CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

Managing Design Information in a Shareable Relational Framework

by

Kincho H. Law

TECHNICAL REPORT
Number 60

January, 1992

Stanford University

If you would like to contact the authors please write to:

c/o CIFE, Civil Engineering,
Stanford University,
Terman Engineering Center
Mail Code: 4020
Stanford, CA 95305-4020

Managing Design Information in a Shareable Relational Framework

Kincho H. Law
Associate Professor of Civil Engineering
Stanford University, Stanford, CA 94305-4020
e-mail:law@cive.stanford.edu

Abstract

The effectiveness of integrated engineering design systems depends on efficient access and expressive presentation of data. Such data may be stored in databases, but is rarely in the form that is suitable for access from engineering design software. In this report, we describe a view object approach for accessing and storing engineering information in a database management system. The object management system can serve as an interface such that applications are built in object-oriented environment and data are provided from relational databases. We propose to extend this view object paradigm to include CAD graphic information as well as to provide a mediator for monitoring design changes.

Keywords: Object-based management, relational database, engineering design, information sharing, structural data model, view object

Acknowledgment

This work is a joint research effort by the author and Professor Gio Wiederhold of Computer Science and is partially sponsored by the Center for Integrated Facility Engineering (CIFE) at Stanford University. This report is a summary of the research effort for the following CIFE's seed research projects:

- Data Modeling Issues for Integrated Structural Design
- A Formal Approach for Managing Design Objects in a Shareable Relational Framework

The author would like to thank Professor Gio Wiederhold, Dr. Arthur Keller and Dr. Thierry Barsalou for patiently answering many of the questions about the structural model and the concept of view-object and to Niki Siambela, Walter Sujansky, David Zingmond, Jennifer Wang and Matsushima Toshiyuki for the implementation effort. The author has also benefited from various discussions with Mr. Paul Scarponcini of McDonnell Douglas Built Environment Technologies on graphic CADD systems and with Dr. Keith Hall of Stanford University on the issues concerning change management for design databases.

All opinions expressed in this report are those of the author and do not necessary reflect the views of the Center for Integrated Facility Engineering.

Contents

Abstract i

Acknowledgment ii

Table of Contents iii

List of Figures iv

1 Introduction 1

2 Abstractions in Engineering Design 3

3 The Structural Data Model 5

4 Object-Oriented Applications and Databases 8

5 Architecture of the Object Management System 10

5.1 Object Generation 11

5.2 Object Instantiation 13

5.3 Object Decomposition 16

6 An Object Management System for Integrated Engineering Design Environment 20

6.1 View-Objects and Design Abstractions 20

6.2 Integration with Object-based Graphic System 22

6.3 Change Management with View Object Paradigm 23

7 Summary 24

References 24

List of Figures

1 Example of a Building Database Schema using the Structural Data Model 6

2 An Example of "Supporting" and "Supported-by" Relationships Implemented Through Structural Connections 8

3 Definition of a Joint Connection 8

4 Multiple View-Objects Sharing Information Stored in a Relational Database 11

5 System Architecture 12

6 Utility Modules of the Object Management System 13

7 Control Flow during Object Definition 14

8 Definition of an Object Template 15

9 Control Flow during Object Instantiation 16

10 Instantiation of an Object Template 17

11 Defining Valid Updates on an Object Instance 18

12 Control Flow during Object Decomposition 19

13 Multiple Views and Representations of Design Objects 21

14 Linking a View Object to Graphic Library 22

1 Introduction

In building design, we deal with large sets of independent but interrelated objects. These objects are specified by data. The data items describe the physical components (for example, columns, beams, slabs) and the topological aspects of the design (for example, member and joint connectivities). The design data need to be stored, retrieved, manipulated, and updated, during all phases of analysis, design, and construction of the project. An efficient data management system becomes an indispensable tool for an effective integrated computer aided analysis and design system.

Using a database to store and describe engineering data offers many benefits [38]. Some of these benefits include:

- Ability to store and access data independent of its use, so that the data can be shared among the participants
- Ability to represent relationships among the data, so that dependencies are documented
- Control of data redundancy, so that consistency is enhanced
- Management of data consistency and integrity, so that multiple users can access information simultaneously
- Enhanced development of application software by separating data management function
- Support of file manipulation and report generation for *ad hoc* inquiry

To maximize these benefits, a database management system 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 be capable of depicting the relationships among the data and must 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 important, from the user point of view, the relational model is completely independent of how data are physically organized. The relational model presents data items as records (tuples) which are organized in 2-dimensional tables (relations), and provides manipulation languages (relational calculus and algebra) to combine and reorganize the tables or relations for processing. The relational approach is simple and effective, particularly for business data processing. However, a "semantic" gap exists between the relational data model and engineering design applications. The relationships among the data items describing an engineering design are often complex.

The lack of a layered abstraction mechanism in the relational model makes it inadequate for defining the semantics of applications and for maintaining the interdependencies of related data items. Furthermore, the traditional set-oriented relational structure does not support well the engineering views of the data. The engineering users have to supply all the intentional semantics in order to exploit the data. That is, traditional relational databases do not support the semantic expressiveness of the data needed by engineering design systems.

Object-orientation has gained much attention in computer aided design in recent years. Object-oriented systems help the user in managing related data having complex structure by combining them into objects. The use of objects permits manipulatation of the data at a higher level of abstraction. Object-oriented data models have been proposed to increase the modeling capability, to provide richer expressive concepts and to incorporate some semantics about engineering data. The main objective here is to reduce the semantic gap between complex engineering design process and the data storage supporting the process. In such a process, an engineer often approaches the 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-oriented nature of the design process can certainly enhance the interactions between the engineers and the system.

In engineering modeling and design, the information that an object represents is often shared by various applications having different views of the data. Data sharing is therefore as important as object-oriented access. However, most research in engineering databases focuses on object management for specific tasks but gives little attention to the shareability of the underlying information. Since object oriented systems entail early binding of data and their semantics, storing objects poses a problem when these objects are to be shared by multiple engineering design tasks. Another type of design task is likely to require another type of binding. Furthermore, the amount of information pertaining to an object grows as each design task requires different information about the object. As a design progresses, an object may become too complex to be efficiently managed. Thus, the benefits of understandability and naturalness of having objects are lost. Storing objects (explicitly) in object format is not desirable, particularly if the objects are to be shared [40].

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. The approach described in this report is intended to manage complex engineering objects in a shareable relational framework [3,25]. Relational model provides information sharing through the definition of views but it lacks the expressive power to represent complex design entities. The concept of views in the relational model can be used as a tool for providing sharing and abstraction in integrating the object and relational models [40]. The mapping between heterogeneous structures of the two models is performed by linking object attributes to correponding relation attributes. The objective is to permit object-oriented access to information stored in a relational database; information which in turn can be shared among different applications.

The proposed object management system is based on the semantic structural data model which augments the relational model with a set of *connections* [41]. These connections define relationships, constraints, and dependencies among data that are of interest to engineering modeling and design [23]. Besides being an effective database design tool, the structural data model can serve as the basis for the development of an object interface to a relational database system, supporting multiple object views for multiple design tasks and applications. The key benefit of the view-object interface is the flexibility to allow adding new attributes and relations to the underlying database as well as changing the definition of objects.

This report is organized as follows: Section 2 elaborates on the requirements for database support in engineering design applications. Section 3 reviews the semantic structural data model. Section 4 discusses various approaches in the integration of database and object oriented applications. The architecture of the object management system is described in Section 5. In Section 6, we discuss extending the object management system for computer aided engineering environment. Section 7 concludes this report with a short summary and discussion. While the examples given in this report are drawn mainly from building and structural design application, the basic concepts can be extended to other engineering domains.

2 Abstractions in Engineering Design

Practical engineering tasks have too many relevant facets to be intellectually represented through a single abstraction process. Manageability of an application can be achieved by decomposing the model into several hierarchies of abstractions. In general, an aspect of a building and its design can be described as a collection of objects or concepts organized hierarchically [9,12,13,16,21,33,35]. The description of a design project grows as it evolves. During the design, additional attributes may be added to the description of existing entitites; similarly, aggregated entities can be decomposed into their constituents. That is, during design, information is added to the hierarchy by refinement in a top-down manner or by aggregation in a bottom-up sequence. The concept of abstraction provides a means for defining complex structures as well as the semantic information about the data. Powell and Bhateja have defined some requirements for an abstraction model in structural engineering application [33] :

- The model must support several applications.
- The model should be in terms of well-defined entities, relationships and dependencies.
- The model must support the creation of abstractions for real structures; that is, it must allow all relevant features of a structure to be represented. In addition, the

concepts used in the model should be familiar to the users.

- The model should allow the level of details to be increased as the design of the structure is progressively refined.

- The model should be able to represent structures of various types.

A structural engineering database system must be capable of supporting such an abstraction model. In addition to enhancing the database design process, an engineering data model should also provide the integrity rules to ensure consistency among the entities.

A relationship is a logical binding between entities. There are three types of relationships that are commonly used: *association*, *aggregation* and *generalization*. *Association* relates two or more independent objects as a merged object, whose function is to represent the many-to-many relationships among the independent objects. Association can be used to describe multiple "member of" relationships between member objects and a merged object. *Aggregation* combines lower level objects into a higher level composite object. In general, aggregation is useful for modeling part-component hierarchies and representing "part of" relationships. *Generalization* relates a class of individual objects of similar types to a higher level generic object. The constituent objects are considered specializations of the generic object. Generalization is useful for modeling alternatives and representing "is a" relationships. These three relationship types, in particular aggregation and generalization, are supported by many semantic data models and have been widely used in computer aided building design research [5,12,13,17,20,21]. For instance, a joint connector can be represented as an association of several structural elements (such as beams and columns) and connecting plates, and carries some information about the connectors to be used. A staircase is an aggregation of many similar parts. A concept "beams" is a generalization of a variety of members supporting gravity loads.

These three relationships impose certain existential dependency among the object entities. For example, the joint information is only meaningful while the referencing beams and plates are part of the design. As another example, assuming that the entities "BEAM" and "COLUMN" are specializations of a generic entity "STRUCTURAL ELEMENT", existence of a "BEAM" or "COLUMN" instance requires that a corresponding instance also exists in the generic entity "STRUCTURAL ELEMENT". When an instance is deleted from the generic entity "STRUCTURAL ELEMENT", corrective measure should be taken to remove the corresponding instance in a specialized entity "BEAM" or "COLUMN"; as a result, consistency between the generic and the specialized entities can be maintained in the database. Dependency constraints of these three types of relationships have been examined in details [3,6,8,22,34].

Besides association, aggregation and generalization, other relationships, such as "connected to", "supported by" (or "supporting"), "influence" and "determinants", are also of considerable interests in building design applications [1,10,16,33]. While incorporating these types of relationship into a data model is desirable, dependencies among the entities participating in these relationships have not been formally defined. For instance, when a supporting element is removed from a structure, corrective measures should be imposed on the objects that it supports. On the other hand, removing a supported element, from the data management point of view, should have little effect on the corresponding supporting object. It is important for a database management system to ensure consistency among the building design data, and to reflect appropriately the semantic relationships among them.

To be general, a database must support all design activities, in addition to capturing the semantic knowledge of the data. For each activity, an engineer works with specific application abstraction of the building, rather than with a complete physical description. For example, in the analysis of a building structure, an engineer is primarily interested in the building frame in terms of the center line of the members, their physical properties and the stiffness of the connections. Other information, such as room spaces and wall partitions, can be ignored. The database system needs to support various abstract views of the information pertaining to a specific domain. The ability to support multiple views for satisfying the requirements of different applications is an important criterion in selecting an engineering data model.

3 The Structural Data Model

Choosing a good data model to represent design data and processes is a major step towards the development of an integrated design system. A data model is a collection of well-defined concepts that help the database designer to consider and express the static properties (such as objects, attributes and relationships) and the dynamic properties (such as operations and their relationships) of data intensive applications [7]. Despite the data management capabilities that it has acquired over the years, the relational data model lacks the ability to provide abstraction mechanism to distinguish semantically different relationships between relations. We thus use the relational data model only to capture the data values, and extend it with the *structural data model* to capture the semantic relationships among the data [39]. The semantics capture knowledge about constraints and dependencies among tuples in the database relations.

The primitives of the structural data model are the *relations* and the *connections* formalizing relationships among the relations [41]. The connection between two relations R_1 and R_2 is defined over a subset of their attributes X_1 and X_2 with common domains. We denote the key and nonkey attributes of a relation R as $K(R)$ and $NK(R)$, respectively. Two tuples, $t_1 \in R_1$ and $t_2 \in R_2$, are connected if and only if the connecting attributes in t_1 and t_2 match.

There are three basic types of connections, namely *ownership*, *reference*, and *subset* connections, that can be used to define the relationships among the entities.. An *ownership* connection between an owner relation R_1 and an owned relation R_2 describes the dependency of multiple owned tuples on a single owner tuple. The ownership connection embodies the concepts of dependency and aggregation, where owned tuples are specifi-

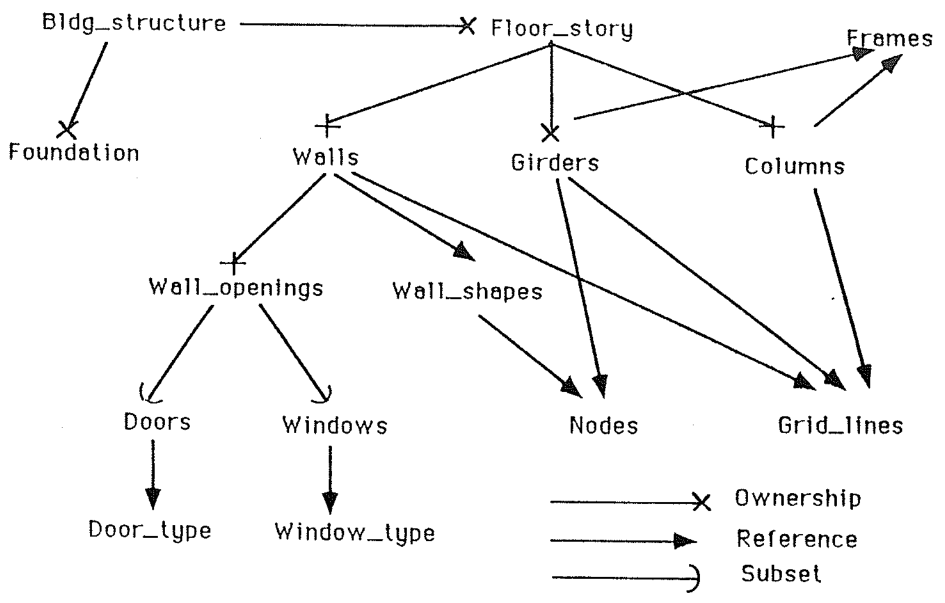


Figure 1: Example of a Building Database Schema using the Structural Data Model

cally related to and dependent on a single owner tuple. As an example, Figure 1 shows the composition of a building structure consisting of a foundation and a number of floor stories. The components exist if the building exists. This owner-component relationship is best represented using the ownership connection. This connection type specifies the following constraints:

1. Every tuple in R_2 must be connected to an owning tuple in R_1.
2. Deletion of an owning tuple in R_1 requires deletion of all tuples connected to that tuple in R_2.
3. Modification of X_1 in an owning tuple of R_1 requires either propagating the modification to attributes X_2 of all owned tuples in R_2 or deleting those tuples.

The ownership connection requires that $X_1 = K(R_1)$ and $X_2 \subset K(R_2)$; its cardinality is $1 : n$.

A *reference* connection between a primary (referencing) relation R_1 and a foreign (referenced) relation R_2 is useful for representing the notion of abstraction; it describes the dependency of multiple primary tuples on the same foreign tuple. The reference connection can be used to refer to concepts which further describe a set of related entities. Referring to the example shown in Figure 1, the doors reference to specific door types that are available. The door type cannot be cancelled if there exists doors using that door type. The reference connection specifies the following constraints:

1. Every tuple in R_1 must either be connected to a referenced tuple in R_2 or have null values for its attributes X_1.
2. Deletion of a tuple in R_2 requires either deletion of its referencing tuples in R_1,

assignment of null values to attributes X_1 of all the referencing tuples in R_1, or assignment of new valid values to attributes X_1 of all referencing tuples corresponding to an existing tuple in R_2.

3. Modification of X_2 in a referenced tuple of R_2 requires either propagating the modification to attributes X_1 of all referencing tuples in R_1, assigning null values to attributes X_1 of all referencing tuples in R_1, or deleting those tuples.

The reference connection requires that $X_1 \subset K(R_1)$ or $X_1 \subset NK(R_1)$ and $X_2 = K(R_2)$; its cardinality is $n : 1$. Association, which allows the relationship of n tuples in R_1 to m tuples in R_2, can be modeled with a combination of ownership and reference connections [41]. It should be emphasized that depending on a specific application, when enforcing the dependency constraints for a reference connection, one can modify the referencing attributes instead of delete the referencing tuple.

A subset connection between a general relation R_1 and a subset relation R_2 is useful for representing alternatives or "is a" type relationship; it links general classes to their specializations and describes the dependency of a single tuple in a subset on a single general tuple. For the example shown in Figure 1, an opening in a wall can either be a door or a window. Deleting a specific instance in the generic class of a wall opening must delete the corresponding instance existing in the subclass. The subset connection specifies the following constraints:

1. Every tuple in R_2 must be connected to one tuple in R_1.
2. Deletion of a tuple in R_1 requires deletion of the connected tuple in R_2.
3. Modification of X_1 in a tuple of R_1 requires either propagating the modification to attributes X_2 of its connected tuple in R_2 or deleting the tuple in R_1.

The subset connection requires that $X_1 = K(R_1)$ and $X_2 = K(R_2)$; its cardinality is $1 : 1$ or 0.

Besides supporting the three basic relationships of aggregation, generalization and association, the connections can also be used to define relationships such as "connected-to" and "supported-by", that are useful in engineering application. In the "supported-by" or "supporting" relationship, the supporting entity should not be removed unless all its supported entities no longer exist. For example, when a wall is removed from a design, the openings, such as windows and doors, located inside that wall have to be removed also. As another example, a column should not be removed unless the references from the components such as walls, beams, slabs that the column supports no longer exist. This dependency property can be modeled using the reference connection as shown in Figure 2.

One application of the "connected-to" relationship in structural engineering is for the description of a joint connecting structural elements. As an example, we can represent a joint connection that is described in an interactive modeling system for the design of steel framed structure (Steelcad) [36] : "The connected members and the joint are logically linked in the database:

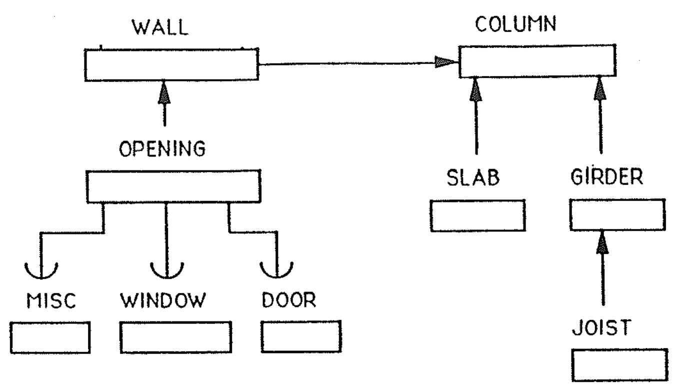


Figure 2: An Example of "Supporting" and "Supported-by" Relationships Implemented Through Structural Connections

- If a joint is removed, then the previously connected members are automatically restored, as they were before the connection was defined.
- If a member is removed, then all connections that it has with other members are also removed and the other members reappear accordingly."

As shown in Figure 3, this description of a joint can be modeled using the three basic connection types. We assume that a joint connection may consist of one or more connectors. Each connector references a structural member and a connecting element. That is, deleting a connector does not affect the connecting members. On the other hand, if a structural member or a connecting plate is removed, the connectors are also removed.

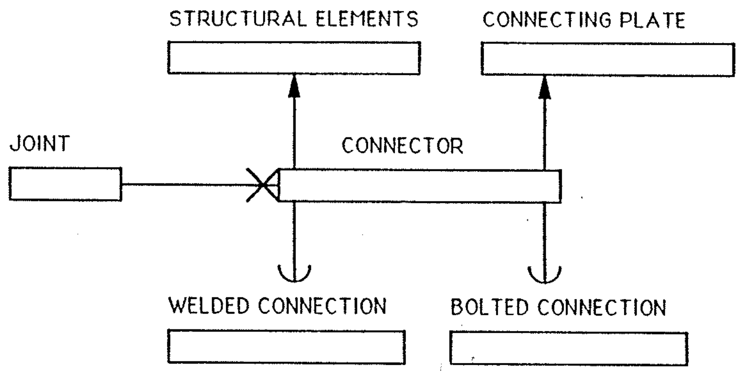


Figure 3: Definition of a Joint Connection

4 Object-Oriented Applications and Databases

Integration of object-oriented programs with databases would enable applications in an object-oriented environment to have shared, concurrent access to persistent storage.

There are two basic approaches to this integration. One approach is to use an object-oriented model for both the applications and the persistent storage [2,29,31]. The design objects would then be retrieved and stored as objects. Alternatively, one can use an object-oriented model for applications and the relational model for persistent storage [3,14,24,27,37]. The system is configured as a front-end system with an object-oriented model and conventional relational databases as back-end storage. Objects are retrieved by evaluating queries to database. In the former case, the sharability and multiplicity of views would suffer, whereas the latter case serves multiple users well. The cost in this case is an added overhead resulting from the mapping between the two models. Our approach follows the latter, trading efficiency for flexibility. However, the formality of the mapping approach we use identifies new data transmission optimizations, which will be important in the modern file-server to workstation system architecture [28].

There are two basic implementations in integrating object-oriented applications with relational databases. One is to map objects to relations [14,27,37], and the other is to map relations to objects [3]. In the first perspective, relation schemas are generated from the object schemas, i.e., types and their hierarchy; relations are used primarily for storing objects. In the second perspective, object schemas are defined from the relation schemas. Relations are the source for generating objects. This perspective is useful for building object-oriented applications on top of existing relational databases. We have adopted the second perspective in our work. The second perspective is compatible with the first one, since it can access relational databases defined in that manner. But now pre-existing relational databases can be accessed as well, and non-relational databases can be accessed if the structural model is used to define their update constraints.

When the design object information is stored in a relational database, we deal with entities that are more complex than single tuples or sets of tuples from a single relation. Quite frequently, an object is a hierarchical group of tuples comprising a single root tuple that defines the object, and one or more associated tuples that further describe the object's properties. Because of normalization theory, these dependent tuples reside in one or more relations distinct from the relation containing the root tuple. Even if such a structure is easily expressed relationally (through joins), it cannot be easily manipulated as a single entity, nor easily displayed in its natural hierarchical structure. We need to make such structures explicitly known to the system. Furthermore, as noted earlier, different users require different views of the information included in an object. Update anomalies and problems of redundancy would arise if the objects corresponding to the different views were to be stored explicitly. Last but not least, changes to the set of classes and to the inheritance can be made quite frequently at various stages of a design project. In contrast, if the objects were explicitly stored, then the schema would have to be changed accordingly.

There are three levels of object orientation for DBMS [11]:

1. Structurally object-oriented implies the capability of representing arbitrarily structured complex objects.

2. Operationally object-oriented implies the ability to operate on complex objects in their entirety, through generic complex object operators.
3. Behaviorally object-oriented implies the availability of classes, messages and data abstraction as in object-oriented programming.

The proposed object management system is structurally and operationally object-oriented.[1] The objective is to develop an object interface to the persistent storage that supports multiple object views for multiple design tasks and applications.

5 Architecture of the Object Management System

Our approach is to define and manipulate complex objects that are constructed from base relations. Our prototype implementation keeps a relational database system as its underlying data repository. Indeed, we believe that the relational model should be extended rather than replaced. The relational model has become a *de facto* standard and thus some degree of upward compatibility should be kept between the relational format and any next-generation data model.

An architecture for combining the concepts of views and objects has been initially proposed by Wiederhold [40]. Both the concepts of view and object are intended to provide a better level of abstraction, bringing together related elements, which may be of different types, into one unit. For example, to display a building frame, we bring together structural entities, such as beams, columns and connections. A set of base relations serve as the persistent database and contain all the data needed to create any specific view or object. First, an object template is specified in terms of the connections of the structural data model. Data is then extracted from the relational database, corresponding to a view-object and assembled into object instances. The mapping between heterogeneous structures of the two models is performed by linking object attributes to correponding relation attributes. However, objects have more complex structure than relations. An object may contain subobjects, thus having a nested structure. Nested objects differ from nested tuples since the type of an attribute may refer to other objects. The idea is not to store the objects directly in persistent form but rather to store their description, which is used later to instantiate objects as needed. Multiple layers of view-objects can be defined so that a view-object can be composed of subordinate view-objects and shared by multiple view-objects. Figure 4 depicts this notion of multiple view objects and their interaction with the underlying set of relations.

The basic schematic diagram of the system architecture is shown in Figure 5. The system provides an object-oriented interface by performing three basic operations: *object generation*, *object instantiation* and *object decomposition*. The prototype system includes 10 basic modules, as shown in Figure 6. In the following, we discuss briefly the three

[1]Research effort is proposed to extend the capability of the current system to support object-oriented languages such as *C++*. The system will then be able to provide the features as in object-oriented programming.

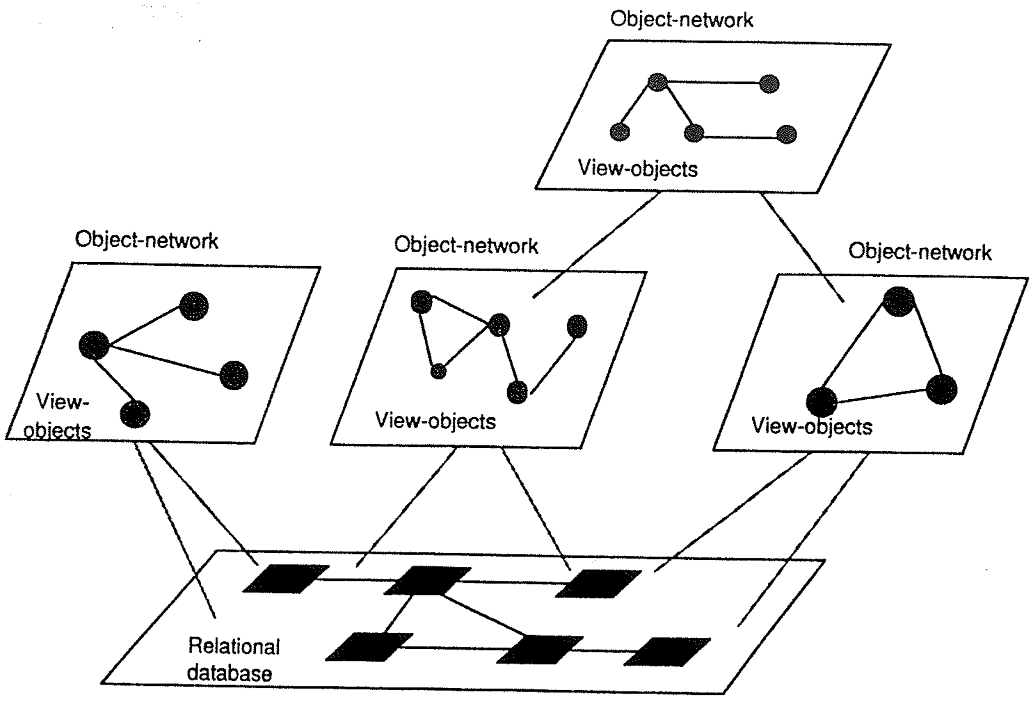


Figure 4: Multiple View-Objects Sharing Information Stored in a Relational Database

basic operations and the roles of the utility modules.

5.1 Object Generation

Each view-object is defined by an object template. The *object generator* maps relations into object templates that describe the structure and Data Access Function (DAF) for the objects. A DAF invokes join operations (combining two relations through shared attributes) and projection operations (restricting the set of attributes of a relation) on the base relations for the retrieval of the object data.

To define an object template the user first selects a pivot relation from a set of base relations. Further information required for generating a template would come from the connections described in the structural data model. The set of candidate relations and the tree structure they generate through the structural connections is called the *candidate tree*. Besides the pivot relation which describes the root of the tree, none, some, or all of the relations and their attributes contained in the set of candidate relations may be included in the object template. The relations selected are called the secondary relations. Secondary relations and their attributes (including those of the pivot relation) are selected from the set of candidate relations. Once an object template is defined, data access functions are derived to facilitate the data retrieval process.

The DAF can then be derived in an algorithmic manner to facilitate the data retrieval process. The key for the object is the same as the key of the pivot relation. Related

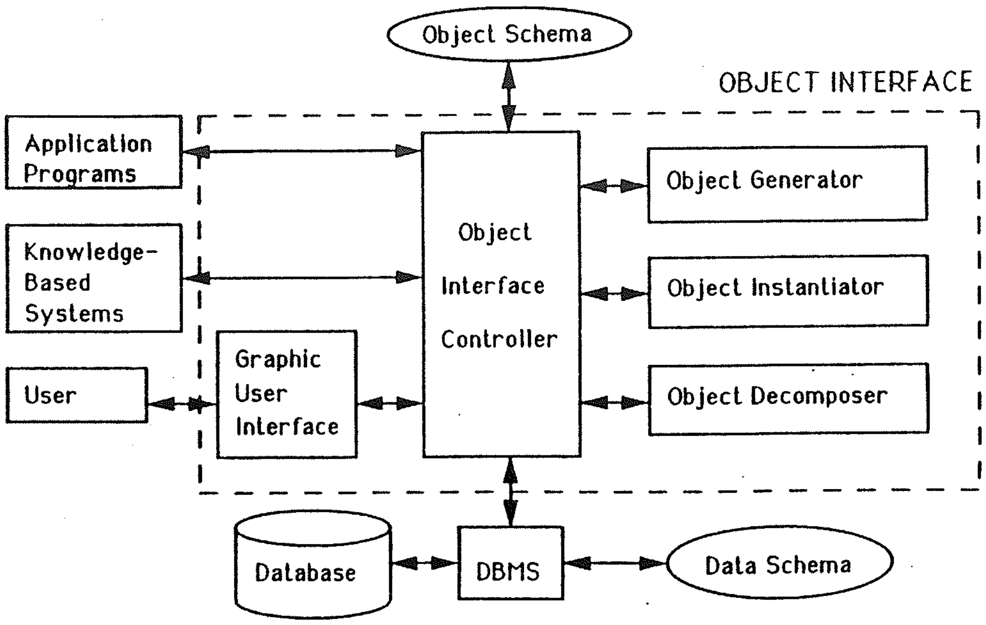


Figure 5: System Architecture

templates can be subsequently grouped together to form an object network, identifying a specific view of the relational database. The whole process is knowledge-driven, using the semantics of the database structure as defined by the connections of the structural data model.

The operations described above are carried out by the system modules as shown in Figure 7 and explained here :

- The *User/Programmatic Interface* is used to create object template and define the pivot and secondary relations for the object
- The *Structural Model Manager* is used to create and delete structural model connections and to store them in the database system. The Structural Model manager also performs consistency checking on the defined semantic connections
- The *Object Generation Manager* takes the structural model from the *Structural Model Manager*, obtains the pivot relation, generates the candidate tree containing the candidate relations for inclusion in the object, creates data access functions (DAF) and object template, and finally sends the object template to the *Object Schema Manager*
- The *Object Schema Manager* stores the template and assigns it to appropriate object network and schema
- The *User/Programmatic Interface* receives confirmation about the object template (i.e. a message indicating the success of object definition procedure)

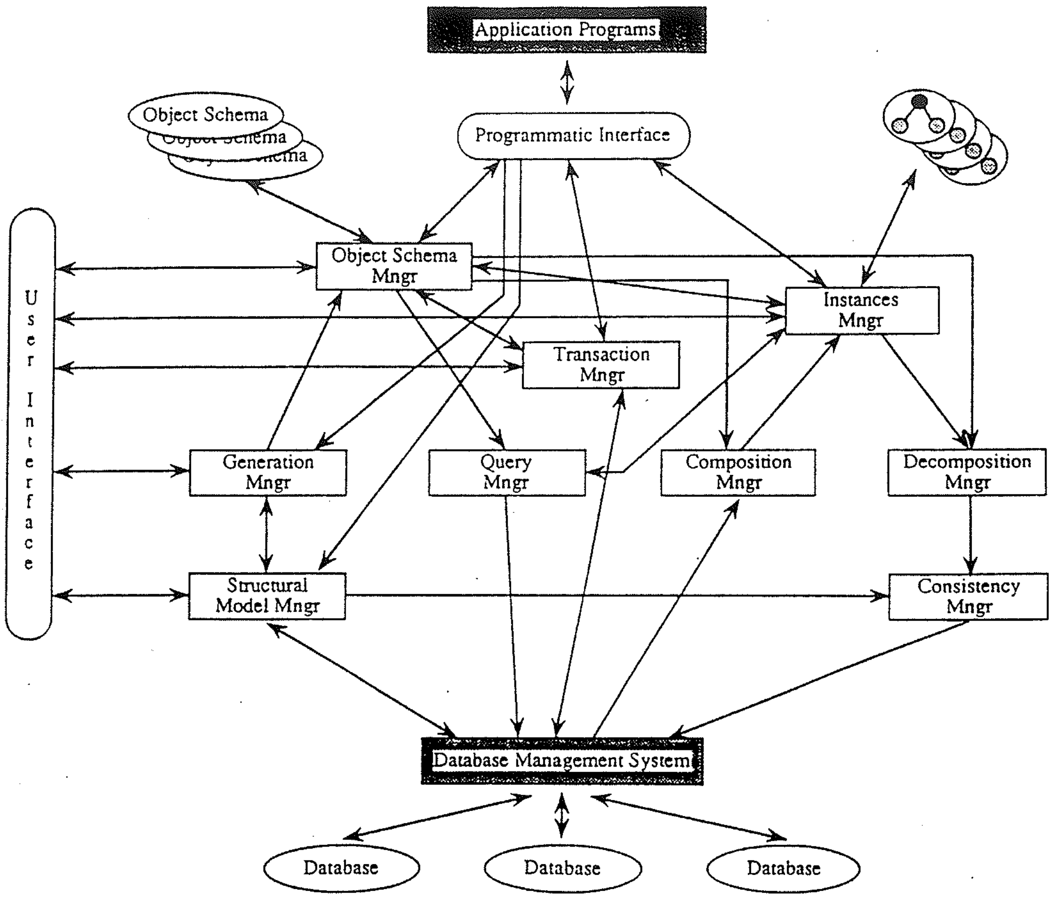


Figure 6: Utility Modules of the Object Management System

Figure 8 illustrates the definition of an object template "WALLS". Figure 8(a) shows the partial definition of a structural data model. To define the object template, we select the pivot relation "WALLS" and the candidate relations "WALL_SHAPES", "WALL_OPENINGS", "WINDOWS" and "DOORS" to be included in the template. For each relation, we can select the attributes to be included in the object template as shown in Figure 8(b) (although all key attributes required to perform the join operations during object instantiation will automatically be included.)

5.2 Object Instantiation

The *object instantiator* provides nonprocedural access to the actual object instances, performing all of the operations for information retrieval and manipulation that are necessary to instantiate selected objects according to an object template and display it in nested form. A declarative query stored with the object template, i.e., the DAF, specifies the relational projection and join operations needed to retrieve the data of interest. The DAF is further combined with user supplied selection criteria, resulting in a relational query transmitted to the relational database system. The matching relational tuples are returned and assembled into the desired object instances. This process is performed

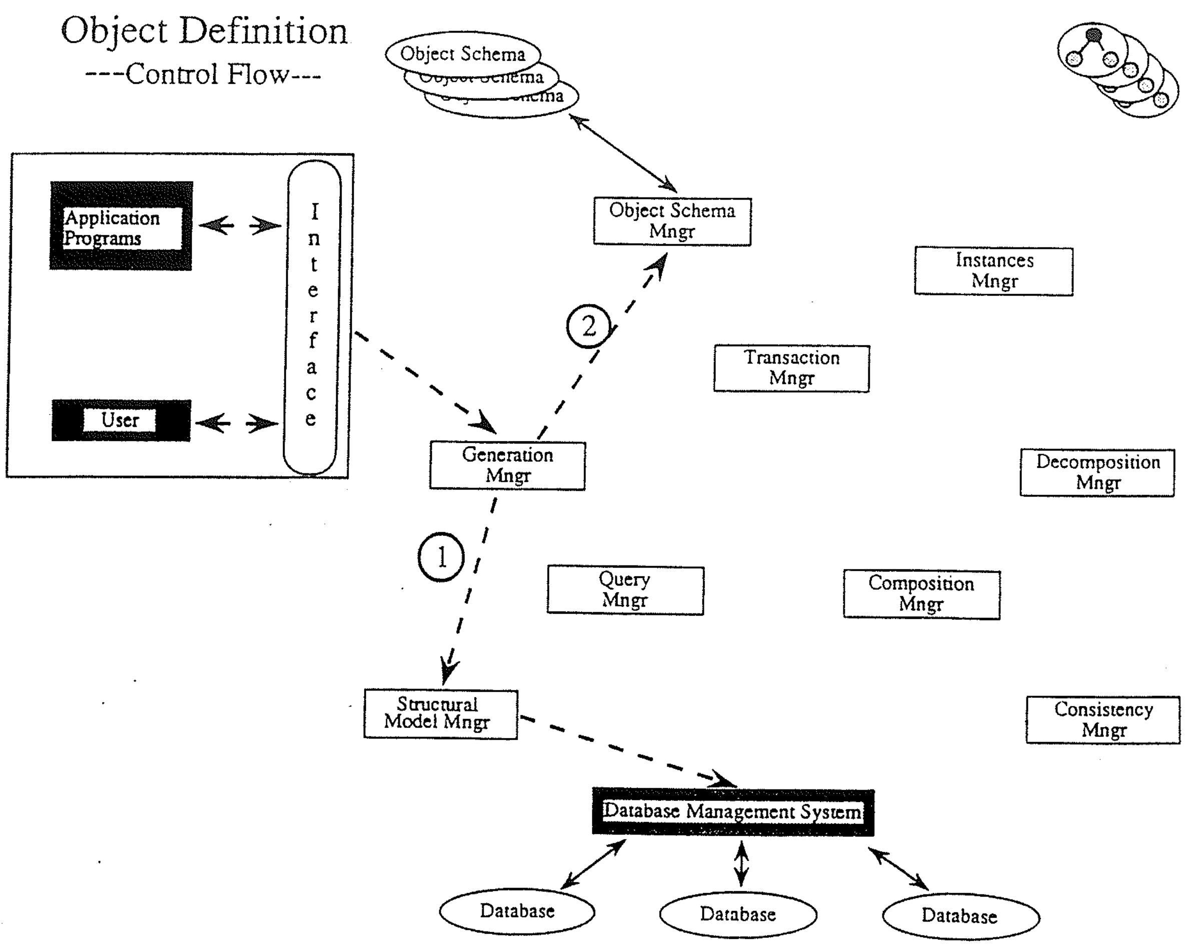
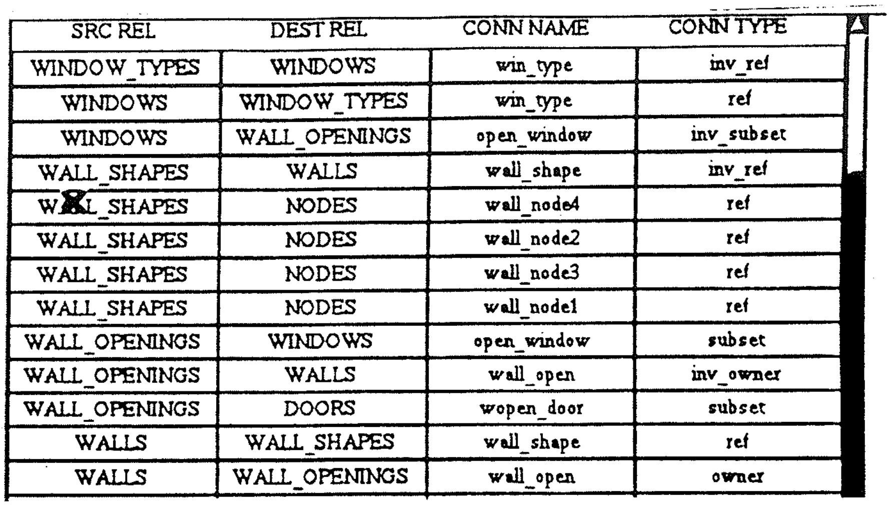


Figure 7: Control Flow during Object Definition

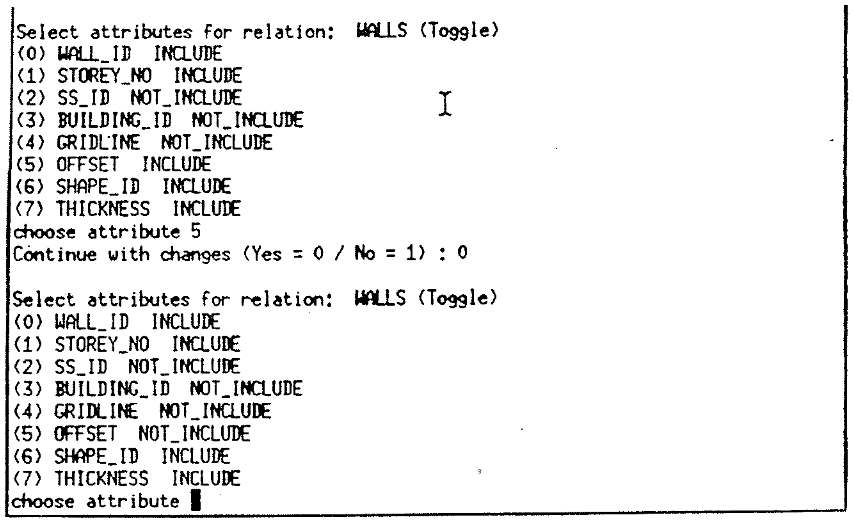
based upon semantics defined in the object template. Figure 9 shows the control flow among the system modules. The operations are carried out as follows:

- The *Programmatic Interface* receives a request for an object instantiation
- The *Object Schema Manager* identifies the object template and passes it to the *Query Manager* along with the selection clause defined in the request
- The *Query Manager* combines the object template and the selection clause, optimizes the query and sends it to the DBMS
- The *Composition Manager* receives the data returned and places it into a structured form, reflecting the proper abstraction type. The composition, as we call it, of the object instances is based on the object template (as defined by the *Object Schema Manager*)
- The *Instance Manager* receives the instances created, stores them in the instance pool and returns a reply to the *Programmatic Interface*



SRC REL	DEST REL	CONN NAME	CONN TYPE
WINDOW_TYPES	WINDOWS	win_type	inv_ref
WINDOWS	WINDOW_TYPES	win_type	ref
WINDOWS	WALL_OPENINGS	open_window	inv_subset
WALL_SHAPES	WALLS	wall_shape	inv_ref
WALL_SHAPES	NODES	wall_node4	ref
WALL_SHAPES	NODES	wall_node2	ref
WALL_SHAPES	NODES	wall_node3	ref
WALL_SHAPES	NODES	wall_node1	ref
WALL_OPENINGS	WINDOWS	open_window	subset
WALL_OPENINGS	WALLS	wall_open	inv_owner
WALL_OPENINGS	DOORS	wopen_door	subset
WALLS	WALL_SHAPES	wall_shape	ref
WALLS	WALL_OPENINGS	wall_open	owner

(a) An Example of Structural Data Model



```
Select attributes for relation:  WALLS (Toggle)
(0) WALL_ID  INCLUDE
(1) STOREY_NO  INCLUDE
(2) SS_ID  NOT_INCLUDE
(3) BUILDING_ID  NOT_INCLUDE
(4) GRIDLINE  NOT_INCLUDE
(5) OFFSET  INCLUDE
(6) SHAPE_ID  INCLUDE
(7) THICKNESS  INCLUDE
choose attribute 5
Continue with changes (Yes = 0 / No = 1) : 0

Select attributes for relation:  WALLS (Toggle)
(0) WALL_ID  INCLUDE
(1) STOREY_NO  INCLUDE
(2) SS_ID  NOT_INCLUDE
(3) BUILDING_ID  NOT_INCLUDE
(4) GRIDLINE  NOT_INCLUDE
(5) OFFSET  NOT_INCLUDE
(6) SHAPE_ID  INCLUDE
(7) THICKNESS  INCLUDE
choose attribute
```

(b) Selecting attributes for Inclusion

Figure 8: Definition of an Object Template

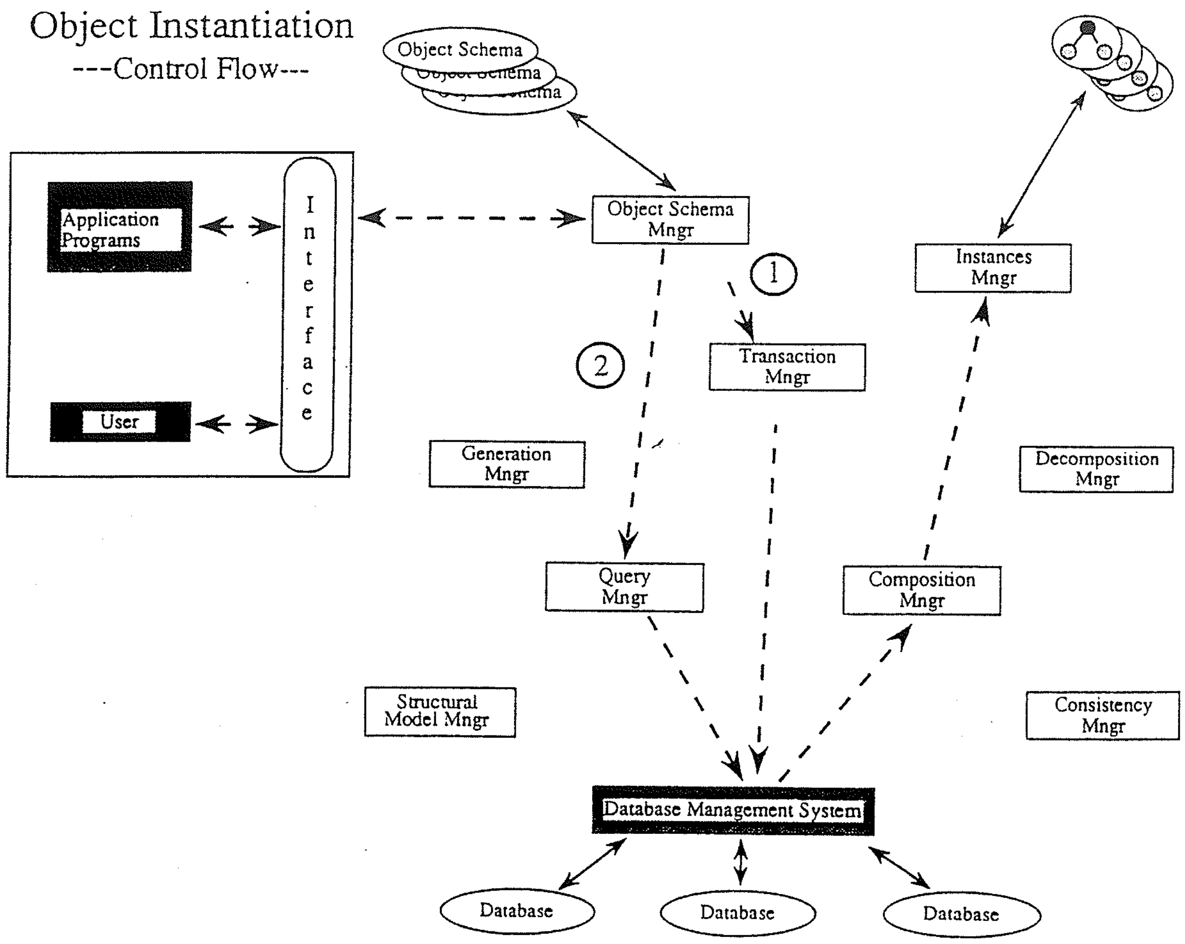
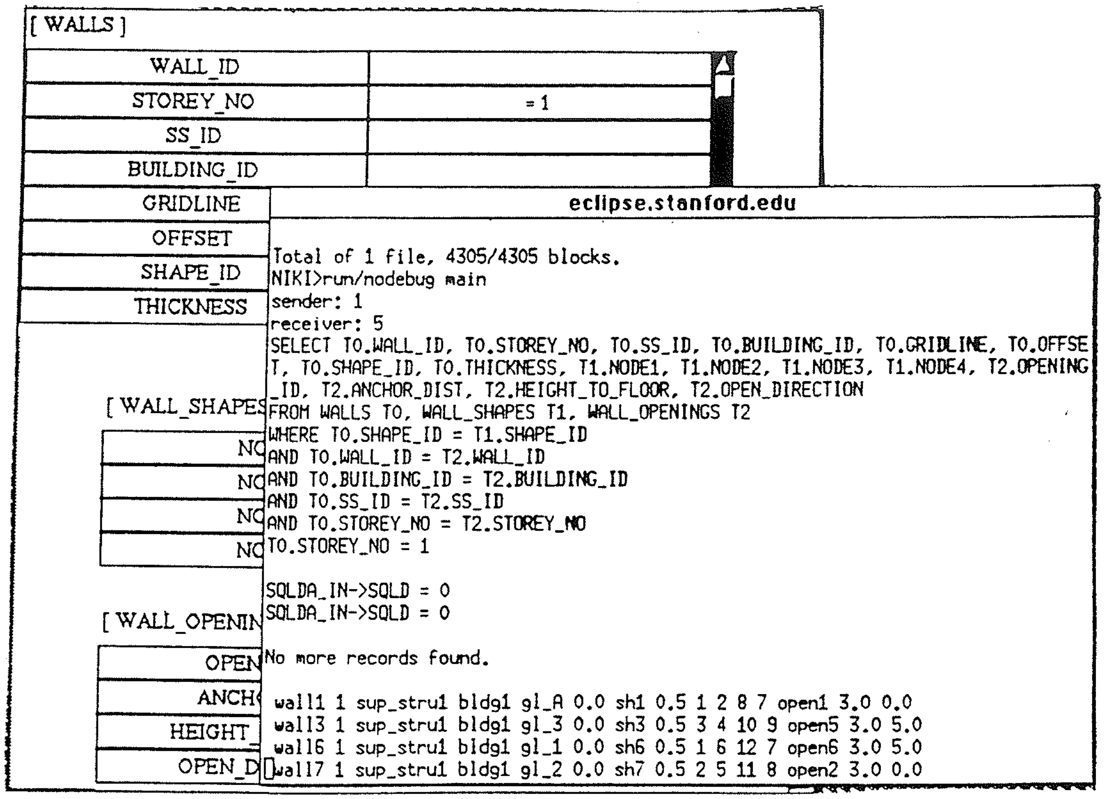


Figure 9: Control Flow during Object Instantiation

As an example, the data access function for a specified query and a resulting object instance are shown in Figure 10. Although it is not shown in the figure, it may be worth pointing out that only the selected attributes for the candidate relations are included in the object template.

5.3 Object Decomposition

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 collecting (potentially) many tuples from several relations. Similarly, one update operation on an object may result in a number of update operations on several base relations. Dependency constraints are enforced during these update operations to ensure the database consistency. These actions are based on the integrity rules imposed by the connections of the structural data model. For example, when "owning" tuples are deleted from a relation connected via an ownership connection, the structural model requires that all corresponding tuples in the "owned" relation also be deleted. An object template however is just like a view. It is defined via join, selection



[WALLS]

WALL_ID	
STOREY_NO	= 1
SS_ID	
BUILDING_ID	
GRIDLINE	
OFFSET	
SHAPE_ID	
THICKNESS	

[WALL_SHAPES

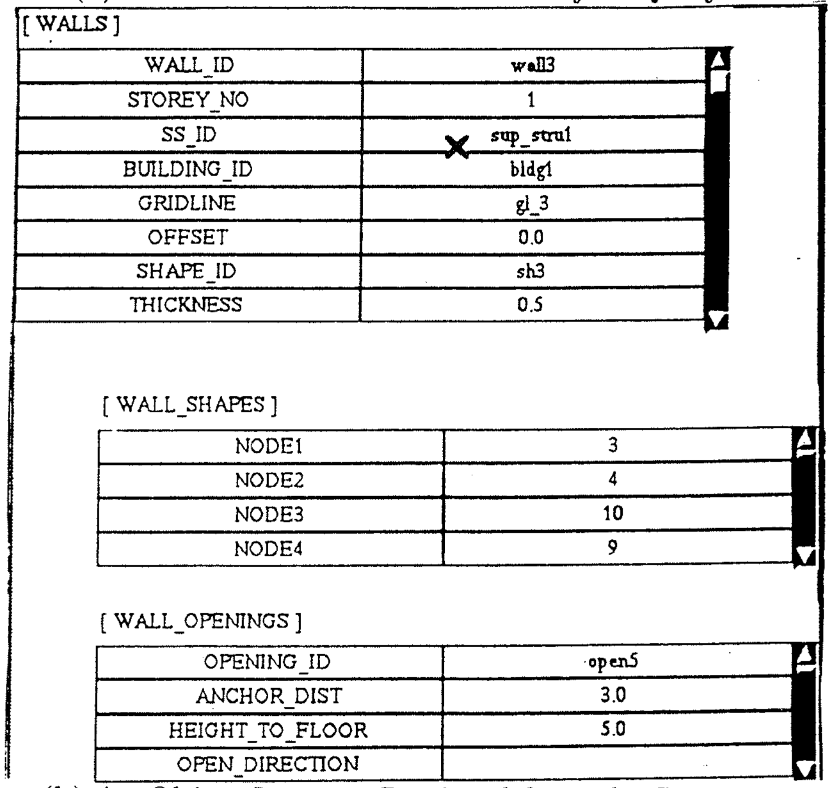
NO
NO
NO
NO

[WALL_OPENIN

OPEN
ANCH
HEIGHT
OPEN_D

```
eclipse.stanford.edu

Total of 1 file, 4305/4305 blocks.
NIKI>run/nodebug main
sender: 1
receiver: 5
SELECT TO.WALL_ID, TO.STOREY_NO, TO.SS_ID, TO.BUILDING_ID, TO.GRIDLINE, TO.OFFSE
T, TO.SHAPE_ID, TO.THICKNESS, T1.NODE1, T1.NODE2, T1.NODE3, T1.NODE4, T2.OPENING
_ID, T2.ANCHOR_DIST, T2.HEIGHT_TO_FLOOR, T2.OPEN_DIRECTION
FROM WALLS TO, WALL_SHAPES T1, WALL_OPENINGS T2
WHERE TO.SHAPE_ID = T1.SHAPE_ID
AND TO.WALL_ID = T2.WALL_ID
AND TO.BUILDING_ID = T2.BUILDING_ID
AND TO.SS_ID = T2.SS_ID
AND TO.STOREY_NO = T2.STOREY_NO
TO.STOREY_NO = 1

SQLDA_IN->SQLD = 0
SQLDA_IN->SQLD = 0

No more records found.

 wall1 1 sup_stru1 bldg1 gl_A 0.0 sh1 0.5 1 2 8 7 open1 3.0 0.0
 wall3 1 sup_stru1 bldg1 gl_3 0.0 sh3 0.5 3 4 10 9 open5 3.0 5.0
 wall6 1 sup_stru1 bldg1 gl_1 0.0 sh6 0.5 1 6 12 7 open6 3.0 5.0
 wall7 1 sup_stru1 bldg1 gl_2 0.0 sh7 0.5 2 5 11 8 open2 3.0 0.0
```

(a) Data Access Function for an Object Query



[WALLS]

WALL_ID	wall3
STOREY_NO	1
SS_ID	sup_stru1
BUILDING_ID	bldg1
GRIDLINE	gl_3
OFFSET	0.0
SHAPE_ID	sh3
THICKNESS	0.5

[WALL_SHAPES]

NODE1	3
NODE2	4
NODE3	10
NODE4	9

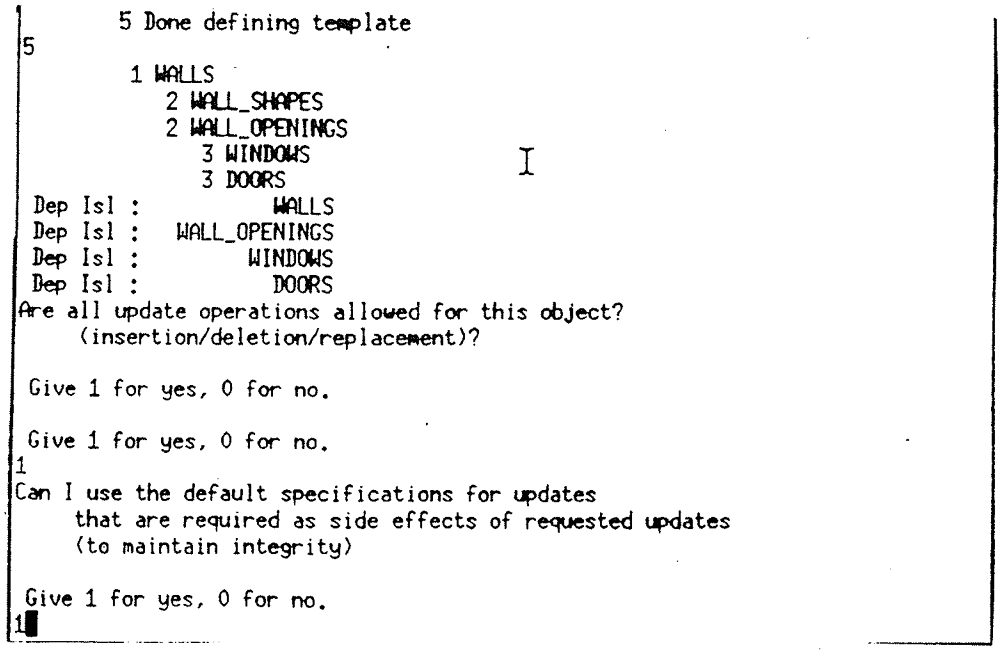
[WALL_OPENINGS]

OPENING_ID	open5
ANCHOR_DIST	3.0
HEIGHT_TO_FLOOR	5.0
OPEN_DIRECTION	

(b) An Object Instance Retrieved from the System

Figure 10: Instantiation of an Object Template

and projection operations. It thus inherits the well known problem of view updates [15]. Updating through views is inherently ambiguous, since a change made to the view can translate into different modifications to the underlying database. The different sets of modifications are called translators. Keller has shown that, using the structural semantics of the database, one can enumerate such ambiguities, and choose a specific translator at view-definition time [19]. Since the object templates are defined using join operations on the database relations, we are then facing the well-known problem of updating relational databases through views involving multiple relations. When creating a new object template, a simple dialogue, the content of which depends on the structure of the object template, allows the user to select one of the semantically valid translators. The chosen translator is then stored as part of the template definition. When an object instance is modified, the object decomposer will use this information to resolve any ambiguity completely and to update the database correctly. Details in the development of a translator for view-object update are described in Reference [4]. Example of a simple dialogue for the translator is shown in Figure 11.



```
        5 Done defining template
5
         1 WALLS
            2 WALL_SHAPES
            2 WALL_OPENINGS
               3 WINDOWS
               3 DOORS
 Dep Isl :           WALLS
 Dep Isl :   WALL_OPENINGS
 Dep Isl :         WINDOWS
 Dep Isl :           DOORS
Are all update operations allowed for this object?
     (insertion/deletion/replacement)?

 Give 1 for yes, 0 for no.

 Give 1 for yes, 0 for no.
1
Can I use the default specifications for updates
     that are required as side effects of requested updates
     (to maintain integrity)

 Give 1 for yes, 0 for no.
1
```

Figure 11: Defining Valid Updates on an Object Instance

Figure 12 shows the control flow for object decomposition. The basic operations are summarized as follows:

- The *Programmatic Interface* receives a request for updating one or several object instances in the object instance pool
- The *Instance Manager* invokes the *Decomposition Manager* and passes to it both the updated and original objects
- The *Decomposition Manager* decomposes the object into relational tuples, validates the update requests against constraints defined in the object template (based on

the information received from the *Object Schema Manager*) and delivers all the information to the *Consistency Manager*

- The *Consistency Manager* checks for global consistency of the updates with respect to the structural model for the entire database and sends the update information to the DBMS
- The DBMS updates the data in the base relations.

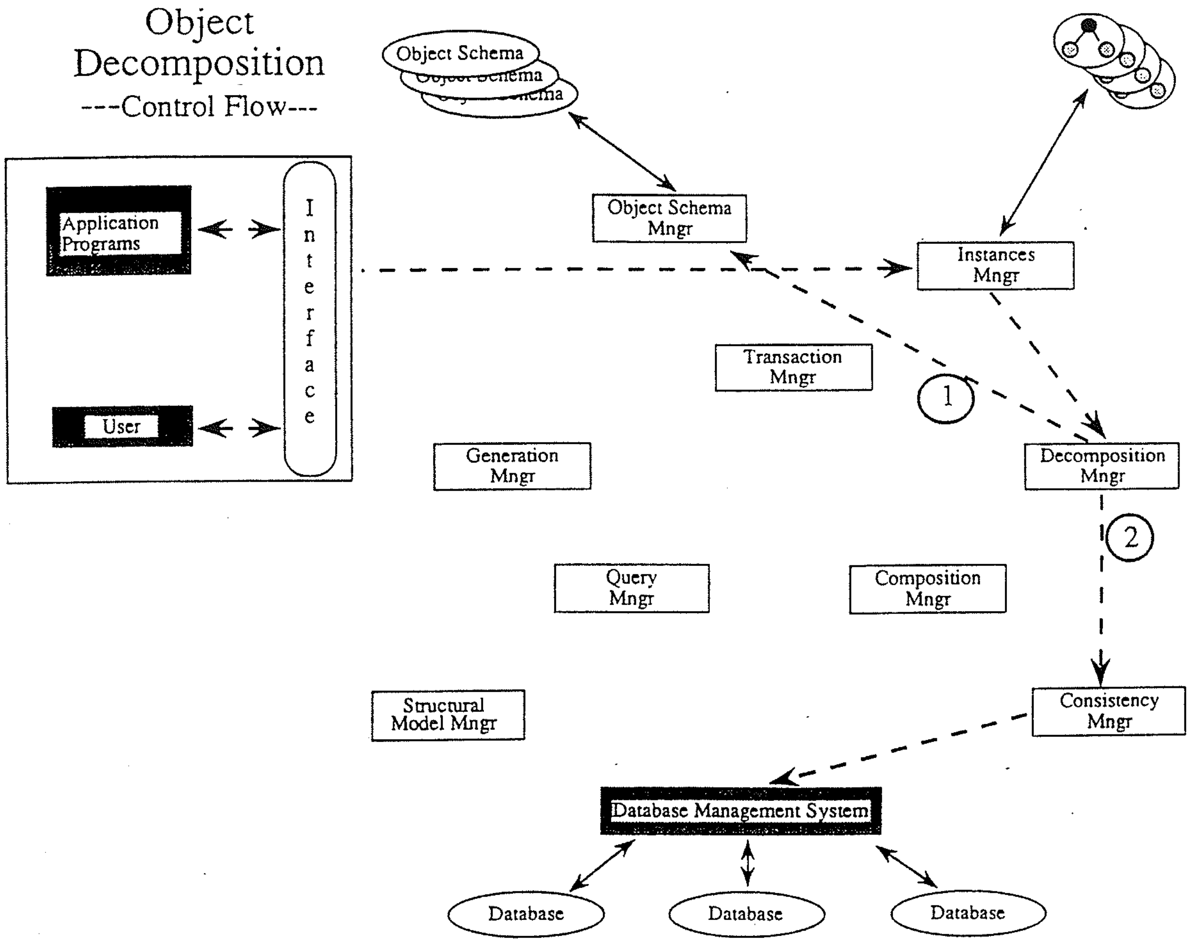


Figure 12: Control Flow during Object Decomposition

In the current prototype, the object schemas are stored in files in the user area; this provides locality of references to the objects created by the user. The object instances are stored in the main memory in some form of structure (such as linked lists or hash tables) that would be convenient for the specific processing modules. A graphic user interface is developed using Xwindow/motif and the prototype system is running on a VAX 3100 system.

6 An Object Management System for Integrated Engineering Design Environment

The prototype object management system can be extended to support integrated computer aided engineering environment. In this section, we first discuss the flexibility of the system for supporting design abstraction and process. We then describe two specific current efforts: One attempt is to integrate the view-object paradigm with CADD graphic software. The other effort is to extend the view-object paradigm for the support of cooperative engineering design.

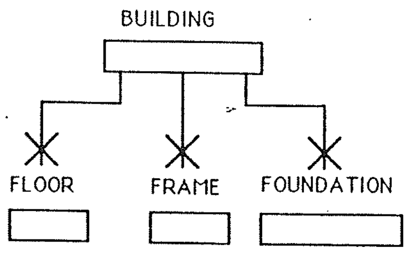
6.1 View-Objects and Design Abstractions

As the design process progresses from conceptualization to design, the way to represent the design is constantly changing. For an integrated design system to be effective, the database system must be able to accomodate the "growth" of the design. As mentioned earlier, the data model must support a wide variety of design representations, sharing the same information in the engineering model. In addition, the data model must allow dynamic changes of the object schema, reflecting the evolutionary process of design, and must be able to minimize database reorganization. The view-object facility described in the previous section allows the engineer to select the object information pertinent to a design task and to ignore the irrelevant details. Furthermore, the separation of the object schema and the database schema can facilitate schema evolution during the design process.

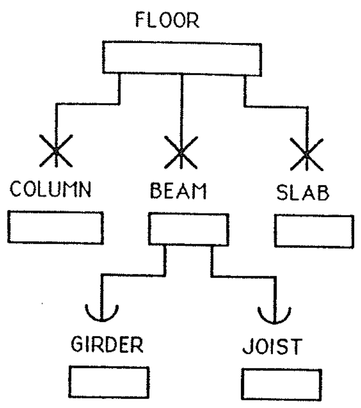
An abstraction view of a building and its components is not unique. An engineer abstracts a specific view of design to focus on a particular task. As an example, an hierarchical decomposition of a building structure is shown in Figure 13(a). As shown in Figure 13(b), for design purposes, a floor composed of beams (including girders and joists), slabs, columns can be conveniently treated as objects. For analysis purposes, a building frame composed of girders and columns is defined as shown in Figure 13(c). We see here two multiple design views sharing the same information. For the respective views, the attributes of a shared entity are not the same. For the description of a floor plan, only the location and orientation of the columns are important. For frame analysis, however, the location, the dimension and the properties of the columns are needed.

By allowing the user to select any number of secondary relations for inclusion, an object can be specified to any level of details that is desired. For example, in the earlier analysis stage, a floor may be treated as the basic component entity but may be expanded to include other subcomponents such as beams and slabs at a later stage of the design; an object schema can be changed dynamically as the design evolves. Hence, the complexity of a design can be managed by suppressing the irrelevant details as necessary. Furthermore, the description of an entity can be refined as needed.

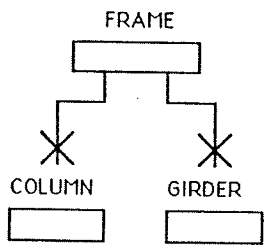
In Figure 13(c), the entity "FRAME" is composed of subentities "GIRDER" and



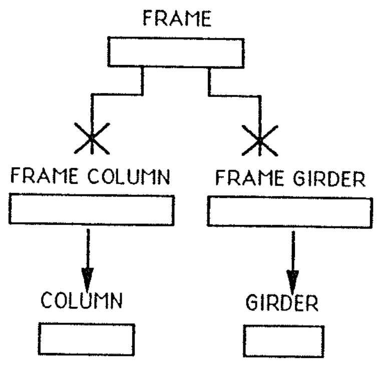
(a) A Hierarchical Decomposition of a Building



(b) Decomposition of a Floor Entity for Design Purpose



(c) Decomposition of a Frame Entity for Analysis Purpose



(d) Alternative Definition of a Frame Entity

Figure 13: Multiple Views and Representations of Design Objects

"COLUMN". However, removing a "FRAME" instance does not necessarily imply that all its constituent components be deleted as well. As shown in Figure 13(d), an alternative may be to augment the "FRAME" object with the auxiliary relations "FRAME GIRDER" and "FRAME COLUMN", which reference relations "GRIDER" and "COLUMN", respectively, and are owned by relation "FRAME". This new structure provides an associative relationship such that removing a "FRAME" instance does not affect the base relations "COLUMN" and "GIRDER"; however, a column cannot be deleted as long as a structural frame containing that column exists. A similar view can be created for the abstract entity "FLOOR". It should be emphasized that modifications made to an individual object template does not necessarily lead to modification of a higher level object. Conversely, modifications of the object network do not affect the definition of the base relations.

6.2 Integration with Object-based Graphic System

One limitation of the current development is that all persistent data must be stored in a relational data base. Relational databases are limited to character and numeric data types and operations and related hashing and B-tree indexing. In general, design/built environment depends heavily on graphical data. Data types associated with graphic applications include points, lines and polygons; operations include intersection, union, transformation etc.; and access is often by indexing. However, a geometric interpreter is needed to convey objects (such as walls, windows) understandable to the designer from points and lines on a drawing [32]. Alternatively, symbols can be assigned as collections of graphic primitives (points and lines) for manipulation as a whole. The Graphic Design System (GDS), for example, extends this concept to named graphic objects [30]. Object and object class level operations are provided.

Relations or tables are useful for tabular data, but not graphic data. One possibility is to extend view object paradigm to include the concept of graphic view objects [26]. The tabular representation of the graphic image of an object would generally require a number of normalized relations. It has already been demonstrated that object based graphic systemcs can be combined with relational database management systems. Using a SQL*CAD approach, data relating to geometry, location and topology are stored as graphic objects. An RDBMS is used to store additional non-graphic data. Nonpersistent relations can be generated from the graphic objects for subsequent relational operations with the RDBMS data. Reversing this strategy and combining it with the view object paradigm results in the concept of graphic view objects. Figure 14 shows an example of using the structural data model for linking the object wall to reference to graphic objects available in a CAD system.

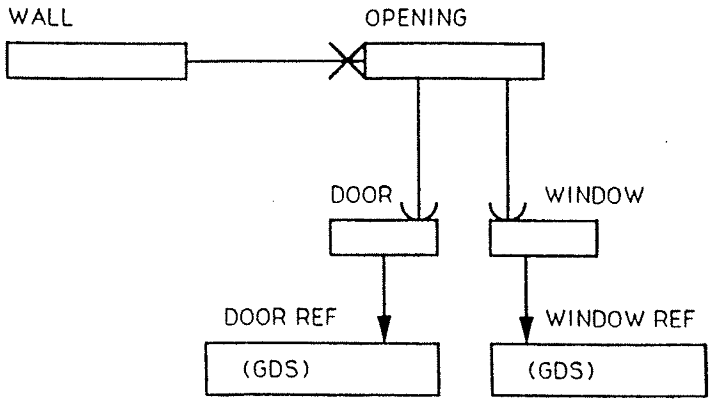


Figure 14: Linking a View Object to Graphic Library

With graphic view objects, geometric, locational, and topological data are stored persistently in an object based graphics systems where the logical view of the data is its graphic representation. Other data is held in a relational database. Graphic view objects are created from a combination of graphic objects and relational data.

6.3 Change Management with View Object Paradigm

The usefulness of a set of computer-based design tools is limited by the degree to which they can work together and share information. In exploring the possibility of an integrated building database to improve design, engineering, construction and maintenance processes, it is often found that data are usually lost during a building's life cycle – information that often must be recreated at considerable cost during the operational phases of a building's life. Furthermore, although there is a lot of information that could be shared, the different tools that each discipline uses do not "talk" to one another.

Design is often a cooperative effort among several engineers. Even in a spirit of cooperation, however, it is quite common for one engineer to modify data that are used by another engineer. It is necessary that the changes are propagated in a controlled manner, and that appropriate parties are informed of conflicting updates to the shared data. Some attempts have been made to build an integrated CAD design system (i.e., a set of design tools which are sensitive to the changes to the design data made by other tools in the set), hardwiring the paths for change propagation. Unfortunately, it is not feasible for each tool to understand the semantics of changes made by all other tools.

One approach is to formalize and develop change managers and change coordinators in design database. The main idea is that each application tool has specific "interests" which are collections of events for which the tool would like to be notified in case the events occur. In the view-object paradigm, some "interests" are expressed through those data shared by different view objects. In the object managment system, tools would express the transactions to retrieve or update data from the database. These include the processes of object instantiation and object decomposition. The interests would be the changes on the shared data in the database. For example a tool may want to be notified if an object instance "b" of class "B" has been changed, or if "any" instance has been changed in class "C". In the first case, we will have many messages passing back and forth, but changes on different instances of the same object class will not be affected. In the latter case, the number of messages would be considerably reduced, but a tool might receive notification for changes that do not really concern it. This subject is of great importance for engineering databases that must support a concurrent engineering environment. Because transactions are dynamic, the number and configuration of change coordinators vary. Additionally the change manager needs to maintain one registry of tools which are executing and includes a module which watches for potential conflicts (which involves changes occurring across transactions). A preliminary effort towards defining a change manager that is suitable for cooperative design environment has been initiated [18].

7 Summary

For an integrated design system to be effective, the database system must be able to accomodate the "growth" of the design as well as to support a wide variety of design representations, sharing the same information in the engineering model. The view-object facility described in this report allows the engineer to select the object information pertinent to a design task and to ignore the irrelevant details. An abstraction view of a building and its components is not unique. An engineer abstracts a specific view of the design to focus on a particular task. Multiple design views can share the same information, but for each respective view, the attributes of the shared entity need not be the same. The view object facility allows multiple views to be defined and to share the same information.

By allowing the user to select any number of secondary relations for inclusion, an object can be specified to any level of detail desired. An object schema can also be changed dynamically a In this report, we have described an architecture for managing design objects in a relational framework. The key benefits of the proposed view-object interface to a relational system include information sharing, data integrity and persistent storage of data. In addition, any new attributes and/or relations added to the underlying database do not affect the exisiting object definitions. Conversely, changes in the definition of any objects do not affect the schema of the underlying database. In other words, the architecture is sufficiently flexible to allow growth as design evolves. Continuing effort is proposed to integrate this prototype object-based system to a CAD system and to extend the view-object concept to support cooperative engineering design environment.

References

[1] Ackroyd, M.H.; Fenves, S.J.; McGuire W. (1988) Computerized LRFD Specification. National Steel Construction Conference.

[2] Bancilhon, F.; et al. "The Design and Implementation of O_2 – An Object-Oriented Database System", In: Advances in Object-Oriented Database Systems. Springer-Verlag, 1988.

[3] Barsalou, T. View Objects for Relational Databases. PhD Thesis, Medical Information Sciences Program, CSD report STAN-CS-90-1310, Stanford University, 1990.

[4] Barsalou, T.; Siambela, N.; Keller, A.M.; Wiederhold, G. Updating Relational Databases through Object-Based Views. Proceedings of 1991 ACM SIGMOD International Conference on Management of Data, Denver, CO, pages 248-257; 1991.

[5] Bjork, B.-C. (1989) Basic Structure of a Proposed Building Product Model. Computer Aided Design 21:71-78.

[6] Blaha, M.R.; Premerlani, W.J.; Rumbaugh, J.E. "Relational Database Design using an Object-Oriented Methodology", Communications of the ACM, Vol. 31, No. 4, 1988, pp. 414-427.

[7] Brodie, M.L. On the Development of Data Models. In: On Conceptual Modeling (Eds. M.L. Brodie, J. Mylopoulos and J.W. Schmidt). Springer-Verlag, pages 19-48; 1982.

[8] Brodie, M.L. "Association: A Database Abstraction for Semantic Modelling", Entity-Relationship Approach to Information Modeling and Analysis, Edited by P.P. Chen, Elsevier Science Publishers, 1983 pages 577-602.

[9] Bryant, D.A.; Dains, R.B. (1977) Models of Buildings in Computers: Three Useful Abstractions. IF, 8(2):9-14.

[10] Darwiche, A.; Levitt, R.E.; Hayes-Roth, B. (1988) OARPLAN: Generating Project Plans by Reasoning about Objects, Actions and Resources. The Journal of Artificial Intelligence in Engineering Design, Analysis and Manufacturing, 2(3):169-181.

[11] Dittrich, K.R. Object-Oriented Database Systems: The Notion and the Issues. Proceedings of the International Workshop on Object-Oriented Database Systems, Pacific Grove, CA; 1986.

[12] Eastman, C.M. (1978) The Representation of Design Problems and Maintenance of Their Structure. IFIPS Working Conference on Application of AI and PR to CAD, IFIP, pages 1-23.

[13] Eastman, C.M. (1988) Automatic Composition in Design. Design Theory '88 (Eds. Newsome, S.L., W.R. Spillers and S. Finger). 1988 NSF Grantee Workshop on Design Theory and Methodology.

[14] Fishman, D.; et al. "Iris: An Object-Oriented Database Management System", ACM TOIS, Vol. 5, No. 1, 1987, pp. 48-69.

[15] Furtado, A.L.; Casanova, M.A. (1985) Updating Relational Views In: Query Processing in Database Systems (Eds. W. Kim, D.S. Reiner and D.S. Batory). Springer-Verlag, New York, NY.

[16] Garrett Jr., J.H.; Breslin, J.; Basten, J. (1988) An Object-Oriented Model for Building Design and Construction. Extended Abstract, Department of Civil Engineering, University of Illinois, Urbana, Illinois.

[17] Garrett Jr., J.H.; Basten, J.; Breslin, J.; Andersen, T. (1989) An Object-Oriented Model for Building Design and Construction. Computer Utilization in Structural Engineering, Structures Congress, ASCE, pages 332-341.

[18] Hall, K. A Framework for Change Management in a Design Database. Ph.D. Thesis, Computer Science Department, Stanford Unversity; 1991.

[19] Keller, A.M. "The Role of Semantics in Translating View Updates", IEEE Computer, Vol. 19, No. 1, 1986, pp. 63-73.

[20] Lavakare, A.; Howard, H.C. (1989) Structural Steel Framing Data Model. Technical Report 12, Center for Integrated Facility Engineering, Stanford University.

[21] Law, K.H.; Jouaneh, M.K. (1986) Data Modeling for Building Design. Fourth Conference on Computing in Civil Engineering, ASCE, pages 21-36.

[22] Law, K.H.; Jouaneh, M.K.; Spooner, D.L. (1987) Abstraction Database Concept for Engineering Modeling. Engineering with Computers, 2:79-94.

[23] Law K.H.; Barsalou, T. "Applying a Semantic Structural Model for Engineering Design", ASME International Conference on Computers in Engineering, Anaheim, CA, 1989, pp. 61-66.

[24] Law, K.H.; Barsalou, T.; Wiederhold, G. "Management of Complex Structural Engineering Objects in a Relational Framework", Engineering with Computers, Vol. 6, 1990, pp. 81-92.

[25] Law, K.H.; Wiederhold G.; Barsalou T.; Siambela N.; Sujansky W.; Zingmond D.; Singh H. An Architecture for Managing Design Objects in a Sharable Relational Framework. International Journal of Systems Automation: Research and Applications, 1:47-65; 1991.

[26] Law, K.H.; Scarponcini, P. A View Object Approach for Managing Design/Built Information. The 7th Conference on Computing in Civil Engineering, ASCE. pages 192-201; 1991.

[27] Learmont, T.; Cattell, R. "An Object-Oriented Interface to a Relational Database", Proceedings of the International Workshop on Object Oriented Database Systems, 1987.

[28] Lee, B.S.; Wiederhold, G. "Outer Joins and Filters for Instantiating Objects from Relational Databases through Views", Technical Report No. 30, Center for Integrated Facility Engineering, Stanford University, 1990

[29] Maier, D.; Stein, J. "Development of an Object-Oriented DBMS", Proceedings of the International Conference on Object-Oriented Programming Systems, Languages, and Applications, 1986.

[30] GDS Reference Manual, McDonnell Douglas Built Environment Technologies, McDonnell Douglas; 1990.

[31] Paepcke, A. "PCLOS: A Flexible Implementation of CLOS Persistence", Proceedings of the European Conference on Object-Oriented Programming, Oslo, Norway, 1988.

[32] Pohl, J.; Myers, L.; Chapman, A.; Cotton, J. ICADS: Working Model Version 1. Technical Report No. CADRU-03-89, California Polytechnic State University; 1989.

[33] Powell, G.H.; Bhateja, R. (1988) Database Design for Computer Integrated Structural Engineering. Engineering with Computers, 4(3):135-144.

[34] Smith, J.M.; Smith, D.C.P. "Database Abstraction: Aggregation and Generalization", ACM TODS, Vol. 2, 1977, pp.105-133.

[35] Spillers, W.R.; Newsome, S. (1988) Design Theory: A Model for Conceptual Design. Design Theory '88 (Eds. Newsome, S.L., W.R. Spillers and S. Finger) 1988 NSF Grantee Workshop on Design Theory and Methodology.

[36] STEELCAD (1989), Cadex Ltd.

[37] Stonebraker, M.; Rowe, L. "The Design of POSTGRES", Proceedings of ACM SIGMOD Conference, 1986, pp. 340-354.

[38] Vernadat, F.B. (1984) A Commented and Indexed Bibliography on Data Structuring and Data Management in CAD/CAM : 1970 to Mid-1983. National Research Council of Canada, Technical Report 23373.

[39] Wiederhold, G.; ElMasri, R. "The Structural Model for Database Design", Entity-relationship Approach to System Analysis and Design. North-Holland, 1980, pp. 237-257.

[40] Wiederhold, G. "Views, Objects, and Databases", IEEE Computer, Vol. 19, No. 12, 1986, pp.37-44.

[41] Wiederhold, G. Connections. In: Managing Objects in a Relational Framework. Technical Report STAN-CS-89-1245, Department of Computer Science, Stanford University; 1989.