

Database Systems for CAD

Vinícius Medina Kern

Department of Industrial and Systems Engineering

ABSTRACT

Conventional database systems are good for business data management, but they are not adequate for CAD data management, since they force some undesirable characteristics, such as flat files, small and fixed-sized records, and only one version of data at a time. Object-oriented databases have emerged as techniques that can match the needs of applications such as CAD, multimedia and expert systems. This paper discusses the needs of CAD database systems using object-oriented modeling and outlines management techniques for schema evolution, establishment of views and relationships, concurrency control, and object version control.

INTRODUCTION

Management of CAD data has been done by means of dedicated mechanisms embedded in CAD systems. This can be a good solution for a designer, but the current trends on concurrent engineering impose the need for sharing data among diverse users and applications for different purposes.

Conventional database systems manage data independently of the application, but impose limitations that are inadmissible in CAD environments. On the other hand, object orientation emerged as a programming language paradigm comprising desirable features for CAD, such as rich data modeling and a uniform framework for engineering and system objects. To meet the needs of CAD data management, recent work [1,2] advocates the extension of the Object-Oriented paradigm to databases.

This paper discusses database management systems and techniques for computer-aided design. Database techniques used to manage business data are outlined, and the reasons why they are inappropriate for complex applications like CAD are explained. Object-oriented features desirable in a CAD environment and the requirements of a database system for CAD in the scope of object-oriented databases are addressed.

CONVENTIONAL DATABASE MANAGEMENT

Conventional databases (currently relational and formerly hierarchical and network) are well-established systems for business data management. A relational database is a collection of relations (tables) consisting of rows and columns. Relational Database

Management Systems (DBMSs) have a set of tools to manage data independently of the application. These tools include facilities for concurrent access control, database schema evolution, information uniqueness, and version control.

Concurrent access is accomplished by locking the data at file or record level, since this procedure takes a very short period of time that is imperceptible to the user. Transactions are data changes accomplished in three well-defined steps: update (implies data lock), and commit (accepts transaction and releases lock) or rollback (discontinues the transaction, keeps original data). The system keeps track of the last updating, so that database stability can be assured in the event of a crash.

Database schema represents the structure of data. This structure is filed in a *data dictionary*. Constraints expressed in the data dictionary are enforced by the database system and any structural change requires a complete system stop. Since business data are somewhat well and statically structured, this lack of dynamic character is admissible.

Identity is based on primary key, which is a field or concatenation of fields in a record (row of a relation) whose content must be unique in the relation. In terms of implementation, a relation corresponds to a file. *Integrity* is guaranteed by the database system. Changes which could duplicate or put a null value in a primary key are not allowed.

Versioning is treated in a very plain fashion in relational databases. All data have only one version: the current. CAD designers must try various versions of design objects until they are able to decide upon the better version, so the relational strategy for versioning is inadequate.

Business data fit in the relational model of flat files or tables, but CAD applications require data that can hardly fit in flat files. Furthermore, the information stored can be very large, making the conventional locking mechanism impractical. A CAD transaction can last for hours, even days--definitely longer than a computer session. A crash in an uncommitted update would provoke rollback, wasting the expensive partial update. There is a critical difference of data models and object manipulation paradigms for languages and conventional databases, which is referred to as *impedance mismatch* [1].

Data independence is a great advantage of database systems. If we want to manage CAD data this way, however, current database systems are not adequate. There is a need for a new database approach to manage applications like computer-aided design.

OBJECT-ORIENTED MODELING IN A CAD ENVIRONMENT

Object-orientation, like all other software paradigms, aims to "model the world" as closely as possible. Joseph *et alii* and Heiler *et alii* [1,2] agree that the engineering design process of defining an initial version and then changing the design is very similar to defining objects and then successively refining them, specifying required constraints and building hierarchies of objects. This process is quite usually concurrent and multilevel, rather than linear. In this sense, object-orientation is a common model that eases the mapping from the designer's mental model of the objects to the user interface (system design tools), and from this user interface to the underlying system facilities.

Abstraction is an object-oriented mechanism that separates two views of the object (interface and implementation) in order to hide the internal implementation details.

Programming languages treat rich data modeling capabilities, such as abstraction and hierarchies, in the transient memory of the computer. There is a need to extend these capabilities to objects that must persist, and give them a database treatment. With object-orientation, databases can be integrated with engineering applications in a non-obstructive manner.

OBJECT-ORIENTED DATABASE SYSTEMS FOR CAD

Not only designs, but management data, metadata (data that explains data), methodologies, programs and tools must persist. But most CAD systems provide poor database support, relying only on user ability to conceptualize, file and retrieve design data. In this section, some crucial issues on persistent object-oriented systems are outlined.

Relationships

Relationships are associations among objects. Heiler *et alii* [2] argue that most object-oriented approaches support only IS-A relationships, a kind of generalization-specialization relationship, where the object belonging to a specialized class inherits the properties of the general object, and adds its own special features. Other relationships have been proposed, such as COMPONENT-OF, INSTANCE-OF, VERSION-OF, CONFIGURATION.

Even these relationships don't cover all applications. The application semantics vary, and there is a need for user-defined relationships. There is also a need for flexible inheritance mechanisms, in a way which allows the user to specify what properties, operations and constraints should be inherited, and how multiple inheritance conflicts can be resolved.

Views

Instead of the two views of interface and implementation provided by the object-oriented mechanism of abstraction, it should be possible to support many views to satisfy the requirements of different applications of users. For instance, tasks such as layout, garbage disposal and budget planning should have their own adequate views.

Schema evolution

Schema evolution deals with changes in which class definitions are the objects to evolve, which means changing the structure of the data rather than the data themselves. In relational systems, database schema is static. Any schema change requires a system shutdown. This is not a serious problem, though, since business data are scarcely subject to structural modifications. Complex applications like CAD, however, demand flexibility in defining and changing class definitions and inheritance in a class lattice. Furthermore, schema is affected at run time. For example, if there are two variables with the same name in two superclasses, what is to be inherited? Some rule must solve this conflict.

This difference between static relational and dynamic object schema is addressed by Joseph *et alii* [1]: "Objects in a database are created using 'templates'; these templates are relations in a relational system and classes in a object system. ... It is not possible to know the correct structure of all templates when the application is first launched".

Kim *et alii* [3] propose rules of schema evolution based on properties called *invariants*. A class lattice in a still state must preserve all the invariants:

- Class lattice invariant: The class lattice is a rooted and connected directed acyclic graph (DAG) with uniquely named nodes and edges labeled uniquely for a node.
- Distinct name invariant: All instance variables of a class must have distinct names.
- Distinct identity (origin) invariant: If one attribute can be inherited in several ways, it must not be duplicated. This situation is illustrated in figure 1 where the inheritance of the attribute *Weight* from *Vehicle* by *Submarine* can be accomplished in two ways, either through *WaterVehicle* or *NuclearPoweredVehicle*.
- Full inheritance invariant: inherit all variables and methods of the superclasses, except when full inheritance causes a violation of the distinct name/identity invariants. In Figure 1, the database system must provide a way to decide whether *Submarine* inherits *Size* from *NuclearPoweredVehicle* or *WaterVehicle*, or from both, assigning different names.

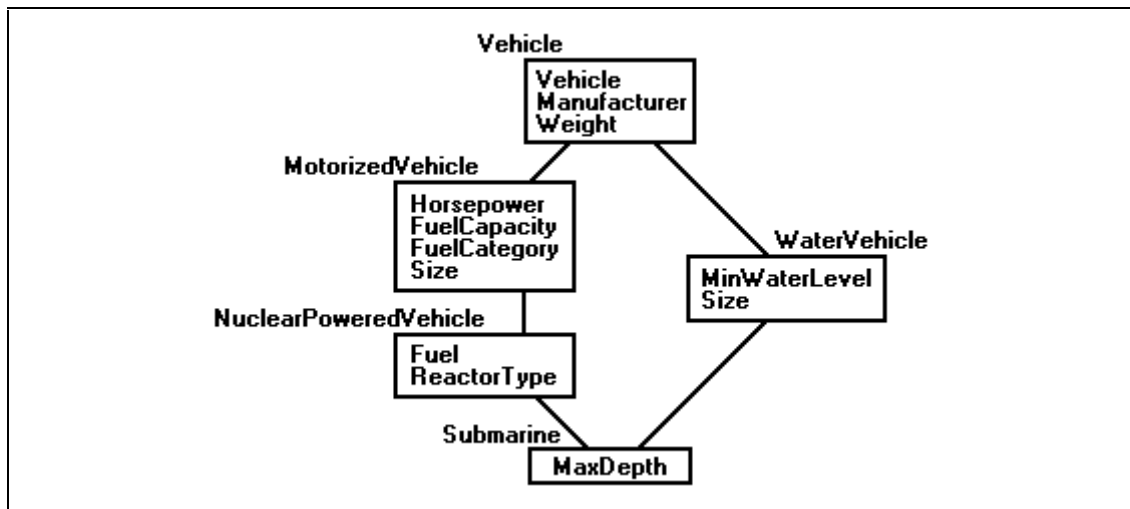


Figure 1 - A class lattice with inheritance conflicts

Source: Kim *et alii* [3]

- Domain capability invariant: Domains of super and subclasses must be compatible.

Concurrency control, transactions and crash recovery

Concurrency control is the task of sharing information among concurrent users, so each user can run a database application as if he/she was the only user in the system. For cooperative work purposes, they can still interact with one another.

It's necessary for the sake of database stability that every change has an *all or nothing* effect [1], which is referred to as *atomicity*. In other words, transactions must either succeed or fail completely. Recovery strategies can be used to rebuild the initial state of an uncommitted transaction.

Usually, CAD teams cooperate from days to months over the same design, so the conventional strategy of locking all the data in a transaction is inappropriate. Furthermore, designers often take cues from the partial results of other designers to guide their work.

Recently Joseph *et alii* [1] pointed out the *check-in/check-out* technique as one of the possible solutions to cooperative work. Using this technique, designs are checked out from the global to a private workspace, then used and changed with no concurrency control, and finally checked in the global database, enforcing normal concurrency control. All the changes in the design operated in the private workspace are treated as a single transaction.

Kim *et alii* [3] criticize the utility of the check-in/check-out technique as too much constrained. Effectively, the effort spent in changing is lost if the portion of the design checked out and changed meets a changed design when trying to check back in. Therefore, the responsibility for data integrity is burdened on the designers.

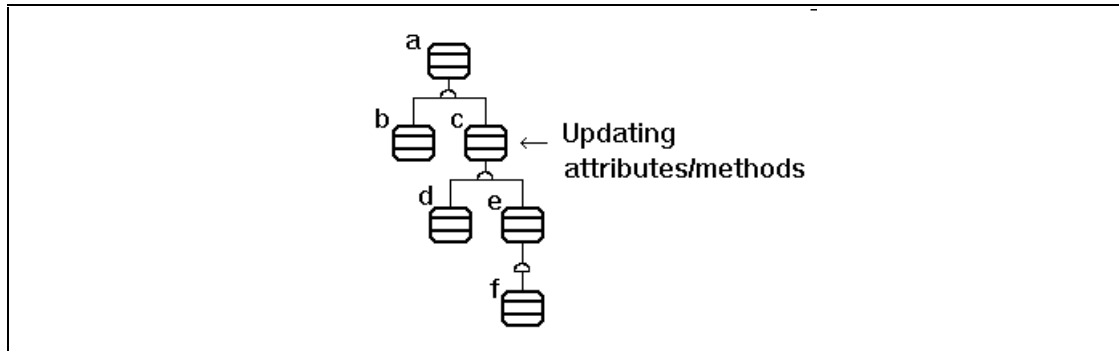


Figure 2: Transaction changing attributes/methods in class **c** locks classes **d**, **e** and **f**. Another transaction attempting to update class **a** must wait.

Concurrent access control was faced by Kim *et alii* [3] by means of the concepts of *sessions* and *hypothetical transactions*, and locking explicitly on all subclasses of the object being updated.

Session is a series of transactions, so one transaction can take effect only when its previous transaction is terminated. The active transaction can exist in multiple windows of a workstation, thus decreasing the problem of long wait until a transaction is terminated. For example, in an evolving design of a car, the active transaction deals with changing the chassis, but this object is being used, so the transaction cannot obtain a lock on the chassis. The designer can work on another window of the same session or transaction rather than wait for the chassis transaction to be resumed.

Hypothetical (rather than normal) is a transaction that always aborts. It is very suitable to *what-if* experiments. Whatever the result of the transaction, the database remains unaffected.

Locking subclasses is as follows: If attributes or methods of a class are created or deleted, the subclasses will face the same effect. Hence, if another transaction attempts to modify the definition of the class or any of its superclasses, it will find that the subclasses are already locked, as shown in figure 2.

Current research on concurrency control for object-oriented databases let several issues open. In brief, it is claimed that it needs to operate at the level of objects in the user model, rather than at the level of files or records [2]. Software or hardware failures should not simply abort a transaction and reject it completely, but should better notify the users and annotate the data. There is a need for user-defined locking strategies, specifying what and when constraints are to be checked, and what actions are to be taken on failure [2].

It's difficult to build generic concurrency control mechanisms. In general, the more improved is the concurrency, the less effective is the control.

Versioning

Designers frequently need to try several versions of an engineering design before decide upon the best choice. This evolution is usually nonlinear, what means that the database system must manage alternative versions of the same object.

The system can save storage space if a version of an object is represented as a *delta* from a slightly different previous version [1].

Kim *et alii* [3] adopted the division in *transient* and *working* versions for the ORION object-oriented database system.

A transient version can be updated/deleted by the user who created it. A new transient version can be derived from an existing one, which is 'promoted' to a working version.

A working version is considered stable and not updateable, but it can be deleted by its owner. Furthermore, a transient version can be derived from a working version and can be 'promoted' to a working version by either by the user or by the system. There is one working version of each portion of the design. This is the way the system keeps track of the current view of the design.

CONCLUSION

CAD data management, usually performed by dedicated mechanisms embedded in CAD systems, was approached in this paper from the point of view of database systems and object-oriented modeling.

Database treatment for CAD systems is desirable because the current trends on concurrent engineering lead to systems where not only design tasks access design data, but any application with its adequate view of the design.

Well-established database tools for business data management were described, and their lacks for CAD data manipulation were emphasized.

The most crucial issues on object-oriented database systems were outlined in a very open fashion.

Object-oriented techniques for CAD databases may help to meet the requirements of concurrent engineering. For instance, generalization of the abstraction mechanism to support arbitrary number of views may provide each application with an adequate view.

Some of the open issues on object-oriented databases seem to recall the unfinished battle among those who advocate the extension of the relational paradigm against those who claim for a completely new approach. Since relational databases are well-established tools, it seems to be natural to extend this approach. Even so, questions like how to treat a long CAD transaction without causing obstruction to any of the users remain unsolved.

REFERENCES

- [1] J.V. Joseph, S. Thatte, C. Thompson and D. Wells, "Object-Oriented Databases: Design and Implementation," *Proceedings of the IEEE*, vol. 79, no. 1, January 1991, pp. 42-64.
- [2] S. Heiler, U. Dayal, J. Orenstein and S. Randke-Sproull, "An Object-Oriented Approach to Data Management: Why Design Databases Need It," *24th ACM/IEEE Design Automation Conference*, Miami Beach, FL, 28 June-1 July 1987, pp. 335-340.
- [3] W. Kim, J. Banerjee, H. Chou and J. Garza, "Object-Oriented Database Support for CAD," *Computer Aided Design*, vol. 22, no. 8, Oct. 1990, pp. 469-479.