Applying a Semantic Structural Model for Engineering Design

Kincho H. Law and Thierry Barsalou

(To appear in the 1989 ASME International Computers in Engineering Conference)

TECHNICAL REPORT
Number 11

May, 1989

Stanford University
Applying a Semantic Structural Model for Engineering Design

Kincho H. Law
Civil Engineering Department
Stanford University
Stanford, CA 94305

Thierry Barsalou
Section on Medical Informatics
Stanford University
Stanford, CA 94305

Abstract

Object-orientation has recently emerged as an active area of engineering database research. In engineering design, one deals with large sets of independent but interrelated objects. For an engineering design database, the system must be able to not only effectively manage the design data, but also model the objects composing the design. The database management system needs to have some knowledge of the intended use of the data, and must provide an abstraction mechanism to represent and model the objects. This paper discusses an ongoing investigation of applying a structural model for engineering database design.

1. Introduction

In engineering design, one deals with large sets of independent but interrelated objects. These objects are specified by data. One way to store and organize these data items in a systematic and consistent manner is through the use of a database management system. Using a database to store and describe engineering data offers many benefits [17]:

- Ability to store or access data independent of its use
- Ability to represent (complex) relationships among the data
- Control of data redundancy
- Maintenance of data consistency and integrity
- Enhanced development of application software
- Support of file manipulation and report generation

To maximize these benefits, a database management system, besides effectively managing the data, needs to have some knowledge of the intended use of the data. That is, the formal structure or model used for organizing the data must capable of depicting the important relationship among the data and must be able to facilitate the maintenance of these relationships. Furthermore, the structure should be sufficiently flexible to allow a variety of design sequences and to aid an engineer to understand the design.

Traditional relational database systems provide many interesting features for managing data; among them are the capabilities of set-oriented access, query optimization and declarative languages. More importantly, from the user point of view, the relational model is completely independent of how data are physically organized. The relational model presents data item as records (tuples) which are organized in 2-dimensional tables (relations), and provides a manipulation language (relational calculus) to combine and reorganize the tables or relations for processing. The relational approach is simple and effective, particularly for business record-oriented data processing. Nevertheless, there exists a “semantic” gap between the relational data model and engineering design application. The relationships among the data items describing an engineering design are often complex. The lack of abstraction mechanisms makes the relational model inadequate for defining the semantics of an application and for maintaining interdependency of related data items. Furthermore, the traditional set-oriented relational structure does not support the engineering views of the data. The engineering users have to supply all the intensional semantics in order to make the data useful.

Object-orientation has been an active area of engineering database research. "Object-oriented" data models have been proposed to increase the modeling capability, to provide richer expressive concepts and to incorporate semantic content of engineering data. Commercial object-oriented database systems are also available [18, 10]. Object-oriented database research follows the development of object-oriented programming languages and their success in modeling complicated applications [11, 6]. In a design process, an engineer often approaches a design in terms of the components (objects) that comprise the design, and the operations (methods) that manipulate the components. A database system that supports the object-orientation of the design process can certainly enhance the interactions between the engineers and the system. The objective is to reduce the semantic gap between complex engineering design process and the data storage supporting the process.

It should be noted that an object-oriented data model does not necessarily imply that the object-oriented paradigm need to be explicitly implemented inside the database system. In engineering modeling and design, the information that an object represents is often shared by various applications having different views of the data [5, 7, 18, 8]. Data sharing is therefore as important as object-oriented access. Wiederhold has argued that storing objects (explicitly) in object format is not desirable, particularly if the objects are to be shared [19]. Barsalou and Wiederhold have proposed an approach, based on the structural data model, that permits object-oriented access to information stored in a relational database; information which in turn can be shared among different applications [1]. This approach of managing objects in a relational framework and its application for engineering modeling and design represent the subject of discussion in this paper.
2. The Structural Data Model

Many semantic data models have been influenced by the semantic network concept in artificial intelligence. The manageability of a complex application, which has too many relevant details to be intellectually represented, can be simplified by decomposing the model into a hierarchy of abstractions. Abstraction concepts are used so that the unnecessary details can be suppressed; only that information pertinent to the problem is emphasized. Focusing on a few selected important pieces of information enables an engineer to deal with complex design entities and their relationships effectively.

The structural data model that is used in this investigation is an extension of the relational model [18, 9]. In essence, the structural model augments the relational model by capturing the knowledge about the constraints and dependencies among the relations in the database. This section reviews briefly the data model. For more detailed description of this model, the reader is referred to References [18, 9, 2, 4].

The primitives of the structural model are the relations and the connections describing the relationships among the relations. The connection between two relations, R₁ and R₂, is defined over a subset of their attributes X₁ and X₂ with common domains. Two tuples, t₁ ∈ R₁ and t₂ ∈ R₂, are connected if and only if the connecting attributes in t₁ and t₂ match. There are three basic types of connections, namely, reference, ownership and subset. (There is a fourth connection type termed identity, which is not being considered in this paper [20].) These connections are used to define the relationships between the relations and to specify and maintain structural integrity constraints among the relations.

A reference connection between a referencing relation R₁ and a referenced relation R₂ specifies that following constraints:
1. Every tuple in R₁ must either be connected to a referenced tuple in R₂ or have null values for its attributes X₁.
2. Deletion of a tuple in R₁ requires deletion of its referencing tuples in R₁, or assignment of null values to attributes X₁ of all the referencing tuples in R₁.

The reference connection is used to refer to concepts which further describe a set of related entities. This connection type is useful for representing the notion of abstraction.

An ownership connection between two relations R₁ and R₂ specifies the following constraints:
1. Every tuple in R₁ must be connected to an owning tuple in R₁.
2. Deletion of an owning tuple in R₁ requires deletion of all tuples connected to that tuple in R₂.

The ownership connection describes the dependency of the owning tuples on a single owner tuple. Aggregation, which is useful for modeling "part-component" relationship, can be modeled using the ownership connection.

A subset connection between two relations R₁ and R₂ specifies the following constraints:
1. Every tuple in R₂ must be connected to one tuple in R₁.
2. Deletion of a tuple in R₁ requires deletion of the connected tuple in R₂.

The subset connection links general classes to their subclasses. Generalization (and specialization), which is useful for modeling alternatives or "is-a" type relationship, can be modeled using the subset connection.

3. An Example of CSG Representation

The Constructive solid geometry (CSG) model provides a simple and interesting example to illustrate the use of a semantic data model for engineering design application. The CSG representation of a solid object is a boolean tree where the root is a solid object and the leaves are primitive volumes. A solid object is represented by constructions or combinations using regularized set operations (such as union, intersection, difference, or complement) of object components. A CSG model can be described by the following BNF grammar [15, 14]:

- **Objects** ::= **Primitive** | **Composite Object**
- **Primitive** ::= **Block** | **Cylinder** | **Sphere** | ...
- **Composite Object** ::= **object** | **set operation**
- **set operation** ::= **union** | **intersection** | **difference** | **complement**

Based on these definitions, the relations corresponding to an example of a wheel represented as a CSG solid model are constructed as shown in Figure 1. Note that, in a CSG model, an object is defined as a composition of other objects as components, which are in turn composed of other objects as components etc. While the relations constructed as shown in Figure 1 cannot express the geographic information, there are inherent relationships among these relations with regard to the physical CSG model.

Among all the relations shown in Figure 1, the relation OBJECT is the key table to which all other relations except the relation MOTION_ARGUMENT refer or connect. For the relation OBJECT, the category of an object can be specified as either a primitive, a combined object or a moved object. For the relation COMBINED_OBJECT, the two attributes L_CHILD and R_CHILD are used to specify the subobjects which correspond to some tuples in relation OBJECT. Similarly, the attribute value M_Ω of relation MOVED_OBJECT specifies the object to be moved and corresponds to some object instance existing in the relation OBJECT. That is, there exists a recursive relationship among the entities OBJECT, MOVED_OBJECT and COMBINED_OBJECT.

It is reasonable to assume that, in the hierarchically structured CSG model, all insertions and deletions are initiated from the relation OBJECT. When inserting a tuple in the relation OBJECT, the tuple must properly connect "to" or "from" the relations PRIMITIVE, COMBINED_OBJECT or MOVED_OBJECT. Proper actions need to be taken with regard to those relations when an object is inserted in or deleted from the model. That is, there are certain semantic constraints and dependencies that need to be maintained. It is not possible in many relational database systems to express a constraint that requires the database to be updated as a result of applying the constraint.

The constraints and dependencies that are imposed by the CSG model can be defined using the structural data model. Figure 2 shows a structural model for the CSG representation. The connections specify the semantic information relevant to the constraints and dependencies among the relations. The specialization of the relation OBJECT to the relations PRIMITIVE, COMBINED_OBJECT and MOVED_OBJECT as well as the specialization of the relation PRIMITIVE to the relations BLOCK, CYLINDER, SPHERE can be represented using the subset connections. Similarly, the relation MOTION_ARGUMENT is referenced by the relations MOVED_OBJECT and PRIMITIVE, defining the locations and orientations of the objects.

Due to the recursive nature of objects, all objects (except the root object in the CSG tree) in the relation OBJECT are connected to the relations COMBINED_OBJECT or MOVED_OBJECT. Reference connections are used to relate the relation OBJECT and the relation COMBINED_OBJECT through the attributes L_CHILD and R_CHILD. Similarly, a reference connection is used to relate the relation OBJECT and the relation MOVED_OBJECT through the attribute M_Ω. In addition, we introduce a self-reference connection for the relation OBJECT, referencing the source attribute PARENT_ID with the destination attribute OBJECT. To properly maintain the hierarchical object-component relationship in the CSG tree, we adopt the constraint that the deletion of a referenced tuple in a referenced relation requires deletion of its referencing tuples in the referencing relation. With the reference and self-reference connections defined as shown in Figure 2, it is easy to observe that deletion of the root object from the CSG tree leads to deletions of all its subobjects from the database, based on the constraint definitions of the reference and subset connections.
Figure 1: A Relation Database for the CSG model

Figure 2: A structural model for CSG representation
Consider the deletion of an object, which is a subobject referenced by the attributes L_CHILD or R_CHILD in the relation COMBINED_OBJECT, from the relation OBJECT. Such a deletion is not allowed without prior modification to the parent object because the parent object will no longer be a combined object. That is, a strong “object” type checking is being defined by the connections. The enforcement of constraints and dependencies, if desired, can be delayed, depending on the situation and the specific implementation of the data model. The issues of enforcing integrity constraints remain an interesting research subject for investigation [13].

4. Views and Objects

Engineering design requires multiple representations of the design artifact and its components. Furthermore, as the design process progresses from conceptualization to design, the way to represent the design is constantly changing and modified. For an integrated design system to be effective, the database system must be able to accommodate the “growth” of the design and to support a wide variety of design representations. In essence, the database system need to support multiple views to satisfy the requirements of different applications.

As an example, Figure 3 shows different views of a simple beam. During the schematic design phase, the beam may be simply viewed as a rectangular component. The beam may eventually be designed as an I-shape steel beam, an encased steel beam or a concrete beam etc., depending on the type of building structure selected. On the other hand, for structural analysis, the beam is commonly modeled as a linear (line) element for which only the information about the center-line and the properties of the beam are needed. A view-object facility should be provided to allow the engineer to select the object information pertinent to the design task and to ignore the irrelevant details.

An architecture for combining the concepts of views and objects has been proposed by Wiederhold [19]. The architecture consists of three basic components: (1) a set of base relations; (2) a set of view-object generators; and (3) a set of view-object decomposers and archivers.

The schematic diagram of the object interface with a database system is shown in Figure 4. The view-object generators extract the data corresponding to an object from a relational database system and assemble the data into object instances based on the definition of the object specified in terms of the connections of the structural data model [2, 1]. Multiple layers of objects can be defined in that an object can be expressed in terms of other objects and yet shared by other objects.

A prototype system (PENGUIN) for this object-based architecture is being implemented by Barsakou [3]. The schematic diagram of the system architecture is shown in Figure 5. The system architecture consists of three basic components:

1. The object generator maps relations into object templates where each template can be complex combination of join (combining two relations through shared attributes) and projection (restricting the set of attributes of a relation) operations on the base relations. An object network groups together related templates, thereby identifying different object views of the same database. The whole process is knowledge-driven, using the semantics of the database structure.

2. The object instantiator provides nonprocedural access to the actual object instances. A declarative query specifies the template of interest. Combining the database-access function and the specific selection criteria, the system automatically generates the relational query and transmits it to the DMBS, which in turn transmits back the set of matching relational tuples.

3. The object decomposer maps the object instances back to the base relations. This component is invoked when changes to some object instances need to be made persistent at the database level. An object is generated by collapsing (potentially) many tuples from several relations. Similarly, one update operation on an object may result in a number of update operations on the base relations.

One of the key benefits of this object interface to a relational system,
Figure 4: Multiple view-objects for sharing information stored in a relational database.

Figure 5: PENGUIN's architecture: An object-based layer on top of a relational DBMS.
besides information sharing, is that any new attributes and/or relations added to the underlying database do not affect the object definitions. Conversely, changes in the definition of any objects do not affect the schema of the underlying database. Besides serving as an object-based management system, the prototype system PENGUIN is also designed to support coupling with expert system applications.

5. Summary and Discussion
In this paper, we discuss the potential application of the structural data model for engineering modeling and design. The connections of the structural data model properly define the constraints and dependencies that are needed to maintain the integrity structure of the hierarchical CSG representation of a solid model. It is shown that the structural data model can serve as the basis for the development of an object interface to a relational database system, supporting multiple views and objects. The architecture of a prototype system has also been briefly described.

Many advantages of this view-object approach can be identified:
- Provide multiple views of the stored information about an engineering model or design.
- Provide a mechanism for storing objects, independent of one specific view or application.
- Eliminate the data redundancy since the information about an object is shared (but not duplicated).
- Maintain integrity of the data for a given object based on the structural constraints as specified by the connections.
- Support growth of a design because of the independence between the object definitions and the database schema.

These advantages are very important for an engineering object-based management system. The potential application of this prototype system, which was originally developed for medical information management, for engineering design application will be a subject of our on-going investigation.

Acknowledgments
The authors are grateful to Professor Gior Wiederhold of the Computer Science Department at Stanford University for his inspiration and encouragement. The work by Dr. Thierry Barsalou was supported by the National Library of Medicine under Grant R01-LM04836, DARPA under Contract N39-84-C-0211 for Knowledge Base Management System, and Digital Equipment Corporation under the Quantum project. The work by Professor Kincho H. Law was supported by the Center for Integrated Facility Engineering at Stanford University.

References