is requested, the whole hierarchy needs be traversed. Moreover, this approach suffers in the case where the instance of an object does not fit in one physical page. In this case, multiple page accesses are required, even in the case where the requested part of the object is really small in size.

As a solution to this problem, Lorie et al. [LORI85] suggested that the hierarchy be stored in a simple way using "logical" pointers to connect the various components. The reason that physical addresses are not used for pointers is because reorganization of the database would result in major reorganization of the data structures themselves. The solution suggested in [LORI85] is the *Indirection Table* which simply serves as a translation mechanism for logical pointers. Of course, the use of the indirection table has the effect of more costly retrievals since the various components of an object may be scattered to random places on the physical device. However, updates are less costly compared to the first proposal because there is no need to move data around.

Although the Lorie et al. approach is very straightforward and easy to implement within relational database systems, it is useful only for full object retrievals. If the retrieval of just a sub–object is requested, still the hierarchy needs be traversed. To overcome this undesirable effect, Valduriez et al [VALD86] suggested a different approach. Each instance of an object is assigned a unique object identifier (*surrogate*). All instances of an object along with their associated surrogates are stored in a relation. Surrogates are used to link the various objects with their components, in the same sense that pointers were used in the Lorie et al. proposal. However, no ordering is assumed in the way the various instances of an object are stored.

Retrieval of an object is achieved through relational joins. Joins are performed on surrogates only. Since the join is a very costly operation, special support is required to accelerate its execution. Moreover, an intelligent caching mechanism based on incremental computation to avoid costly maintenance of the objects is needed. On top of that, caching of the whole object may be possible depending on the frequency with which such an object is requested as well as on the frequency that such an object or its components are updated. In general the support for complex objects can range from

1) none, in which case joins must be used to retrieve the various components, to

2) caching surrogate pairs (see section 4.1), in which case the surrogates provide direct access to sub–objects, to

3) caching full object components, in which case no retrieval from the database is needed.

Clearly the cost of maintaining the above schemes increases in going from strategy (1) to (2) to (3). We choose to experiment with the second solution because it offers the advantages of efficient update propagation and requires less space to store cached results. The specific storage requirements, retrieval and update frequencies of the objects are used to decide an optimal cache allocation.

3.6. Incremental/Decremental database version management

EIS databases consist of a large collection of objects that continuously evolve. As mentioned above, one can think of an object as a collection of relational views attached to a surrogate. A version of an object is a set of pointers that point to the logs of the changes to these views comprising the object. This storage model reduces redundancy and allows

incremental/decremental generation of versions using differential files.

Concurrent control protocols for derived and bound data objects are needed to maintain multiple versions. Such a protocol has been proposed in [ROUS86a,b]. Concurrent multiple accesses of derived objects are not delayed by this protocol, but, to the contrary, it may speed the access of them. This surprising result can be simply explained by the way incremental version update algorithms work. When a process, say A, locks a derived data object D to obtain a version VD1 of it, any other process, say B, trying to access VD1 or another later version VD2 of the same object D, is locked out waiting. When A finishes the derivation of VD1, it releases it for B and any other process blocked in the meantime. However, as soon as VD1 is released, B and all the other waiting processes can use it (as is or to generate VD2) without having to repeat the version regeneration. Since B requests VD1 after A, B will obtain VD1 sooner than if it had to do the version generation alone by itself! For this reason it is called **slow down for speeding up** protocol.

Decremental computation is useful for two reasons: first, database version regeneration (for recovery or any other reason) becomes easier to manage compared to checkpoint techniques. Reversing the database is done by undoing the changes stored in the differential files. Second, it is very important in large and evolving EIS's to be able to see information as of some specific time point in the past without affecting the current state of the database. Other approaches to answering historical queries degrade so much the queries to the current state that they make them useless. The decremental approach can be used to efficiently construct views from the database that reflect a past state of the database.

3.7. Bindings Between Objects in Tightly Coupled Interoperable Tools

In an EIS environment the basic assumption is that there is always a distinction between special purpose workstations or tools and the mainframes housing one or more EIS's. This calls for a mainframe–workstation architecture of the EIS that tightly couples the workstations with the mainframe and allows downloading of data and processing on the workstations. The downloaded data objects are bound to data objects on the mainframe. These bindings must be maintained incrementally to avoid massive data transmission and/or recomputation.

The proposed architecture in [ROUS86a,b] consists of a mainframe database management system and a large number of workstations running a stripped version of the same DBMS doing independent processing. Each workstation has enough disk capacity which is used for data buffering between the mainframe and the local processing. Data and processing is downloaded to the workstation thus alleviating the mainframe's load.

To guarantee database consistency, updates to base files (i.e. non–derivable from other files) are done only on the mainframe while propagation of these updates to derived objects on the workstation use the locking protocol described in the previous section. Downloaded objects bound to mainframe ones are updated on demand incrementally. Incremental update is done on the workstation by transmitting the differential files on demand, that is when the the bound objects are accessed. This **deferred update strategy** with no *broadcasting* was adopted in [ROUS86a,b] because it drastically reduces the overhead caused by the message traffic to synchronize the updates in an environment with hundreds of workstations. In an ordinary distributed DBMS architecture the message traffic due to broadcasting for obtaining the appropriate locks is extremely high, four times the number of workstations.

4. Implementation Status

We have implemented the access methods of a database system based on the principles of Incremental Computation Models. Our just finished prototype, called ADMS [ROUS84,ROUS85,ROUS87c], exhibits tremendous speeds in accessing learned access paths based on incremental computing. The improvement over conventional re–execution systems ranges from ten to eighty times faster, depending on the size of the increments of the utilized access paths. Our expectations based on analytical results [ROUS87a] were topped by the implementation. Our main results show that

(a) the smaller the increments, and

(b) the higher the complexity of the computation,

the higher the improvement. The second characteristic is due to the fact that multi–level computations observe a lot of access path locality that subsumes a lot of the computation that ordinary re–execution models would have to repeat.

Another important characteristic of the ADMS incremental algorithms is that they are one–pass and, thus, permit interleaving of the update and cache of the access paths. This allows ADMS to produce results much earlier than any re–execution model. Therefore, the response time is very close to zero simply because ADMS starts displaying records of the previous computations while updating. In simulations performed on a VAX 8600 and for complex queries, the response to display the first record was below 2 seconds versus minutes response time using the conventional re–execution technique of query modification. After the first record is displayed, the flow of display is bounded by the speed of the terminal which is much slower than the rate of production of the incremental algorithms. This is very important for real–time systems because no existing system can come close to such response times in queries that involve, say, two or three join operations.

Another advantage of using ADMS is that an extension of it to work in an integrated mainframe and multi–workstation environment using incremental bindings of distributed data objects has been already studied [ROUS86a,b]. This is important considering the fact that EIS's are mostly workstation oriented.

In summary, ADMS is a very powerful database system with a sophisticated view system and access methods that are especially useful for an efficient implementation of large bodies of engineering knowledge, rules, interoperability, and complex objects.

We are currently focusing on two major subjects. First, we study the theoretical foundations, the algorithms, and the design issues of the various components. Second, we have initiated an efforts towards the implementation of our ideas on top of ADMS. Among others, we study the design of knowledge base catalogs and rule and meta–rule definition languages. Similarly, definition and manipulation languages for update dependencies have been devised and are under implementation. In terms of query processing algorithms we have implemented relational operators using cache techniques and have initiated an effort towards the implementation of multiple–query optimization algorithms. We have currently a simple prototype for defining and manipulating complex objects and are currently working on improving it with sophisticated data structures. Finally, the protocols for workstation–mainframe communication have been worked out in detail. We expect that a working version of our prototype will be available in 2 years.

5. Conclusion

In this paper we have suggested efficient means for building high performance engineering information systems. These systems will be able to support engineering knowledge bases of significant size, innovative self–improving performance capabilities based on novel cache structures and access methods tuned for very sophisticated inference engines. Our design goals are

(1) To support large Engineering knowledge bases in a workstation based environment

(2) To provide very fast query processing

(3) To provide very efficient mechanisms to support various kinds of inferencing in engineering systems ranging from deductive retrieval rules to general purpose update propagation rules.

The innovations in our approach lie in:

(1) The use of incremental computation models for reusing previous computations in conjunction with optimal persistent cache structures which speed up tremendously the access of large knowledge bases. Preliminary simulation results on ADMS, the first incremental database management system developed at the University of Maryland, show speed ups of orders of magnitude.

(2) Use of a novel model for specifying update dependencies of interdependent engineering knowledge. Trigger mechanisms are programmed in this model for controlling the evolution of engineering changes, versions, and interoperable engineering tools.

(3) Novel optimization techniques for deductive queries based on the incremental approach, which recognize the similarities of deductive queries and reuse partial results generated during processing a query.

(4) Efficient methods for storing and searching engineering facts and rules based on clustering and efficient optimization techniques for processing complex queries.

6. References

[BATO85] Batory, D.S. and Kim, W., "*Modeling Concepts for VLSI CAD Objects*", Proceedings of the 1985 ACM–SIGMOD International Conference on the Management of Data, Austin, TX, May 1985.

[BUNE79] Buneman, O.P. and Clemons, E.K., "*Efficiently Monitoring Relational Databases*", ACM Transactions on Database Systems, (4) 3, September 1979.

[CAMM84] Cammarata, S. and Melkanoff, M., "*An Interactive Data Dictionary Facility for CAD/CAM Data Bases*", in [KERS84].

[COPE84] Kopeland, G. and Maier, D., "*Making Smalltalk a Database System*", Proceedings of the 1984 ACM–SIGMOD International Conference on the Management of Data, Boston, MA, June 1984.

[DAYA85] Dayal, U. and Smith, J.M., "*PROBE: A Knowledge–Oriented Database Management System*", Proceedings of the Islamorada Workshop on Large Scale Knowledge Base and Reasoning Systems, February 1985.

[DERR86] Derrett, N.P. *et al*, "*An Object–Oriented Approach to Data Management*", Proceedings of the 1986 IEEE Spring Compcon Conference, San Francisco, CA, March 1986.

[DUDA78] Duda, R. et al., "*Development of the Prospector Consultation System for Mineral Exploration*", SRI International, October 1978.

[FALO87a] Faloutsos, C., Sellis, T. and Roussopoulos, N., "*Object Oriented Access Methods for Spatial Objects: Algorithms and Analysis*", in preparation.

[FALO87b] Faloutsos, C., "*Gray Codes for Partial Match and Range Queries*", IEEE Transactions on Software Engineering, 1987. (to appear)

[FALO87c] Faloutsos, C., "*Design and Analysis of Integrated Access Methods for Text and Attributes*", Department of Computer Science, University of Maryland, College Park, June 1987. (submitted for publication)

[FUTO78] Futo, I. et al., "*The Application of Prolog to the Development of QA and DBM Systems*", in [GALL78].

[GALL78] Gallaire, H. and Minker, J., Logic and Data Bases, Plenum Press, New York, 1978.

[GUTT84] Guttman, A., "*R–Trees: A Dynamic Index Structure for Spatial Searching*", Proceedings of the 1984 ACM–SIGMOD International Conference on the Management of Data, Boston, MA, June 1984.

[HARH87] Harhalakis, G., Mark, L., Bohse, M., and Cochrane,B., "*An Integration of Manufacturing Resource Planning (MRP II) and Computer Aided Design (CAD) Based on Update Dependencies*", Submitted to the Conference on Data and Knowledge Systems for Engineering and Manufacturing, Hartford, Connecticut, October 1987.

[IOAN84] Ioannidis, Y. et al, "*Enhancing INGRES with Deductive Power*", Position Paper, in [KERS84].

[IOAN86] Ioannidis, Y., "*Processing Recursion in Deductive Database Systems*", PhD Thesis, University of California, Berkeley, July 1986.

[KATZ82] Katz, R.H., "*A Database Approach for Managing VLSI Design Data*", Proceedings of the 19th Design Automation Conference, June 1982.

[KERS84] Kerschberg, L., Editor, *Proceedings of the First International Workshop on Expert Database Systems*, Kiawah Isl., SC, October 1984.

[KERS86] Kerschberg, L., Editor, *Proceedings of the First International Conference on Expert Database Systems*, Charleston, SC, April, 1986. Database Engineering, (**5**) 3, September 1982.

[KIMU82] Kimura, F. et al, "*Construction and Uses of an Engineering Database in Design and Manufacturing Environments*", File Structures and Databases for CAD: Proceedings of the IFIP WG 5.2 Working Conference, Seeheim, West Germany, 1982.

[KOCH84] Kochan, D., "*Integrated Information Processing for Manufacturing--From CAD/CAM to CIM*", Computers in Industry (**5**) 4, December 1984.

[KOWA74] Kowalski, R., "*Predicate Logic as a Programming Language*", Information Processing, North Holland, 1974.

[LIND80] Lindsay, R.K., et al., **Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project**, Mc–Graw Hill, Inc, New York, 1980.

[LINN86] Linn, J. and Winner, R., "*Engineering Information Systems: Operational Concepts and Requirements*", Institute of Defense Analysis, Alexandria, Virginia, July 1986.

[LORI83] Lorie, R. and Plouffe, W., "*Relational Databases for Engineering Data*", IBM Research, Technical Report RJ–3847, San Jose, CA, April 1983.

[LORI85] Lorie, R. et al., "*Supporting Complex Objects in a Relational System for Engineering Databases*", in **Query Processing in Database Systems**, Eds. W. Kim, D.S. Reiner and D.S. Batory, Springer–Verlag, 1985.

[MARK85] Mark, L., "*Self-Describing Database Systems - Formalization and Realization*", Technical Report TR–1484, Department of Computer Science, University of Maryland, College Park, April 1985.

[MARK86a] Mark, L. and Roussopoulos, N., "*Meta-Data Management*", IEEE Computer Magazine, (**19**) 12, December 1986.

[MARK86b] Mark, L. and Roussopoulos, N., "*Operational Specification of Update Dependencies*", submitted to ACM Transactions on Database Systems, 1986.

[MELK84] Melkanoff, M., "*The CIMS database: Goals, Problems, Case Studies and Proposed Approaches Are Outlined*", Industrial Engineering, 1984.

[MILL82] Miller, R.A., et al., "*INTERNIST-1, An Experimental Computer-Based Diagnostic Consultant for General Internal Medicine*", The New England Journal of Medicine, (**307**), 8, August 1982.

[MYLO80] Mylopoulos, J. et al, "*A Language Facility for Designing Database Intensive Applications*", ACM Transactions on Database Systems, (**5**) 2, June 1980.

[McDE82] McDermott, D., "*R1: A Rule Based Configurer of Computer Systems*", Artificial Intelligence, (**19**) 1, September 1982.

[RAMA86] Ramamohanarao, K. and Shepherd, J., "*A Superimposed Codeword Indexing Scheme for Very Large Prolog Databases*", Proceedings of the Third International Conference on Logic Programming, London, England, 1986.

[ROUS82a] Roussopoulos, N., "*View Indexing in Relational Databases*", ACM Transactions on Database Systems, (**7**) 2, June 1982.

[ROUS82b] Roussopoulos, N., "*The Logical Access Path Schema of a Database*", IEEE Transactions on Software Engineering, (**8**) 6, November 1982.

[ROUS84] Roussopoulos, N., Bader, C., and O'Connor, J., "*ADMS: An Advanced Database Management System: Design Document*", Department of Computer Science, University of Maryland, College Park, January 1984.

[ROUS85] Roussopoulos, N. and O'Connor, J., "*ADMS: Query Language and Programmatic Interface*", Department of Computer Science, University of Maryland, College Park, April 1985.

[ROUS86a] Roussopoulos, N. and H. Kang, "*Preliminary Design of ADMS±: A Workstation–Mainframe Integrated Architecture for Database Management Systems*", Proceedings of the 12th International Conference on Very Large Data Bases, Kyoto, Japan, August 1986.

[ROUS86b] Roussopoulos, N. and H. Kang, "*Principles and Techniques in the Design of ADMS±*", IEEE Computer Magazine, (**19**) 12, December 1986.

[ROUS87a] Roussopoulos, N., "*The Incremental Access Method of View Cache: Concept and Cost Analysis*", submitted for publication to the ACM Transactions on Database Systems, March 1987.

[ROUS87b] Roussopoulos, N., "*Incremental Computation Models*", Department of Computer Science, University of Maryland, College Park, March 1987.

[ROUS87c] Roussopoulos, N., "*Overview of ADMS: A High Performance Database Management System*", Invited Paper, Fall Joint Computer Conference, Dallas, TX, October 25–29, 1987.

[SCIO84] Sciore, E. et al, "*Towards an Integrated Database-PROLOG System*", in [KERS84].

[SELL85] Sellis, T. and Shapiro, L., "*Optimization of Extended Database Languages*", Proceedings of the 1985 ACM–SIGMOD International Conference on the

Management of Data, Austin, TX, May 1985.

[SELL86a] Sellis, T., "*Global Query Optimization*", Proceedings of the 1986 ACM–SIGMOD International Conference on the Management of Data, Washington, DC, May 1986.

[SELL86b] Sellis, T., "*Optimization of Extended Relational Database Systems*", PhD Thesis, University of California, Berkeley, July 1986.

[SELL87a] Sellis, T., Roussopoulos, N. and Faloutsos, C., "*The R^+-tree: A Dynamic Index for Multi-Dimensional Objects*", Proceedings of the 13th International Conference on Very Large Data Bases, Brighton, England, September 1987 (to appear).

[SELL87b] Sellis, T., Roussopoulos, N., Mark, L. and Faloutsos, C., "*High Performance Expert Database Systems: Efficient Support for Engineering Environments*", Submitted to the Conference on Data and Knowledge Systems for Engineering and Manufacturing, Hartford, Connecticut, October 1987.

[SHIP81] Shipman, D., "*The Functional Model and the Data Language Daplex*", ACM Transactions on Database Systems, (**6**) 1, March 1981.

[STON86] Stonebraker, M. and Rowe, L. "*The Design of POSTGRES*", Proceedings of the 1986 ACM–SIGMOD Conference on the Management of Data, Washington, DC, May 1986, pp. 340–355.

[STON87] Stonebraker, M., Hanson, E. and Hong, C., "*The Design of the POSTGRES Rules System*", Proceedings of the Third International Conference on Data Engineering, Los Angeles, CA, February 1987.

[ULLM85] Ullman, J., "*Implementation of Logical Query Languages for Data Bases*", Proceedings of the 1985 ACM–SIGMOD International Conference on the Management of Data, Austin, TX, May 1985.

[VALD86] Valduriez, P., Khoshafian, S. and Copeland, G., "*Implementation Techniques for Complex Objects*", Proceedings of the 12th International Conference on Very Large Data Bases, Kyoto, Japan, August 1986.

[ZANI83] Zaniolo, C., "*The Database Language GEM*", Proceedings of the 1983 ACM–SIGMOD International Conference on the Management of Data, San Jose, CA, May 1983.

[ZANI84] Zaniolo, C., "*PROLOG : A Database Query Language for all Seasons*", in [KERS84].

[ZANI85] Zaniolo, C., "*The Representation and Deductive Retrieval of Complex Objects*", Proceedings of the 11th International Conference on Very Large Data Bases, Stockholm, Sweden, August 1985.