REPORT ON THE 1989 SOFTWARE CAD DATABASES WORKSHOP

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M89/35

10 April 1989

REPORT ON THE 1989 SOFTWARE CAD DATABASES WORKSHOP

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M89/35

10 April 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

REPORT ON THE 1989 SOFTWARE CAD DATABASES WORKSHOP

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M89/35

10 April 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Report on the 1989 Software CAD Databases Workshop

Lawrence A. Rowe

Computer Science Division-EECS, University of California at Berkeley
Berkeley, CA 94720, USA

A workshop was held to develop a better understanding of the features and database requirements of software development environments. It was organized into a series of moderated discussions between all participants.

The major conclusion was that software development tools need most features found in commercial relational database systems and many features found in next generation object-oriented database systems currently being developed. Specific features required include: object-oriented data models, navigational and set-oriented query languages, complex object support, long transaction support, derived data support, and alerters. It was also apparent that better logical and physical database design tools would significantly improve the development of these new systems.

1. Introduction

A two day workshop on the topic of software CAD databases was held in Napa California in February 1989. Approximately 10 people from the database community and 40 people from the software engineering community attended the workshop. The group included a mixture of people from academia and industry. Attendance was limited to encourage dialog between the two communities. Attendees were selected by a program committee that read position papers submitted by people who wanted to participate. These position papers were published in a workshop proceedings [RoW89].†

The goal of the workshop was to develop better understanding in the software engineering and database communities about the database requirements for software CAD databases,‡ the capabilities of existing commercial database systems (DBMS), and the capabilities of next generation object-oriented database systems (OODBMS) that are currently being developed. The workshop was organized into sessions that covered the following topics: SDE services, database requirements for SDE's, and alternative DBMS architectures. The last session was used to develop general conclusions with which the group could agree. Each session began with a short presentation on the issues and followed by a moderated discussion. A designated person took notes during each session. These notes will be published at a later date.

This paper summarizes the session discussions and conclusions. It was not possible to have all attendees read and comment on the paper due to tight publication deadlines so I apologize in advance for any errors or omissions. The remainder of the paper summarizes the discussions in each session. The session leader is indicated in parentheses.

2. SDE Services *(B. Boehm, TRW)*

This session addressed the services that an SDE system should provide. The goal was to identify the data that should be stored in a DBMS and the kinds of operations that might be performed on that data. Following this discussion, several people presented short "war stories" about their attempts to build an SDE's on a DBMS.

A SDE database must include all information relating to the software lifecycle process. This information includes:

1. Product data (e.g., specifications, code, documentation, etc.).
2. Resource data (e.g., people, facilities, equipment, budgets, etc.).
3. Management data (e.g., schedules, action items, problem reports, etc.).

Figure 1 shows several queries that might be answered by querying this database. The first query involves complex queries over data that is derived from the data stored in the database. The second query may require a change to the product definition (i.e., application schema change). The third query triggers an automated activity. The fourth query shows an example of a fine granularity query on the source code. And finally, the fifth query is an example of a fuzzy query.

The database people at the workshop claimed that queries one, two, and four can be solved with conventional DBMS's assuming that reasonable database designs are used. Queries three and five, on the other hand, are much harder. The ensuing discussion identified several issues related to database support for SDE's including the fact that current commercial DBMS's provide inadequate support for dynamic changes to the database design (i.e., schema evolution), derived data (i.e., data computed from data stored in the database), complex objects, and version control.

---

† A limited number of copies of the proceedings can be ordered from Sharon Wensel who can be contacted by phone (415-642-4662), email (wensel@postgres.Berkeley.EDU), or by postal mail at the same address as the author.

‡ One problem that immediately became apparent is that there is no generally agreed upon term for programming environment tools. The term software CAD (SCAD, pronounced "ess-cad") was suggested by Bill Scherlis at DARPA. In the software engineering community people use other terms including: integrated project support environments (IPSE), software engineering environments (SEE), and software development environments (SDE). In the remainder of the paper I will use the term SDE.

Query 1
List the programmers and managers of all tasks on the critical path with over 5 days of slippage in their current milestones.

Query 2
Take the "computer experience" cost driver attribute for each module in the system and split it into the "computer experience" for the host-system and target-system.

Query 3
Perform an appropriate set of regression tests and report the possible adverse side-effects of every module change.

Query 4
List all exceptions that could be raised by the system for which there is no exception handler.

Query 5
If we change the security level of a specific piece of data, describe how it will effect the security of the complete database.

Figure 1: Example queries.

Several people presented "war stories" about their attempts to build SDE's on a DBMS. William Paseman described the evolution of the Atherton Technology products from a programming language environment tool to an integrated project support environment. The programming language tool supported multiple user access to source code and cross-reference data. The IPSE added support for management control data. Atherton has built an object storage system that supports version and configuration management. They concluded that a programming language environment tool does not require sophisticated database services (e.g., sharing, access control, and associative queries) but that it did need good data modelling, efficient support for fine granularity objects (i.e., abstract syntax tree nodes) and navigational queries (i.e., get next object given an object identifier (OBJID)[§]

Dennis Heimbigner from the University of Colorado at Boulder described his experiences developing a system that manages requirement specifications (REBUS) on top of the Cactis research prototype DBMS [HuK87]. The novel feature of Cactis is that it supports automatic recomputation of derived data in the database.[+] Heimbigner had to develop an interface between ADA and Cactis. He described a variety of problems with interfacing an existing programming language to a DBMS that are well known in the database community (e.g., type compatibility, incompatible data models, etc.). Other problems he described related to the fact that Cactis was a research prototype that did not provide all the functions a commercial DBMS provides (e.g., dynamic schema changes, secondary indexes, sophisticated query optimization, and transaction management). This discussion raised an issue that came up several times during the workshop. A SDE has many database requirements that can be satisfied by features found in different DBMS's. The problem is that no single DBMS provides all the required features.

Mark Dowson, currently at the Software Productivity Consortium (SPC), described two systems: one built on a custom DBMS and one that is being built on a commercial DBMS. The first system, called ISTAR, was built on a federated DBMS, that is, a collection of independent communicating DBMS's. The advantage of this approach is that it may be possible to integrate existing tools into an SDE by interfacing the tool-specific DBMS to the federated DBMS. The disadvantage is that a federated DBMS is really a distributed heterogenous DBMS. Consequently, the standard distributed DBMS problems must be solved (e.g., distributed query optimization, distributed transactions, replicated data, and catalog design and maintenance) [CeP84]. In most cases an independent DBMS cannot be changed so it may be impossible to implement all required facilities (e.g., distributed transactions require that the federated database master process be able to access the local database lock tables or to set timeouts on transactions to implement distributed deadlock detection). In addition, Dowson noted the problems associated with building a custom DBMS. Specifically a DBMS is a large complex software system that requires considerable resources to build and maintain. He also described an effort at SPC to use a commercial SQL-based DBMS to build an SDE. The primary problem that they have encountered is that the conventional transaction model is not appropriate for SDE's. This topic is discussed in more detail below.

The final "war story" was presented by Ian Thomas from GIE Emeraude. He described the PCTE project's Object Management System (OMS). A major goal of PCTE is to create a tool interface abstraction that allows existing tools to be integrated with the SDE. OMS has an entity-relationship model with some object-oriented capabilities (e.g., attribute and relationship inheritance). Two problems were encountered. First, interfacing existing tools to an SDE is a very hard problem. And second, developing a good database design that supports tool integration is difficult. Several people who have tried to build SDE's on databases commented on the difficulty of developing good database designs. The importance of good design tools and the ability to rapidly change a design are well-known problems in the database community.

While some progress on the database design problem has been made in the past decade, too much expertise and effort are required to build a complex database application. Database systems should monitor

[§] An object identifier is a unique identifier assigned by the DBMS that never changes [KhC86].

[+] Cactis uses an attribute grammar to specify the derived data computation. Other research database systems are exploring the use of rules to specify derived data (e.g., POSTGRES [StR86] and STARBURST [LMP87]).

access patterns and automatically change the storage structures so that queries can be executed efficiently. In addition, better support is needed to reduce program and data translation required when the logical database design is changed.

3. SDE Database Requirements *(W. Paseman, Atherton Technology)*

The second session explored in more detail some of the database requirements that were identified in the first session. Several lists of database requirements for SDE's have been published. Figure 2 shows a list developed by Maria Penedo from TRW that was discussed during this session. While a consensus did not emerge, several different viewpoints did emerge during this discussion. First, several database people argued that most of these requirements have already been addressed by commercial relational DBMS's or are being addressed in one of the research prototypes that are currently being developed. A second viewpoint was offered by some of the software engineering people who were unsure that a future, unknown, and unproven DBMS that would solve the SDE problem will be forthcoming within a reasonable timeframe. Finally, others argued that a radically different open database architecture was needed that would allow programming languages to selectively use powerful database features (e.g., associative access, crash recovery, etc.) on data in the database and non-persistent data created by the program. This last proposal is discussed in more detail in the next section.

Extensible data model.
Meta-schema support (i.e., schemas stored as data).
Operations stored with objects and encapsulation.
Explicit relationships.
Support for derived data (i.e., rules).
Transitive closure queries to access hierarchical data.
Multiple programming language interfaces.
Query optimization and indexing.
Complex object support.
Support for large data sets.
Version support.
Automatic selection of storage structures.
Comprehensive access control facilities.
Bulk data load and unload.
Short and long transaction support.
Crash recovery.
Undo facility.
Portable DBMS (i.e., it must run on many platforms).
Client-server architecture.
Distributed database support.
Acceptable performance.

Figure 2: SDE database requirements.

The remainder of this session covered a variety of topics on transactions, query optimization, data models, and historical databases. The most interesting discussion centered around the topic of transactions. Gail Kaiser from Columbia University presented a short overview of the capabilities of a transaction system and the conventional DBMS strategies that are used to implement these capabilities. Several problems were identified including the following.

1. SDE's need more capabilities than a conventional transaction system provides. Specifically, an SDE must be able to manage inconsistency. For example, a tool might require consistency within a complex object such as a program module but inconsistency between complex objects such as the other modules that use the module being modified by the tool. Another example is that a tool may want to enforce consistency, but delay notification to others that an update has been made to the database.
2. SDE's need to support multiple processes within a single transaction. For example, two tools running on a workstation may be showing different views of the same data (e.g., the source code for a procedure and the call graph for the system). Updates can be made to the data through either tool but the database should see them as one transaction.
3. A SDE needs efficient support of different types of transactions. Some applications read and update relatively little data in a transaction. These transactions are called *short transactions*. Other applications execute transactions that run for a long time while the user browses and updates many different objects in the database. These transactions are called *long transactions*. Conventional DBMS's provide excellent support for short transactions. However, these systems have trouble with long transactions because users are prohibited from accessing the data read and written by the transaction.

Kaiser described several approaches that researchers are experimenting with to solve these problems. The first approach uses nested transactions [Mos82]. A nested transaction allows a transaction to spawn a sub-transaction that can commit before the parent transaction commits. The Sun Network Software Environment uses nested transactions [Ade89]. In both systems a user can make several changes to a virtual copy of the database. These changes can be viewed as nested transactions on the virtual database within the larger transaction that will be completed when these changes are merged back into the main database. This approach solves problems 1 and 2 above.

A second approach to solving some of these problems is to use naming domains to control access to the database. In a naming domain, all versions of objects are retained. A user operates on a "configuration" that defines a set of object versions. A transaction is executed with respect to an initial configuration. An update transaction that commits creates a new configuration.

Naming domains can be used to solve problem 1 above, namely, managing inconsistency between complex objects. This approach is being investigated in the COSMOS system [Wae88].

A third approach is called *participant transactions* [Dow89, Kai88] The idea is that several processes can participate in the transaction. Transactions are named so that a process can join a running transaction. Consequently, multiple processes can execute within a single transaction (i.e., it solves problem 2 above). Each process sees the database with all participant's updates, but the rest of the users do not see them.

A fourth approach is to use commit-serializability (CS) transactions. CS allows a transaction to split into several distinct transactions as long as they have disjoint write sets (i.e., the set of objects the transaction has updated) and the read set of each new transaction is disjoint from the other new transactions being created in the split. These new transactions can commit or abort independently or they may join with any other transaction in the system to create another new transaction. All transactions that commit are serializable, but they may be completely different than the set of transactions that were initially created [PK88]. The idea is that transactions are created, split, merged, and committed as the user examines and updates the database. CS transactions can be used to solve problems with long transactions.

Lastly, database researchers are exploring another approach to solving the long transaction problem, called *sagas*. A saga is a long transaction that can be broken up into a collection of sub-transactions that can run at the same time with other transactions. These sub-transactions are related to each other and all must commit for the saga to commit. Sub-transactions are non-atomic which means that database updates made by the sub-transaction can be undone at a later time by a "compensating transaction" that must be defined for each sub-transaction. The advantage of sagas is that more concurrent access is possible because sub-transactions can be completed and the resources they control can be released [GaS87].

Most people agreed that there is still much work to be done in this area.

Another topic discussed in this session was the requirement that a rule in the database invoke some action when the predicate becomes true. For example, a manager might want to be notified when the bug count in a particular part of the system had reached a certain threshold. This capability is called an alerter in the database community [BuC79]. Few, if any, commercial DBMS's support alerters.

4. Alternative DBMS Architectures *(D. DeWitt, U. of Wisconsin)*

The majority of this session was used to allow the developers of various database systems to describe their systems. The following systems were discussed:

Software BackPlane† (Atherton Technology) [Pas89]
Cactis (U. of Colorado at Boulder) [HuK87]
EXODUS (U. of Wisconsin) [Cae88]
Gemstone (Servio-Logic) [CoM84]
Iris (HP Laboratories)
Observer/Encore (Brown U.) [SZR86]
POSTGRES (U.C. Berkeley) [StR86]
A Yet to be Named Product (Ontologic) [And89]

Several themes emerged from these presentations. First, all of the systems are object-oriented in the following senses: 1) they provide richer type systems than a conventional relational DBMS, 2) they support some form of object identity, and 3) they support inheritance. Some, but not all, systems extend a set-oriented query language (e.g., SQL) with user-defined procedures and methods and some store methods and procedures in the database.

The second theme was the importance of support for complex objects. Typically, this support includes some mechanism to load an object composed of many objects with different types that are highly interdependent (i.e., they contain many attributes with references to other objects in the complex object) very quickly. Object references are represented by OBJID's that are assigned by the DBMS and never changed. The load process usually translates the database representation of values to an appropriate representation for the program. This translation is called *swizzling*. Most systems convert OBJID's to main memory pointers, called *pointer swizzling*, so that subsequent references can be implemented very efficiently. Main memory performance is critical for many of the applications that these systems are addressing, including SDE's.

A third theme that emerged was that any next generation DBMS must provide all functionality that is provided by current commercial relational DBMS's. This fact was apparent both from the requirements list presented in figure 2 and the discussion during the workshop. Specifically, the DBMS must support associative queries, multiple programming language interfaces, database procedures (i.e., the ability to dynamically link application code into the DBMS process), and conventional transactions.

The discussion then turned to an object-oriented programming system with an integrated database that allows a programmer to use database functionality on any object. The basic idea is that some database functions (e.g., associative queries and atomic operations) should be available on objects created by a program that are not persistent. In addition, these functions should be applied uniformly to across all objects (i.e., persistent and non-persistent). Examples are queries that search for data in the database, in a program cache that holds objects that have been fetched from the database, and non-persistent objects in the program. Another example is that it should be possible to define a rule on database and program objects.

The software architecture that runs on the distributed system shown in figure 3 was proposed by several

† Trademark of Atherton Technology.

people. The object cache holds database and program objects. Object references in the program access this cache directly. Associative queries are handled by the distributed DBMS code in the client machine. This code treats the object cache as another local data manager similar to the DBMS that runs on the server machine. While this architecture is conceptually clean, many hard problems remain to be solved. For example, how does the system optimize a complex query that joins database and program objects where some of the database objects have been fetched into the object cache and modified by the program. Several groups in the database and programming language communities are working on similar systems [Bae82, Kie88, Row86, SZR86].

At the end of the session two issues related to standards were raised. First, someone said that they wanted a better object-oriented data model than the model provided by C++. This issue was raised because most attendees recognized that C++ will be the most widely-used object-oriented programming language model due to the popularity of C. The problem with a C++ data model is the absence of a standard set abstraction, a rules system, and a set-oriented query language. Tim Andrews from Ontologic identified the real problem when he noted that his company had developed a better data model in their VBASE product but that the marketplace was not interested in it.

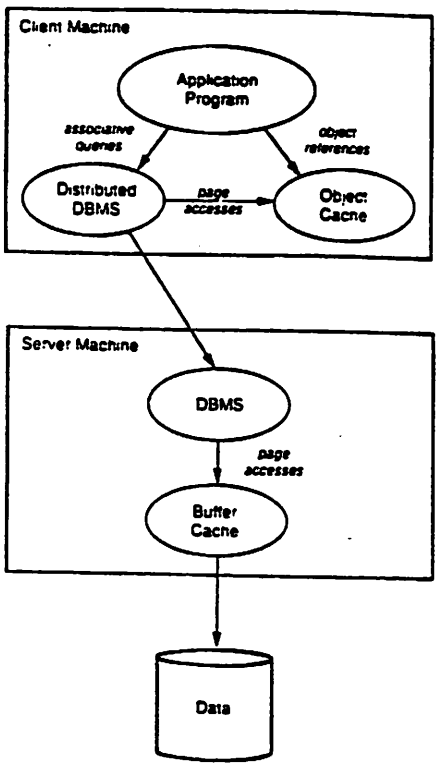

Figure 3: Integrated programming environment architecture.

The second issue raised was whether SQL was the right query language. As with C++, SQL is clearly the dominant query language and it is likely to remain so for a very long time. The problem with SQL is the difficulty of extending it to support new features (e.g., transitive closure queries and complex object support).

5. Workshop Conclusions *(L. Rowe, U.C. Berkeley)*

The final session produced the following list of conclusions with which the majority of attendees could agree.

1. There are many inconsistent and confusing terms in both communities. Everyone who attended agreed that the meeting had been productive in that it exposed some of this confusion and in some cases led to agreement on common terminology (e.g., participant transactions).
2. The development of an SDE, viewed as a database application, requires more programmer control than conventional business applications. Specifically, there is an urgent need for more functionality (e.g., complex object support, versions, database rules, alerters, transitive closure queries, non-traditional transaction models, better integration of database services and program environments, and schema evolution support) while at the same time providing acceptable performance.
3. Next generation DBMS's will be object-oriented and they will have to provide a superset of the capabilities found in current commercial relational DBMS's.
4. Version management is not well understood and there is no evidence that database systems will provide the required support for the sophisticated version systems required by an SDE.
5. Better schema evolution support is needed. At one point during the workshop people discussed the idea of the SDE being able to run consistently across major changes to the SDE schema and still answer the kinds of complex queries described above. This capability presents a major challenge to database researchers that might not be achievable.
6. The majority of attendees were skeptical that an acceptable, commercially supported DBMS with all the features required by an SDE will be forthcoming in a reasonable timeframe.
7. Interfacing existing tools to an SDE is a very hard problem and nobody has any good ideas about how to solve it. Some people thought this will be a critical requirement for future SDE's.
8. Lastly, many people agreed that there must be life after C++ and SQL, but everyone reluctantly agreed that the marketplace would continue to make them the dominant languages.

Acknowledgements

I want to thank Rick Adrion, Dave Dewitt, and Lee Osterweil who supported this workshop. Without their help, it would never have happened. I also want to thank the session note takers: Gail Kaiser, Dennis Heimbigner, and Maria Penedo. I could not have written this report without their input.

References

[Ade89] E. Adams and et. al., "Object Management in a CASE Environment", *Proc. 11th Int. Conf. on Software Engineering*, Pittsburgh, PA, May 1989.

[And89] T. Andrews, "Database Support for Software Design", *Proc. 1989 ACM SIGMOD/SIGSOFT Workshop on Software CAD Databases*, Napa, CA, Feb. 1989, 3-4.

[Bae82] R. Balzer and et. al., "Specification-Based Computing Environments", *Proc. 8th VLDB Conf.*, Sep. 1982, 273-279.

[BuC79] O. P. Buneman and E. K. Clemons, "Efficiently Monitoring Relational Databases", *ACM Trans. Database Systems*, Sep. 1979, 368-382.

[Cae88] M. Carey and et. al., "The EXODUS Extensible DBMS Project: An Overview", Computer Sciences Technical Report #808, Univ. of Wisconsin, Nov. 1988.

[CeP84] S. Ceri and G. Pelagatti, *Distributed Databases – Principles and Systems*, McGraw-Hill, New York, 1984.

[CoM84] G. Copeland and D. Maier, "Making Smalltalk a Database System", *Proc. 1984 ACM-SIGMOD Conf. on Management of Data*, June 1984.

[Dow89] M. Dowson, "Nested Transactions and Visibility Domains", *Proc. 1989 ACM SIGMOD/SIGSOFT Workshop on Software CAD Databases*, Napa, CA, Feb. 1989, 36-38.

[GaS87] H. Garcia-Molina and K. Salem, "Sagas", *Proc. 1987 ACM-SIGMOD Conf. on Management of Data*, San Francisco, CA, May 1987.

[HuK87] S. E. Hudson and R. King, "Object-Oriented Database Support for Software Environments", *Proc. 1987 ACM-SIGMOD Conf. on Management of Data*, San Francisco, CA, May 1987.

[Kai88] G. E. Kaiser, "Extended Transaction Models for Software Development Environments", Technical Report CUCS-404-88, Columbia Univ. Dept. of Comp. Sci., 1988.

[KhC86] S. N. Khoshafian and G. P. Copeland, "Object Identity", *Proc. 1986 OOPSLA Conf.*, Portland, OR, Sep. 1986, 406-416.

[Kie88] W. Kim and et. al., "Integrating an Object-Oriented Programming System with a Database System", *Proc. 1988 OOPSLA Conf.*, San Diego, CA, Sep. 1988, 142-152.

[LMP87] B. Lindsay, J. McPherson and H. Pirahesh, "A Data Management Extension Architecture", *Proc. 1987 ACM-SIGMOD Conf. on Management of Data*, San Francisco, CA, May 1987.

[Mos82] J. E. B. Moss, "Nested Transactions and Reliable Distributed Computing", *Proc. 2nd Symp. on Reliability in Dist. Soft. and Database Sys.*, Pittsburgh, PA, July 1982. Available from IEEE Computer Society Press.

[Pas89] W. Paseman, "Architecture of the Atherton Software BackPlane", *Proc. 1989 ACM SIGMOD/SIGSOFT Workshop on Software CAD Databases*, Napa, CA, Feb. 1989, 105-108.

[PK88] C. Pu, G. E. Kaiser and N. Hutchinson, "Split-Transactions for Open-Ended Activities", *Proc. 14th Int. Conf. on Very Large Data Bases*, Los Angeles, CA, Aug. 1988, 26-37.

[Row86] L. A. Rowe, "A Shared Object Hierarchy", *Proc. Int. Wkshp on Object-Oriented Database Systems*, Asilomar, CA, Sep. 1986.

[RoW89] L. A. Rowe and S. Wensel, editors, *Proc. 1989 ACM SIGMOD/SIGSOFT Workshop on Software CAD Databases*, Napa, CA, Feb. 1989.

[SZR86] A. H. Skarra, S. B. Zdonik and S. P. Reiss, "An Object Server for an Object-Oriented Database System", *Proc. Int. Wkshp on Object-Oriented Database Systems*, Asilomar, CA, Sep. 1986.

[StR86] M. R. Stonebraker and L. A. Rowe, "The Design of POSTGRES", *Proc. 1986 ACM-SIGMOD Conf. on Management of Data*, Washington, DC, June 1986.

[Wae88] J. Walpole and et. al., "A Unifying Model for Consistent Distributed Software Development Environments", *Software Eng. Notes 13*, 5 (Nov. 1988).