EXTENSIBLE DATABASE SYSTEMS

by

Michael J. Carey
David J. DeWitt

Computer Sciences Technical Report #585

February 1985

EXTENSIBLE DATABASE SYSTEMS

Michael J. Carey
David J. DeWitt

Computer Sciences Department
University of Wisconsin

This research was partially supported by the Department of Energy under contract #DE-AC02-81ER10920 and the National Science Foundation under grant MCS82-01870.

Abstract

Research and development in the database management systems area has traditionally concentrated on supporting business applications. Recently, the design of database systems capable of supporting non-traditional application areas such as engineering applications for CAD/CAM and VLSI data, scientific and statistical applications, expert database systems, and image/voice applications has emerged as an important direction of database system research. These new applications differ from more conventional applications (such as transaction processing and record keeping) in a number of critical aspects, including data modeling requirements, processing functionality, concurrency control and recovery mechanisms, and access methods and storage structures.

A number of groups are attempting to meet the demands of these new application areas by either building interfaces on top of existing relational systems or by trying to extend the functionality of such systems. While these approaches may appear promising, we feel that neither is likely to provide acceptable performance. Instead, we feel that it time to design and prototype a new generation of database management systems. Our goal is to produce a *core database management system* that can be easily extended to meet the demands of new applications. We envision that the system will permit extensions to the data modeling, query processing, access method, storage structure, concurrency control, and recovery components of the system. To achieve this goal we are currently exploring the use of an object-oriented approach as the basis of such a system.

1. INTRODUCTION

Until recently, research and development efforts in the database management systems area were focused on supporting traditional business applications. The design of database systems capable of supporting non-traditional application areas, including engineering applications for CAD/CAM and VLSI data, scientific and statistical applications, expert database systems (which can really be viewed as adding database system technology to an AI application), and image/voice applications, has now emerged as an important new direction for database system research. These new applications differ from more conventional applications like transaction processing and record keeping in a number of important areas:

(1) **Data modeling requirements**. Each new application area requires a different set of data modeling tools. The types of entities and relationships that must be described for a VLSI circuit design are quite different from the data modeling requirements of a banking application.

(2) **Processing functionality**. Each new application area has a specialized set of operations that must be supported by the database system. Consider, for example, a database containing satellite images. It makes little sense to talk about doing joins between these images or even components of a single image. Instead, we are more likely to be interested in analyzing the image using specialized image processing and/or recognition algorithms. For example, if we are looking for crop diseases we will want to apply an algorithm that examines the images for the crop disease signatures. As another example, if we wanted to look for particular types of ships, we would be interested in using an image recognition algorithm to compare the satellite images with those of the relevant types of ships. As we will discuss in more detail below, we contend that it does not make sense to implement such algorithms in terms of relational database operations (or CODASYL or network data model primitives either).

(3) **Concurrency control and recovery mechanisms**. Each new application area also has slightly different requirements for concurrency control and recovery mechanisms. While locking and logging are the accepted mechanisms for conventional database applications, a versioning mechanism looks most appropriate for engineering applications. For image databases, which tend to be principally accessed in a read-only fashion, perhaps no concurrency control or recovery mechanism is needed.

(4) **Access methods and storage structures**. Each new application area also has dramatically different requirements for access methods and storage structures. Access and manipulation of VLSI databases is facilitated by new access methods such as R-Trees [GUTT84]. Storage of image data is greatly simplified if the database system supports large multidimensional arrays as a basic data type (a capability provided by no commercial database system at this time). Storing such images as tuples in a relational database system is generally either impossible or terribly inefficient.

A number of new applications for database system technology have emerged in the past few years, and it is clear that new applications will continue to emerge. Three alternative solutions to the problem of providing the database system technology needed by current and future applications seem fairly obvious. We call these solutions the "PL/I" approach, the "custom" approach, and the "extension" approach. We comment on these three alternatives in more detail below.

(1) The first approach, which we refer to as the "PL/I" approach, is to build one monolithic database system that satisfies the needs of all application areas. Like PL/I, the resulting system would undoubtably be complex to use and its performance would most likely be disappointing. Furthermore, large monolithic software systems tend to be complex to extend and difficult to maintain.

(2) The second strategy, which we call the "custom" approach, would be to construct a new (custom) database system for each new application. Obviously this approach would provide excellent performance for each application area. However, since database systems are difficult and time-consuming to construct, there would be a long lead time between the recognition of the need and when the system could be brought on-line. Also, new results on algorithms and access methods appear fairly frequently in the database literature, and such improvements would probably be difficult to incorporate once the system was up and running. Furthermore, although each application is different, the resulting systems would inevitably contain a number of similar pieces of software.

(3) The third approach, which we term the "extension" approach, is to build interfaces for these new application areas on top of existing or slightly extended database systems [STON82, STONE83, HASK82]. At first glance, this approach appears promising. In Section 2, we will address why systems produced

using this approach will inevitably provide unacceptable performance.

We feel that the most promising solution is a different one altogether, to design and prototype a new generation of database management systems. The goal of our research is to design a core database management system that is easily extended to meet the demands of new applications. We envision that the system will permit extensions to the data modeling, query processing, access method, storage structures, concurrency control, and recovery components of the system. While we would like to make these extensions as simple to accomplish as possible (i.e. by a database administrator), in practice what we really expect is that it will take a programmer a relatively short period of time (e.g. 2-3 months) to add the desired extensions. Initially, we will explore using an object-oriented approach as the basis for the proposed system. In Section 3, we will present an overview of the system and our initial research objectives.

2. EXTENDING EXISTING DATABASE SYSTEMS

Since relational database systems are now widely available, there are a number of efforts underway to utilize such systems as the basis for expert, scientific, statistical, CAD/CAM, and VLSI database management systems. In this section we will explore in detail why we feel that this approach will produce systems with disappointing performance.

As a first step in understanding this claim, consider the reasons given by Date for using a database system [DATE81]:

(1) Insure data independence.

(2) Reduce redundancy in stored data.

(3) Share stored data.

(4) Enforce standards.

(5) Centralize application of security restrictions.

(6) Balance requirements of conflicting user groups.

(7) Maintain data integrity.

The software required to achieve these objectives varies. No software is needed to reduce redundancy in the stored data, to permit sharing of stored data, to enforce standards, to centralize application of security restrictions, or to balance conflicting requirements. To obtain data independence (which is defined to be "the immunity of application programs to changes in storage structures or access methods" [DATE81]), the database system needs schema-driven query processing routines. That is, all operations on the database must be performed according to a model of the logical and physical database design that is captured in the schema. Concurrency control and recovery software is needed to maintain data integrity during concurrent access by multiple users and during hardware and software failures.

Consider, on the other hand, the internal structure of a typical relational database management system as shown in Figure 1. Of the software modules in Figure 1, the query processing routines, concurrency

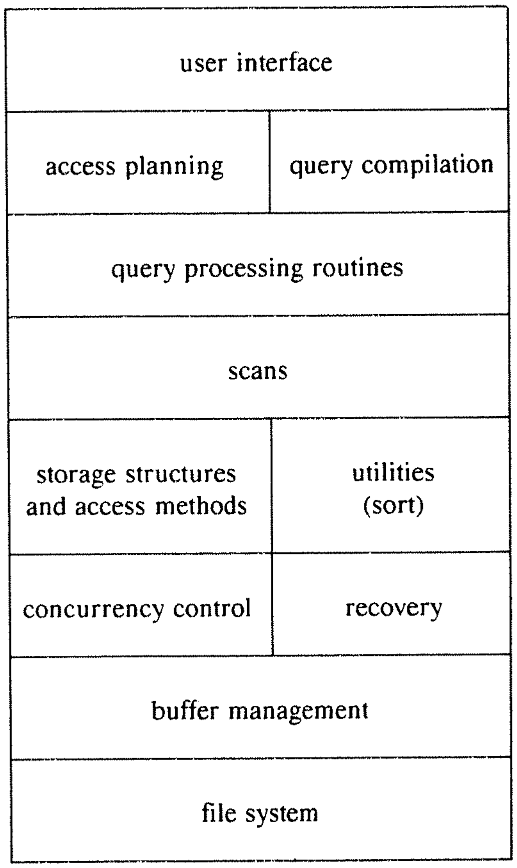

Figure 1: Architecture of a Typical Relational DBMS.

control, and recovery modules are really the only necessary ones[1]. The remaining pieces (query optimization and compilation, scans, storage structures and access methods, and buffer management) can really be considered as "glue". This is not to say that the glue is not important. Rather, it is really only there to insure that the system has reasonable performance. Furthermore, the glue varies from data model to data model. For example, a CODASYL system has little or no need for sophisticated query optimization and compilation modules.

When one attempts to build a database system for a new application (e.g. CAD/CAM) using a relational database system as the basis, a number of problems arise. First, the appropriate glue for a relational database management system is frequently the wrong glue for the new application area. Consider, for example, the application shown in Figure 2. The access methods provided by a relational system (i.e. one dimensional indices) are not adequate for a CAD/CAM environment. As a solution, Guttman recently introduced a new access method called R-trees for handling searches on spatial data [GUTT84]. The same is true for the buffer management routines. In [KOSH84], we show that for operations on a statistical database, significant gains in performance can be obtained by letting the operation itself control replacement of pages in the buffer pool rather than by using a generalized buffer manager.

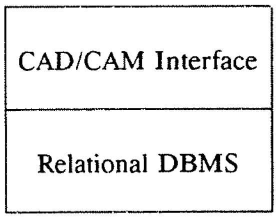

Figure 2: One Approach to CAD/CAM Database Management

While one could cite a number of other examples, these examples should make it obvious that the glue for a relational system is simply not the right glue for most (and perhaps all) of the emerging application areas. Furthermore, it appears that each of these applications will have its own glue just as relational, CODASYL, and hierarchical database systems do presently.

In [STON83, STON84], Stonebraker addresses adding abstract data types and abstract indices to

[1] Even the user interface can be viewed as unnecessary, as it need be present only to support an ad-hoc query interface.

INGRES [STON76] and implementing QUEL as a data type as a way of addressing the needs of new application areas. While the proposed additions to INGRES are certainly a step in the right direction, we do not feel that they go far enough. In particular, even with the abstract indices proposal, a good deal of the wrong glue remains (access planning and buffer management). Furthermore, it is not obvious[2] that anybody other than RTI or the University INGRES group could add a new access method such as R-trees to INGRES.

We contend, instead, that a new strategy for implementing database management systems is needed which permits glue to changed/added/deleted in order to customize the system for its intended application. In addition, we feel that although all applications-oriented database systems will have query processing routines and concurrency control and recovery mechanisms (as it is these components that really make a piece of software a database system), each of these mechanisms will also vary from system to system. For example, while one might be able to implement an expert database system by adding deduction and a rule base to a relational database system, a statistical database system must support sophisticated data analysis routines (e.g. regression analysis) as its basic query processing capabilities [KOSH84]. Furthermore, it is generally accepted that standard two-phase locking and logging (which are the algorithms most commonly used in a relational database system) are not the right concurrency control and recovery mechanisms for CAD/CAM and VLSI database systems [LORI83].

In the following section we will outline our research plan for designing and prototyping an extensible database system. This system will make it simpler (though maybe not trivial) to implement database systems for new application areas.

3. RESEARCH OBJECTIVES

As a solution to the problems outlined above, we propose to design, prototype, and evaluate a new generation of database management systems. The goal is to produce a prototype system that can be easily extended to satisfy new and future applications while still providing excellent performance. Our principal research objectives include:

[2] This is based on our experiences using INGRES code as a basis for the front end of DIRECT [DEWI79].

(1) Definition, exploration, and evaluation of alternative architectures for an extensible database system.

(2) Query optimization and compilation techniques that permit the addition of new operators at the data model level and the addition of new access methods.

(3) Concurrency control and recovery techniques.

(4) Query processing strategies.

(5) Buffer management policies.

Following completion of the research phase of this project we intend to implement two prototypes. The first will be that of a conventional relational database management system. Such an effort will serve two functions. First, by seeing how long it takes to build such a system, we will be able to see how realistic our claims of "easy" were. Second, we have found from past work [CHOU83, BITT83] that benchmarking is a very effective way of illustrating performance bottlenecks in a system. By benchmarking the performance of a conventionally developed relational database system such as INGRES against that of a relational system developed using our extensible database management system, we will be able to identify performance bottlenecks in the new system and determine whether the approach provides a satisfactory level of performance.

We will also construct a second prototype system to illustrate that the system can be used successfully for new application areas. Three alternatives look appealing. An expert database system would require extending the first prototype to include new data objects for storing rules and new query processing routines for executing the rules. A prototype for a statistical database system would illustrate the flexibility of the system for supporting complex operations on the database (e.g. multivariate regression analysis) and operator driven buffer replacement strategies. A third alternative might be a database system for CAD/CAM. Wisconsin's Mechanical Engineering department is very active in this area and could serve as a user community as well as a source of guidelines for developing the system.

3.1. A Preliminary Architecture for an Extensible Database System

We are currently examining the use of an object-oriented approach as the basis for an extensible database system. An object-oriented approach appears attractive for the following reasons:

- First, a number of researchers have shown that objects[3] are useful as data modeling tools. Examples of this application of objects can be found in [WEBE78, BROD78, ROWE79, SHEP84]. In [WEBE78] classes are used to represent entity sets and relationships between entity sets. For example, in a supplier-parts database, there would three classes: one for the suppliers entity set, one for the parts entity set, and one for the relationship between suppliers and parts. Instances of the class supplier correspond to supplier objects (i.e. actual suppliers). Associated with the supplier class is a data structure corresponding to the attributes of the entity set and procedures to add/delete suppliers, to change the address of a supplier, etc. The operators associated with the supplier-part class (e.g. no_longer_supplies, new_supplier_of) would be implemented using the operators provided by the supplier and part classes. The operators available to the user of such a system are exactly those provided by the outermost set of classes.

- Objects have also been shown to be a reasonable implementation strategy for database systems. In [BARO81, BARO82] we have shown that by using one class for each modeling construct provided by the data model (e.g. the relation construct in the case of the relational data model), the high level operators associated with the data model (e.g. joins, selects, etc.) can be implemented entirely in terms of the operators associated with the basic class types. For the relational data model there is one class of type relation. With a supplier-parts database, instances of this class would be the objects corresponding to the suppliers, parts, and SP relations. Associated with each of the objects is an implementation of all the operators associated with the class. However, the operators of each object are customized to accommodate the internal schema information (i.e. storage structures and access methods) of each relation.

[3] While we have chosen to use the word **object**, others have used **abstract data type** or **module** for the basis of their work. We consider an object to be an encapsulation of a data structure and a set of externally visible operators that operate on this data structure. For our purposes we do not care whether these operations are invoked via messages (ala Smalltalk [GOLD83, HAGM83, KAEH83]) or via procedure calls (ala Mesa [MITC79]). Each object is assumed to have a unique identifier by which it can be addressed. A **class** is a generic description of a set of like objects.

- Finally, objected-oriented programming ala Smalltalk 80 [GOLD83] provides a very flexible approach for constructing software systems. We feel that these same characteristics should be present in a database system constructed using the object-oriented methodology.

  Two other groups are presently implementing object-based database management systems. Servio-Logic [COPE84] has extended Smalltalk 80 into a database system. Beech [BEEC83] has defined an object-based data model and is in the process of implementing a database system to support this model. While these approaches have a similar flavor to the proposed research, they differ in their goals. Neither project addresses using an object-based mechanism as a basis for simplifying extensions to the underlying components of the supporting database system. We view our research to be complementary to these other efforts.

Figure 3 presents our first idea of a design for the proposed system. We envision a 4 or 5 level architecture. At the the bottom level of the system are a collection of storage structure objects. The second level consists of access method objects. Like a conventional database system, the function of the access method objects is to facilitate access to objects at the storage structure level. Level 3 contains query processing objects. It is the function of this level to implement the query processing functionality. Finally, at level 4 we have the data model objects. If the data model objects layer is used to directly implement the conceptual schema and operations on these entities (ala [WEBE78] and [ROWE79] and others, as described above) then this is the final layer. If, on the other hand, the data model objects correspond to the modeling tools provided by a data model [BARO81, BARO82] (e.g. the relation construct in the relational model and the set and record type constructs in the network data model), then a fifth layer of objects corresponding to the entity sets and relationships of the conceptual schema may be present. In addition to this hierarchy of objects, additional pieces of software will be present for query optimization, query compilation, buffer management, concurrency control, and recovery. Our goal is to make these pieces of software generic by implementing them entirely using the operations associated with the classes at each of the levels.

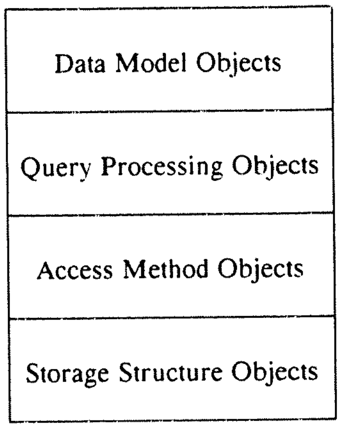

Figure 3: Architecture for an Extensible DBMS.

Associated with each object[4] are a set of operators that can be applied to the object. It is these operators that appear to be the key to constructing a flexible/extensible database management system. For example, we expect that a storage structure object will provide something like the following set of operators:

- **read** - to move the object from mass storage to main memory
- **write** - to move the object from main memory to mass storage
- **obtain shared access** - a primitive used as part of the concurrency control mechanism
- **obtain exclusive access** - a primitive used as part of the concurrency control mechanism
- **write undo information** - a primitive used as part of making updates recoverable
- **write redo information** - a primitive used as part of making updates recoverable

While these operators look rather mundane, they are a key component of our system because they provide a uniform interface to manipulating objects at the storage level. Consider, for example, the read operator. In the case of fixed length records, read is simple. The object identifier is first translated to a disk address, and then the proper block is read into memory. On the other hand, the read operator for a more complex storage object that spans multiple disk blocks [CHOU83, HASK82] will require a much more complex read operation. We find this approach attractive as it permits access method objects to be completely ignorant of the implementation of the storage structure objects that they provide access to.

[4] Or more accurately, with each class type

A related problem that we will examine is how to automatically provide access methods (indexes) for many applications. An index provides a way to efficiently locate objects with certain properties or attribute values. For example, a hashed index provides fast lookup for finding objects with a particular attribute value, and a B+ tree index provides fast lookup for finding objects whose attribute value lies within a range of values. We plan to provide a set of access methods for organizing collections of objects in certain standard ways — for instance, as a *mapping*, where the access method supports exact match searches, or as an *ordering*, where the access method supports both exact match and range searches. In order for a class of objects to be indexed in these ways, the implementor of a storage structure class will have to provide routines to return an attribute value given an object (for use in building the index and searching index leaf pages), to test for attribute value equality (for searching a mapping index), and to test for attribute value precedence[5] (for searching an ordering index). Note that what we are calling "attribute values" could just be simple field values for records, or they could be derived attribute values for complex objects. An example of the latter type of attribute value might be the power consumed by VLSI circuit objects — one could build an ordering index to organize a VLSI cell library according to the power demands of the cells. Our index research will initially focus on identifying the set of standard access methods desired, such as the mapping and ordering access methods described above. Other standard access method types might support such query classes as multi-dimensional exact-match, range, and partial match queries [ULLM82, ROBI81, NIEV84]

The concurrency control primitives also appear interesting. For example, obtaining write access to one type of object may result in setting (or attempting to set) an exclusive lock in a lock table and then checking for deadlock. On the other hand, if a versioning scheme [CHAN82, REED83] for concurrency control is used, invoking the "obtain write access" operator may cause a new copy of the requested object to be generated.

Details of the operators associated with the objects at the other levels in the design have not yet been developed. In the following sections we will outline some of the research problems that must be addressed.

---

[5] Such as "less than".

3.2. Query Optimization and Compilation in an Extensible DBMS

One of our early research goals is to tackle the problem of query optimization and compilation. As is obvious from reading the definitive paper by Selinger et. al. [SELI79] on this topic, query optimization and access path selection for a conventional database system is a complex piece of code to implement. As evidence of this claim, our benchmarks [BITT83] illustrate that a number of commercial products have simply left this piece of code out of their system.

Since our proposed system can be extended both by adding new operators and new access methods, query optimization becomes even more complicated. Consider the case when the set of operators is fixed and we extend the system by adding a new access method. To exploit this new access method two things need to occur. First, algorithms must be specified for those operators that can benefit from using the new access method. Second, the query optimizer must extended to understand the CPU and I/O costs associated with using these new algorithms and access method (plus any side effects, such as changes in the ordering of the objects that the algorithm produces that could affect subsequent costs).

The approach that we are considering is to use an expert system as the basis for the query compilation and optimization process. This strategy appears attractive for several reasons. First, it seems that an expert system is ideally suited to the task of evaluating alternative access plans. This task is similar to design tasks for which expert system technology has proven useful in the past, such as configuring a computer system [MCDE80] or selecting a good hardware design given a register-level description of a computer architecture [KOWA83].

The second attraction of the expert system approach to query optimization is that incorporating new access methods and operators into the optimization process will require simply that a few new rules be added. For a new access method, rules will describe the cost of accessing an object (or a collection of adjacent objects) via the access method, the query patterns for which the access method is applicable, the cost of inserting and deleting objects using the access method, and the names of the routines associated with the access method's implementation. For a new operator, rules will describe the query pattern for which the operator is applicable, the ordering properties of the operator (i.e., those that determine how it can be moved around within a query during the optimization process), the names of the alternative routines that implement the operator and their associated costs, etc. The bulk of the query optimization expert system, which will include rules that describe how to identify a "good" query execution plan, how to perform algebraic transformations on a query, how to recognize candidate access methods and operator routines, how to cut off unlikely regions of the access plan search space, and how to generate the plans themselves, will remain unchanged when new access methods and operators are added.

To test the effectiveness of the expert system approach to query optimization, we intend to use it in the construction of a new relational database machine that we are implementing. Our current plan is to use the OPS5 production system language for this purpose [FORG81], as OPS5 has been the implementation language for several successful expert systems (including [MCDE80] and [KOWA83]).

3.3. Concurrency Control and Recovery in an Extensible DBMS

There appear to be two main flavors of transactions that any such database system must support: short and long duration transactions. Debit/credit transactions fall into the first category, while CAD/CAM and VLSI design transactions fall into the second category. Since database systems have traditionally been designed to maximize the performance of debit/credit transactions, locking and logging are the customarily used mechanisms. On the other hand, a shadow/versioning mechanism [LORI83, HASK82, REED83, KATZ84] appears to be the best alternative for long duration transactions that must survive system crashes.

Implementing both mechanisms in one database system presents a number of interesting research problems. As should be obvious from above, each class of storage-level objects will have several concurrency control operators associated with it, and different classes may use different concurrency control mechanisms. One problem to be addressed is resolving conflicts between transactions that access objects which use different concurrency control primitives. Several researchers have examined the problems associated with the use of mixed concurrency control mechanisms [BORA84, BERN81], but this work seems to preclude the storage of data in shared buffer space (i.e., each transaction needs its own private workspace). We expect our work to be more heavily influenced by the work of Schwarz and Spector on type-specific locking and log-based recovery algorithms [SCHW83, SCHW84]. In addition to the fact that Spector and Schwarz have shown locking to be applicable to objects of various abstract data types, other researchers have shown that versions can be integrated with locking [BAYE80, STEA81, CHAN82], and IBM researchers have described lock protocols for long duration transactions based on persistent locks (i.e., locks stored on stable storage) [LORI83]. Our research in the concurrency control and recovery area will involve making it easy to tailor a well defined set of concurrency control and recovery protocols to new types of objects. In particular, we will investigate how long duration transactions can be supported through type-specific versioning and persistent lock protocols.

A second research area that we plan on exploring is how to best do concurrency control on objects at the access method level. In particular, we need to understand the relationship between concurrency control at the access method level and the concurrency control primitives used at the storage structure level.

3.4. Main Memory Management of Objects

As discussed above, each class of storage structure objects has associated with it primitives to move class instances between mass storage and main memory. Buffer management, however, will be implemented at a much higher level in the system. As predicted by Stonebraker [STON81] and demonstrated by our research [CHOU85, KOSH84], significant performance gains can be obtained by using an operation controlled buffer management policy instead of a more general replacement mechanism such as LRU. At any instance in time, all main memory access method and storage objects will be associated with some transaction. In the case of objects that are being shared by multiple transactions, the transaction that most recently touched the object will viewed as the owner (sort of a "hot potato" algorithm). Instead of managing the objects associated with a transaction in an LRU fashion as is done by the hot-set algorithm [SACC82], when the buffer manager needs to replace a storage object it will invoke a free-space operator associated with the query processing object. This operator will return the name of the storage/access method object that it views as the most desirable to eject from main memory. The cost of losing this object might also be returned. If the cost value is sufficiently high, the buffer manager may instead try to get space from another transaction. Alternatively, it may be possible to have all of the transactions that are sharing an object provide their estimate of its replacement cost, in which case a globally inexpensive object could be selected for replacement.

4. CONCLUSIONS

In this paper we have outlined a set of research objectives for exploring the design and implementation of a new generation of database management systems. This research work should result in the implementation of a prototype database system that permits the data modeling. query processing. access method. storage structures. concurrency control. and recovery components of the system to be easily extended/adapted to best satisfy the needs of new applications. At this point in time we are concentrating on an objected-oriented approach as the basis for implementing such a system.

5. REFERENCES

[AGRA83] Agrawal, R., and DeWitt, D.J., "Integrated Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation", Technical Report No. 497, Computer Sciences Department, University of Wisconsin, Madison, February 1983.

[AGRA85] Agrawal, R., Carey, M.J., and Livny, M., "Models for Studying Concurrency Control Performance: Alternatives and Implications", Proceedings of the ACM SIGMOD International Conference on Management of Data, 1985.

[ASTR76] Astrahan, M., et. al., "System R: Relational Approach to Database Management", ACM Transactions on Data Systems, V 1, N 2, (June 1976), pp. 119-120.

[BARO81] Baroody, A.J. and D.J. DeWitt, "An Object-Oriented Approach to Database System Implementation", ACM Transactions on Database Systems, Vol. 6, No. 4, December, 1981.

[BARO82] Baroody, A.J. and D.J. DeWitt, "The Impact of Run-Time Schema Interpretation in a Network Data Model DBMS", IEEE Transactions on Software Engineering, March 1982.

[BAYE80] Bayer, R., Heller, H., and Reiser, A., "Parallelism and Recovery in Database Systems", ACM Transactions on Database Systems V 5, N 2, June 1980.

[BERN81] Bernstein, P., and Goodman, N., ''Concurrency Control in Distributed Database Systems'', *ACM Computing Surveys* 13(2), June 1981.

[BEEC83] Beech, D., "Introducing the integrated data model", Hewlett-Packard Computer Science Laboratory Technical Note CSL-15, January, 1983.

[BITT83] Bitton, D., DeWitt, D. and C. Turbyfill, "Benchmarking Database Systems - A Systematic Approach", Proceedings of the 1983 Very Large Database Conference, Florence, Italy, (October 1983), pp. 8-19.

[BORA84] Boral, H. and I. Gold, "Towards a Self-Adapting Centralized Concurrency Control Algorithm", Proceedings of the 1984 SIGMOD Conference, Boston, Ma., 1984.

[BROD78] Brodie, M. and H. Schmidt, "What is the use of abstract data types in databases?", Proceedings of the 1978 VLDB Conference, pp. 140-141, 1978.

[CARE84a] Carey, M.J., and Muhanna, W., "The Performance of Multiversion Concurrency Control Algorithms", Technical Report No. 550, Computer Sciences Department, University of Wisconsin, Madison, August 1984.

[CARE84b] Carey, M.J., and Stonebraker, M.R., "The Performance of Concurrency Control Algorithms for Database Management Systems", Proceedings of the Tenth International Conference on Very Large Data Bases, Singapore, 1984.

[CHAN82] Chan, A., et al. "The Implementation of an Integrated Concurrency Control and Recovery Scheme", Proceedings of the ACM-SIGMOD International Conference on Management of Data, 1982.

[CHOU83] Chou, H.T., DeWitt, D.J., Katz, R., and A. Klug, Design and Implementation of the Wisconsin Storage System, to appear, Software Practice and Experience, also Computer Sciences Technical Report No. 524, November, 1983.

[CHOU85] Chou, H.T., Buffer Management of Database Systems, Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, February, 1985.

[COPE84] Copeland, G. and D. Maier, "Making Smalltalk a Database System", Proceedings of the 1984 SIGMOD Conference, Boston, Ma., 1984.

[DATE81] Date, C.J., *An Introduction to Database Systems*, Addison-Wesley, 1981.

[DEWI79] DeWitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems", IEEE Transactions on Computers, June 1979, pp. 395-406.

[FORG81] Forgy, C.L., "OPS5 User's Manual", Technical Report No. CMU-CS-81-135, Carnegie-Mellon University, 1981.

[GOLD83] Goldberg, A. and D. Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, 1983.

[GRAY79] Gray, J., "Notes On Database Operating Systems", in *Operating Systems: An Advanced Course*, Springer-Verlag, 1979.

[GUTT84] Guttman, T., "R-Trees: A Dynamic Index Structure for Spatial Searching", Proceedings of the 1984 SIGMOD Conference, Boston, Ma., 1984.

[HAGM83] Hagmann, R., "Preferred classes: a proposal for faster Smalltalk-80 execution", in *Smalltalk-80: Bits of History, Words of Advice*, G. Krasner, Editor, Addison-Wesley, 1983, pp. 323-330.

[HASK82] Haskin, R. and R. Lorie, "On Extending the Functions of a Relational Database System", Proc. ACM SIGMOD Conference, Orlando, FL, (June 1982), pp. 207-212.

[KAEH83] Kaehler, T. and T. Krasner, "LOOM — Large objected oriented memory for Smalltalk-80 systems", in *Smalltalk-80: Bits of History, Words of Advice*, G. Krasner, Editor, Addison-Wesley, 1983, pp. 323-330.

[KATZ84] Katz, R. H. and T. J. Lehman, "Database Support for Versions and Alternatives of Large Design Files", IEEE Transactions on Software Engineering", V SE-10, N 2, (March 1984).

[KOSH84] Koshafian, Setrag, "A Building Blocks Approach to Statistical Databases", Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, June, 1984.

[KOWA83] Kowalski, T., and Thomas, D., "The VLSI Design Automation Assistant: Prototype System", Proceedings of the 20th ACM-IEEE Design Automation Conference, June 1983.

[LORI83] Lorie, R. and W. Plouffe, "Complex Objects and Their Use in Design Transactions", Proceedings of the Engineering Design Applications of the 1983 ACM-IEEE Database Week, San Jose, CA, 1983.

[MCDE80] McDermott, J., "R1: An Expert in the Computer Systems Domain", Proceedings of the First Annual National Conference on Artificial Intelligence, Stanford, CA, 1980.

[MITC79] Mitchell, J.G., Maybury, W., and Sweet, R., "Mesa Language Manual", Xerox Research Center, Palo Alto, CA, 1979.

[NIEV84] Nievergelt, J., Hinterberger, H., and K.C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure", ACM Transactions on Database Systems, V 9, N 3, (March 1984), pp. 38-71.

[REED83] Reed, D., "Implementing Atomic Actions on Decentralized Data", ACM Transactions on Computer Systems, V 1, N 1, (March 1983).

[ROBI81] Robinson. J. T.. "The K-D-B Tree: A Search Structure-for Large Multidimensional-Dynamic Indices". Proceedings of the 1981 SIGMOD Conference. Ann Arbor. MI.. 1981.

[ROWE79] Rowe. L. and K. Schoens. "Data Abstraction. Views. and Updates in RIGEL. Proceedings of the 1979 SIGMOD Conference. Boston. MA.. 1979.

[SACC82] Sacco. G. M. and M. Schkolnick. "A Mechanism for Managing the Buffer Pool in a Relational Database System Using the Hot Set Model". Proceedings of the 1982 Very Large Database Conference. Mexico City. Mexico. (September 1982). pp. 257-262.

[SCHW83] Schwarz. P.M.. and Spector. A.Z.. "Recovery of Shared Abstract Types". Technical Report No. CMU-CS-83-151. Carnegie-Mellon University. October 1983.

[SCHW83] Schwarz. P.M.. and Spector, A.Z.. "Synchronizing Shared Abstract Types". ACM Transactions on Computer Systems, V 2, N 3, August 1984.

[SELI79] Selinger. P.G.. et. al.. "Access Path Selection in a Relational DBMS". Proceedings of the 1979 SIGMOD Conference on the Management of Data, June 1979.

[SHEP84] Shepherd. A. and L. Kerschberg. PRISM: A Knowledge Based System for Semantic Integrity Specification and Enforcement in Database Systems", Proceedings of the 1984 SIGMOD Conference. Boston. Ma., 1984.

[STEA81] Stearns. R.. and Rosenkrantz. D.. "Distributed Database Concurrency Controls Using Before-Values". Proceedings of the ACM SIGMOD International Conference on Management of Data. 1981.

[STON76] Stonebraker. M.. Wong. G.. Kreps. P.. and G. Held. "The Design and Implementation of INGRES". ACM Transactions on Database Systems. V 1. N 3. (September 1976). pp. 189-222.

[STON81] Stonebraker. M. R.. "Operating System Support for Database Management". Communications of the ACM. V 24, N 7. (July 1981). pp. 412-418.

[STON84] Stonebraker. M.. et. al.. "Query as a Data Type". Proceedings of the 1984 SIGMOD Conference. Boston. Ma.. 1984.

[STON83] Stonebraker. M., et. al.. "Application of Abstract Data Types and Abstract Indices to CAD Databases". Proceedings of the Engineering Design Applications of the 1983 ACM-IEEE Database Week. San Jose. CA. 1983.

[ULLM82] Ullman. Jeffrey D.. *Principles of Database Systems*. Computer Sciences Prese. 1982.

[WEBE78] Weber. H. "A Software Engineering View of Database Systems. Proceedings of the 1978 VLDB Conference, pp. 36-51. 1978.