Published in Automation in Construction **8**(1):49-72, November 1998

Intelligent Representation for Computer-Aided Building Design

Lachmi Khemlani
Department of Architecture, University of California at Berkeley
lachmi@ced.berkeley.edu

Anne Timerman
Department of Architecture, University of California at Berkeley
atimerma@ced.berkeley.edu

Beatrice Benne
Department of Architecture, University of California at Berkeley
bbenne@ced.berkeley.edu

Yehuda E. Kalay
Department of Architecture, University of California at Berkeley
kalay@ced.berkeley.edu

ABSTRACT

At the core of any computational system that can support design development, analysis, and evaluation is an "intelligent" building representation which should be able to represent all the different components that make up a building, along with the manner in which they come together. In other words, the representation must be informationally complete and semantically rich. The paper discusses these two criteria and briefly reviews other research efforts aimed at developing building representations for computer-aided design that attempt to meet them. Our solution to this problem is then presented. It is aimed primarily at the schematic design phase, the rationale for which is also stated. Taking the view that buildings are unique assemblies of discrete, mostly standardized components, our representation is clearly divided into two components: the Object Database (ODB) which stores detailed information about various building elements, and the Project Database (PDB) which holds information about how these elements are assembled to make up a particular building. An ODB may be shared by many building projects, while the PDB must necessarily be unique to each. The data schemas of both the PDB and the ODB are described in detail and their computational implementation, to the extent that it has been completed, is illustrated.

1. INTRODUCTION

The use of computers to enhance the quality of the processes used for designing buildings, as well as the quality of the built environment itself, has been the subject matter of considerable research for more than three decades. Early efforts concentrated primarily on evaluating and optimizing various building functions, such as their layout and energy consumption (see, for example, Armour & Buffa 1968, Shaviv & Shaviv 1977). These efforts, while effective in some narrow domains, failed to bring about the sought improvements for the building industry in general. Since the late 1970s, many researchers have recognized that the fundamental obstacle to the desired improvements is the lack of an adequate computational representation of buildings.

Traditional representation methods used by architects and engineers for hundreds of years, such as scale drawings, renderings, and three dimensional scale models, contain only a small part of the information needed to interpret and assess the quality of the design. Drawings, for example, represent only the geometry of the building, in highly symbolic form. Even when this information is accompanied by textual labels and specifications (materials, products, construction techniques, etc.), it still relies heavily on the intelligence and professional training of the (human) observer, who must augment the information with his/her own knowledge of the field.

The almost direct importation of these traditional representation methods to computer aided design have propagated the same informational deficiencies. Augmentation of the data with computer-readable "knowledge" is difficult, given that inference of meaning is one of the most difficult abilities to impart to computers (witness the difficulties encountered in the field of Artificial Intelligence). Ad-hoc efforts to add intelligence to specific applications were generally unsuccessful in enhancing the utility of computers in

building design, except in some very well defined, narrow domains. What is needed is an overall more "intelligent" representation, which will embody some of the knowledge added to the interpretation of drawings by the human observers. The history of research in architectural CAD, from the late 1970s on, has been marked by the quest for this "holy grail" of intelligent building representation.

2. WHAT IS AN INTELLIGENT BUILDING REPRESENTATION?

While different researchers proffer different definitions of what an intelligent building representation should be (see, for example, Carrara & Kalay 1994), they generally agree that a computational representation that can support the qualitative aspects of building design needs to have the following two qualifications:

1. It must include information that describes all aspects of the building.
2. This information must be semantically meaningful.

2.1. Informational completeness

The first qualification, whose importance is self-evident, must further meet the following requirements:

1. *Well formedness*: The data must be self-consistent. For example, a window representation in a plan drawing must have the same location and dimensions as it has in an elevation or a section.
2. *Generality*: The data must include the necessary information to assess the building from many different points of view. For example, it must be able to support evaluation that pertains to spaces as well as the enclosure, and to structural qualities as well as architectural ones.

2.2. Semantics

The second qualification is more difficult to define and to accomplish. How, and how much of our own "understanding" can we impart to computers? A similar set of questions have frustrated AI researchers in architecture as well as in other fields for the past 40 years (see, for example, Coyne et al 1989). Yet, when considered from a *representational* point of view, these questions take on a somewhat less formidable character: it is not necessary to make the computer be able to "understand" the meaning of the data, only to be able to *identify* it, for the benefit of other programs or human experts that will use the data intelligently. For example, it is not necessary to make the computer understand the essence of "door-ness," as a portal that enables social processes like "entry" and "exit." From a representational point of view, the computer only needs to be able to identify a door when one exists in a given location. This ability will support qualitative assessments such as access, egress, and privacy, which will be performed by expert systems or other types of programs, to be developed separately from the building representation.

3. OTHER EFFORTS

Considerable research effort throughout the CAAD community has been devoted to developing a representation that can provide adequate support for computer-aided building design. One of the foremost and mostly widely known of these efforts is STEP, a product model concerned with defining a standard for the representation and exchange of data (Bjork and Wix 1991). STEP however, does not focus only on building information but product information in general. Therefore, it is too general-purpose and too limited, as far as semantic content is concerned, to provide an efficient and workable solution to the problem of building representation.

The COMBINE project, working with the STEP standard, deals with a neutral file exchange of data among various application programs—primarily energy and HVAC (Amor et al 1995). It does have, at its core, an integrated data model for the building description, which gets translated to the applications. As this data model was developed by synthesizing the individual data schemas of the various design tools it wanted to integrate, each time a new tool has to be added to the system, the data model has to be re-worked. It therefore, does not qualify as an informationally complete building representation. Similarly, systems like OXSYS (Richens 1977)

and IBDE (Fenves et al 1994) that deal with hospital buildings and high-rise office buildings respectively are also too specialized to provide the solution to a generic building representation.

One effort directed specifically towards enhancing the semantic content of building representations was KAAD (Carrara et al 1994). It employed an object oriented approach, where entities called SPACE UNITS, BUILDING UNITS and FUNCTIONAL ELEMENTS possessed parameters that defined specific attributes of the represented objects (Figure 1). In addition, the objects included *methods*, which are functions that calculate values assigned to an object, based on values already assigned to other objects. Objects also included several kinds of semantic links, such as inheritance links (AKO, ISA) and association links (IMS). Using these attributes, methods, and links, KAAD was able to help in assessing the quality of nursing units in Italian hospitals in terms of circulation, space allocation, fire egress, energy consumption, code compliance, and other performances. It was, however, a closed system, useful only as a concept demonstration rather than an actual tool.

```
(dg3 (ako value su))
        (description (value "space unit for patient's nursing room" ))
        (ims     (value hfur3 ite ))
        (sup     (min 22)
                 (unit mq)
                 (max 28)
                 (unit mq)
                 (description "minimum and maximum net area"))
        (wtemp   (range 19 21)
                 (unit °C)
                 (description "interior winter temperature"))
        (stemp   (range 25 27 )
                 (unit °C)
                 (description "interior summer temperature"))
        (vent    (value 2 )
                 (unit vol/h )
                 (description "number of air exchanges"))
        (vela    (value 0.2 )
                 (unit m/sec )
                 (description "air velocity"))
        (sound   (max 42)
                 (unit dbA )
                 (description "maximum noise level"))
        ...................
```

Figure 1 - A partial definition of a typical SPACE UNIT object in the KAAD system (Carrara et al, 1994).

Probably the effort that comes closest to our definition of an intelligent building representation is the EDM (Eastman and Siabiris 1995). It was developed to address issues in the design of buildings with wide ranges of design abstractions, construction technologies, and variations in building use, and has representations for each of these in the form of the constructs BOUNDED_SPACE, CONSTRUCTED_FORM, and ACTIVITY, respectively. It is meant to be open-ended so that additional descriptions can be added. EDM thus aims to be general-purpose as well as complete. EDM, however, separates the geometric representation of the building from the integrity constraints that endow it with semantic validity. The semantics are computed as and when needed by each application program individually, based on its own pre-conditions and its domain of interest. In contrast, we take the view that most, if not all, of the semantic integrity should be built into the topological representation of the building itself, so that no computation is needed to establish the semantic validity of the model once it has been built.

4. OUR APPROACH

Having witnessed the successes and failures of these efforts, we set out to develop an informationally complete and semantically rich building representation that will be capable of supporting the interpretation and assessment of buildings. This representation is based on the following three principles:

1. It addresses *only one phase* in the overall design process, specifically the schematic design phase (not just for architecture, but for all the building-related disciplines).
2. It recognizes that building design is, in many ways, the process of *assembling universal components into a unique composition*.

3. It makes a conscious effort to integrate the representation of *spaces and enclosures* in one unified representation.

4.1. Schematic design

Schematic design is the phase where the gross features of the building are defined, including its overall shape, size, structure, adjacencies, circulation, and the materials it is made of. While each feature will, no doubt, undergo refinement and elaboration in subsequent design phases, schematic design is the phase when most of the cardinal properties of the building are determined, or at least are first being considered.

It is, of course, difficult to define precisely what constitutes schematic design. Different architects will likely consider different parts of their work as "schematic" vs. "design development" or even "detailed design." It is even possible that such designations change from one project to another. While we do not attempt to define "schematic design," we are cognizant of other efforts that tried to do so. For example, a systematic effort to characterize the elements of different phases in the design of energy-efficient buildings was made by Shaviv & Kalay (1991). In that study, various energy-related design parameters were identified by the design phase in which they are considered first.

We have chosen to limit our efforts to support this particular phase of the design process because we consider it the phase when designers are most in need of the assistance that could be offered by computers—in terms of assessing whether the areas that were allocated are appropriate, the adjacencies among them proper, whether the proposed structure will stand up, whether the budget has not been exceeded, whether the form that is being considered can support all the functional needs, and so on.

Our choice of design phase was also influenced by the simplification our decision brought to some of the representational aspects we need to deal with. The major simplification is due to the geometry: at the schematic phase, only the gross geometric features of the building need to be specified. As such, our data structure is limited to representing only prismatic forms (ones where there is no change in geometry along the Z axis) which are bounded by planes. Furthermore, we ignore small geometric changes, for instance, the ones along walls that would be caused by columns and window sills. This allows us to also simplify the tools that will be used to create and edit the representation at the schematic phase. We assume that the design developed at this phase will eventually be moved on to some commercial CAD system where its form will be further elaborated, even modified within some semblance of the schematic form.

Another simplification that accrues from limiting the representation to the schematic design phase pertains to the semantics that need to be embedded in the database to facilitate meaningful analysis. These semantics need not account for all the details of the building. In fact, most CAAD research efforts in the area of evaluation and prediction emphasize the importance of analyzing the building in the preliminary design phase when all the major design decisions are made. Therefore, several evaluation tools are specifically targeted at the schematic phase for providing useful feedback to the respective building professionals on various design aspects such as spatial allocation, cost, structures, energy consumption, fire safety, and so on without requiring detailed design input. We strongly believe in this approach to evaluation and our representation is, therefore, aimed at addressing the needs of such evaluation tools.

It must be emphasized that our representation does not attempt to deal with the preliminary "fuzzy" design conceptualization phase where the architect is sketching various shapes and forms, which might bear no relation to actual building elements. Instead, we are dealing with that phase in which the design has made enough progress to be modeled into building plans, since we believe that only at this stage can the performance analysis of various design aspects really begin. Further discussion on this self-imposed limitation appears after the representation has been described, in the Conclusion section of the paper.

4.2. Two kinds of information

Buildings, like many other artifacts, are unique *assemblies* of discrete, mostly standardized *components*. In developing a computational building representation, therefore, it would be prudent to separate the information into two categories:

1. Information that pertains to the *components.*
2. Information that pertains to the *assembly*.

We call the information pertaining to the components *Object Data*, because it typically represents discrete objects (doors, walls, windows, etc.). We call the information pertaining to the assembly *Project Data*, because it is typically project-specific. A certain project may use several object databases, and an object database can serve many projects (Figure 2).

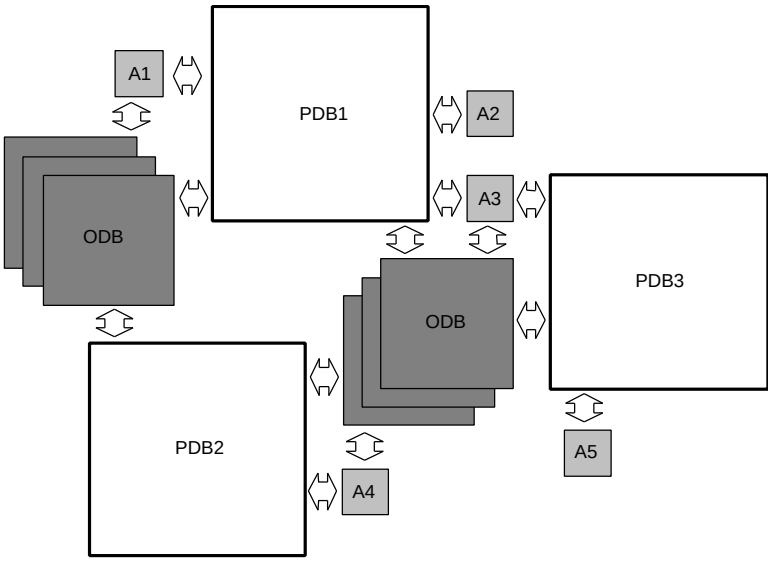

Figure 2 - The relationship between the PDB and the ODBs, and the applications that interface with them for evaluating various aspects of building design.

The first kind of information is typically the province of the manufacturers or the specialists who produce building components. The second kind of information is the province of the building design professionals. In many ways, such a separation already exists today in the paper-based practice of building design, where general product catalogs like SWEETS and THOMAS' are routinely used along with standardized office details to complete the information pertaining to a specific design.

Other research efforts in the field do not make the separation between assembly and object and have both kinds of information integrated within a single representation (see, for example, the component-based building representation effort by Harfmann & Chen 1993). In contrast, we have found that by separating the components (objects) from the assemblies in which they are used, we can simplify both representations (of the assemblies and the components), and make the databases easier to implement and to maintain. Yet, the ability of the separate representations to encode all the information necessary for representing a building is not affected, because the two come together through the applications that evaluate the proposed design (Figure 2), similar to the manner it is currently done in the construction industry.

5. OBJECT DATABASE (ODBs)

ODBs are domain-specific knowledge bases of project-independent information that is needed to generate and to evaluate objects of the kind they represent. These can be spatial objects (rooms, external spaces) or physical building components (e.g. walls, slabs, doors, windows, furniture and equipment, etc.). For example, rather than define a hung door every place in every project that uses one, it is described only once in the appropriate ODB, then instantiated whenever needed. Only attributes and requirements that are specific to the instantiating project need to be added to the generic definition, such as the location of the door and its association with other objects in the project (e.g., wall, spaces, etc.).

The practice of augmenting drawings with paper-based product catalogs, and more recently with CD-ROM-based electronic catalogs, is not new. Currently, this useful practice is being transformed again, using the World Wide Web in lieu of CD-ROMS (see, for example, *www.constructionsite.com*, *www.sweets.com*, and *www.pacificbuilding.com/product/product.html*). The Web, which enables sharing media-rich information across platforms without regard to the location of the information, provides much more up-to-date product information while saving product suppliers the cost of producing many copies (and versions) of the CD-ROM. Furthermore, when the information provided by the product manufacturers is made compatible with the CAD systems and the applications that use them, product data carried by the Web could be integrated directly with the project data. This integration would allow for dynamic updates (e.g., for costing and scheduling), early detection of incompatibilities among products, substitution of similar but less expensive products wherever possible, and generally linking the project database directly to the manufacturers of the components that are needed to realize it.

However, before such benefits can accrue, two functional and several organizational problems need to be solved. The functional problems are:

1. *Usability*: To benefit from the full potential of the electronic implementation of an object database, the content of the objects in that database must first be made semantically meaningful. Then they need to be linked to the project database, so that the information they contain is accessible to the applications that combine the two separate databases into one whole representation. These applications will have to be able to parse not only the building data itself (the PDB), but also the object data (ODB), so they could take off quantities from the PDB and correlate them with the relevant objects in the ODBs.

2. *Searchability*: There is a vast number of building-related components, and an even larger number of manufacturers who produce them. Each manufacturer uses different terms to describe the attributes of their products, which makes it difficult to identify the sought components and to compare them to each other. Some kind of a standardized organization of all this data is needed.

The organizational problems associated with this idea pertain to the legal responsibilities associated with distributed information systems, their maintenance and update, and of course, the reliability of the information in the ODBs. Although we have some thoughts about how to solve these organizational issues, our research addresses only the functional issues.

5.1. Content and structure of the ODBs

Each object in an ODB must contain all the information needed to define and to evaluate objects of the kind it represents. This information includes:

1. *Attributes*, such as the object's name, its form (shape), the materials it is made of, and other properties (e.g., manufacturer, cost, etc.).
2. *Classification* relationships, which define an inheritance path for properties this object shares with other, more generalized objects of the same type (e.g., that a hung-door is a kind of door, which is a kind of opening).
3. Information that describes the *function* of the object (e.g., that a door facilitates entrance and egress to spaces).
4. *Integrity constraints*, which are logical propositions associated with attribute values and define generic expectations from the object (e.g., that the door must lie within a wall).
5. *Cases*, in multimedia form, which provide anecdotal information about the object.

Unlike other product databases (e.g., EDM), our ODB includes only two types of relations:

1. *Extra-object* relationships of two kinds:
    a. The relationship between sub-classes and their super-class in the abstraction hierarchy. We call this relationship AKO (A Kind Of).

b. The relationship between an instance and its defining class. We call this relationship IS-A.

2. *Intra-object* relationships: The relationship between an object and its attributes.

Notice the absence of assembly (Part Of) relationships, which would be useful to define composite objects (a gypsum partition, a brick wall, a window), but would also significantly complicate the representation (witness similar efforts such as COMBINE and EDM). Instead, we embed assembly information in the representation of the object itself, and make extensive use of the classification hierarchy to compensate for the absence of Part-Of links. Thus, a Window object, for example, can be described as a single entity with certain properties and performance characteristics, or as a composition of the materials and products of which it is made. Detailed versions of the window, which include information about specific glass and finishes might be stored as sub-classes of the window. In this manner, the components of the window will be treated as properties of the more specific window object, and the assembly information that is considered integral to the object definition will be embedded in the representation of the object itself.

5.2. Classification System for the ODBs

Objects are stored in hierarchical classification structures, which facilitate compact, non-redundant, and consistent representation—a method that has now been popularized by object-oriented programming languages. Objects with similar properties and functions are grouped into classes. A class is defined at large and then refined into subclasses. The properties of objects located higher in the abstraction hierarchy (superclass) are inherited by the objects located lower in the hierarchy (subclass). Subclasses add their unique properties to the definition of the superclass. Common attributes are stored at the highest level in the tree that is still shareable by all the objects that can benefit from this information. An inheritance mechanism makes higher-level information available to objects that are stored in lower levels of the hierarchy.

An example of a Hung Door object and its inheritance hierarchy is depicted in Figure 3. This object shares many attributes with other doors, such as revolving, sliding, and composite doors, in that it consists of a screen, usually made of wood, metal, glass, or a combination of materials, installed to swing, fold, slide, or roll in order to close an opening to a room or building. A Hung Door adds to these properties its own unique attributes, such as that it has some kind of a hinged assembly.

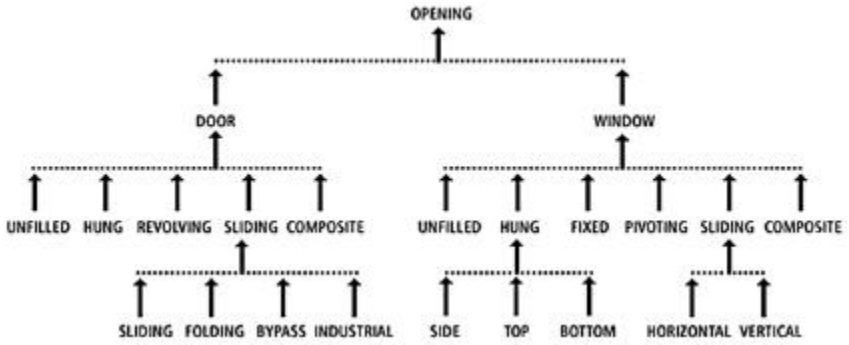

Figure 3 - A Hung door inheriting properties in an "Openings ODB."

The structure of the classification tree directly influences the content of the information, due to the inheritance path it defines. The difficulty of constructing a classification hierarchy for building design resides in the fact that the decomposition of an object is subjective, and depends on individual judgment. For example, a WALL class can be decomposed as LOAD BEARING WALL subclass and PARTITION subclass. Another possibility of decomposition is to say that the LOAD BEARING COMPONENT class is decomposed as WALL, COLUMN, SLAB and BEAM subclasses (Figure 4). In the first case the WALL is the superclass, in the other case LOAD BEARING COMPONENT is the superclass. There is no one correct decomposition—architects, engineers, and construction managers may see the same structure in different ways.

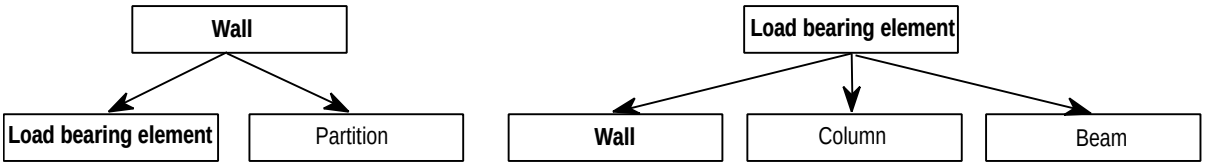

Figure 4 - Different classification hierarchies of the same object.

To help us decide on a classification system, we reviewed three systems of classification that are widely used in architectural offices: Masterformat, AIA CAD layer guidelines, and SfB. The use of a standard classification system introduces uniformity into the database (e.g., it unifies the number of steps from general to specific objects) and provides good ground for extending the database. Moreover, the considerable effort that is needed to develop a classification system can be spared.

Masterformat and SfB organize information hierarchically, from more general (broad in scope) to the specific (narrow in scope). AIA layer guidelines use tags, rather than a hierarchy and, therefore, cannot be used for the ODB.

Masterformat is the construction coordinating system used in the USA and in Canada. It was developed by construction specifiers to provide a standard format for specifications and a cataloging system for product literature. Masterformat allows coordination between the specifications and the manufacturers' literature and makes it easier to reuse the specifications. It supports classification of the physical elements of the building (construction systems, building elements, materials), but does not have provisions to classify spatial properties and requirements.

SfB provides for representing information about spaces in addition to physical building components. It was developed in Sweden, and is broadly used in most European countries, especially in Great Britain. SfB was developed as a "common language" which can be used by all the professionals involved in the design and construction of buildings to assist communication and to make it easier to find information. Contrary to Masterformat, SfB has been used for coding drawings. Furthermore, it has been used at the briefing stages of the design process, to represent data on user requirements for spaces (e.g. kitchens, bathrooms).

5.3. Implementation of the ODBs

We have been implementing the ODBs using two kinds of objects:

1. Generic objects
2. Catalog objects

Generic objects form the upper levels of the inheritance hierarchy. They define all the properties of the objects. The Catalog objects indicate specific instances of the objects that are actually manufactured by specific producers (Figure 5).

5.3.1. *Generic Objects*

Each generic object represents a building component such as a wall, a door, or a window. It describes the nature and basic qualities of that component. Generic objects do not specify default values for any of the attributes. Shaviv and Kalay (1991) showed that default values, or assumptions about most frequently used materials, construction techniques and dimensions, may lead to incorrect results in energy evaluation and other applications. Figure 6 shows an example of what a Generic Object in our implementation looks like.

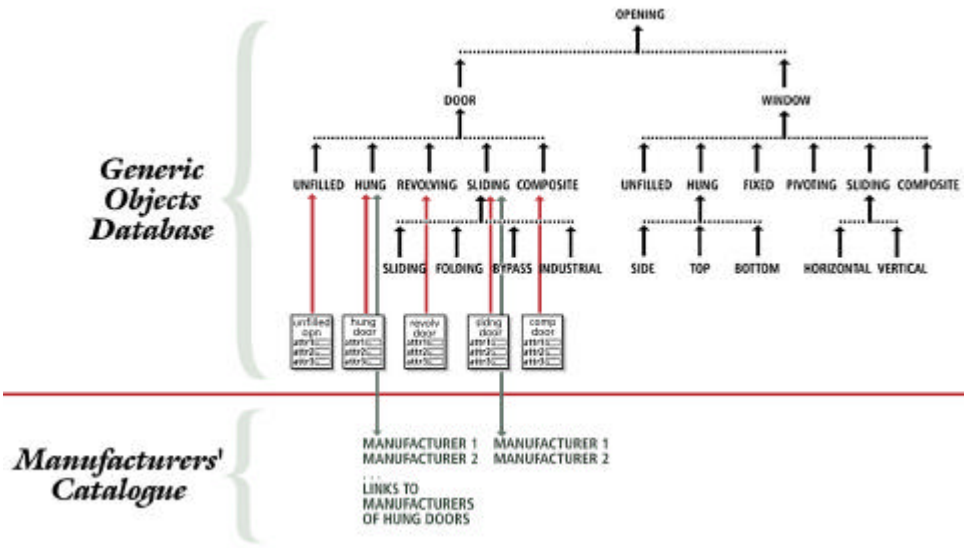

Figure 5 – Object Database: Generic objects and Catalog objects.

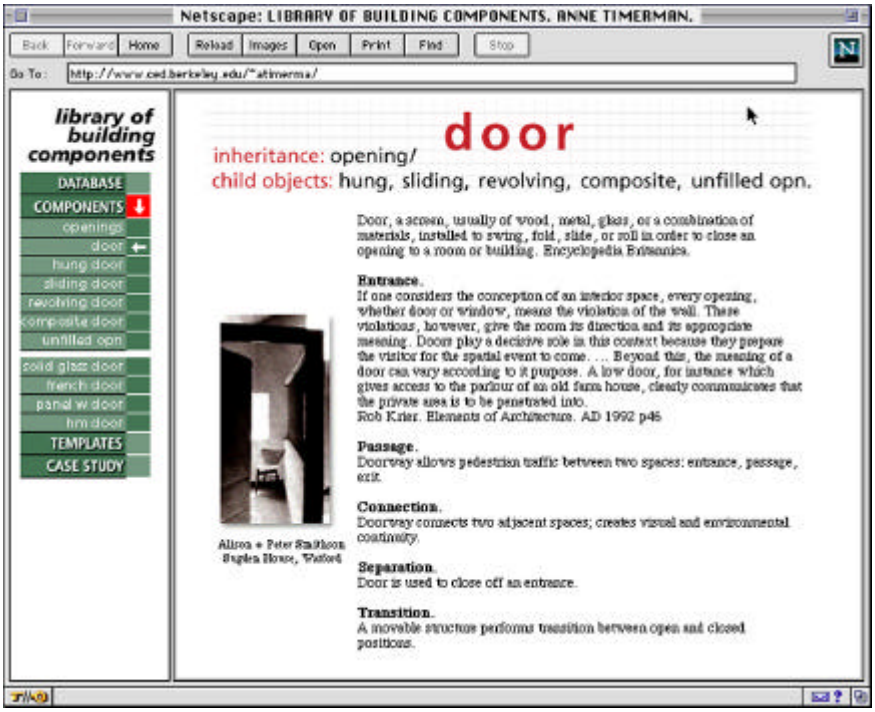

Figure 6 – Door as a Generic object.

Our earlier decision not to include assembly information in the ODBs may lead to exponential proliferation of unique combinations of specific objects. For example, there can be a wide variety of swinging (hung) doors, because construction types, door types, operation types and core materials of swinging doors appear in nearly all combinations. Door types may be superimposed on any door construction. Operation types, such as hinged, pivoting, balanced, pivoting double acting, may be combined with almost all door types. To avoid having to represent explicitly each combination, we have provided a mechanism for creating “ad-hoc” objects—ones that represent a unique combination of attributes. This mechanism is in the form of a template that describes the variable attributes of the object (e.g., construction), along with the allowable values that may be assigned to the

attribute (e.g. open, flush, etc.). Users can "create" a "new" object by choosing a value for each attribute (Figure 7). The database ensures that objects created with the template are well-formed by limiting the value choices to only those that are consistent with the ones already selected.

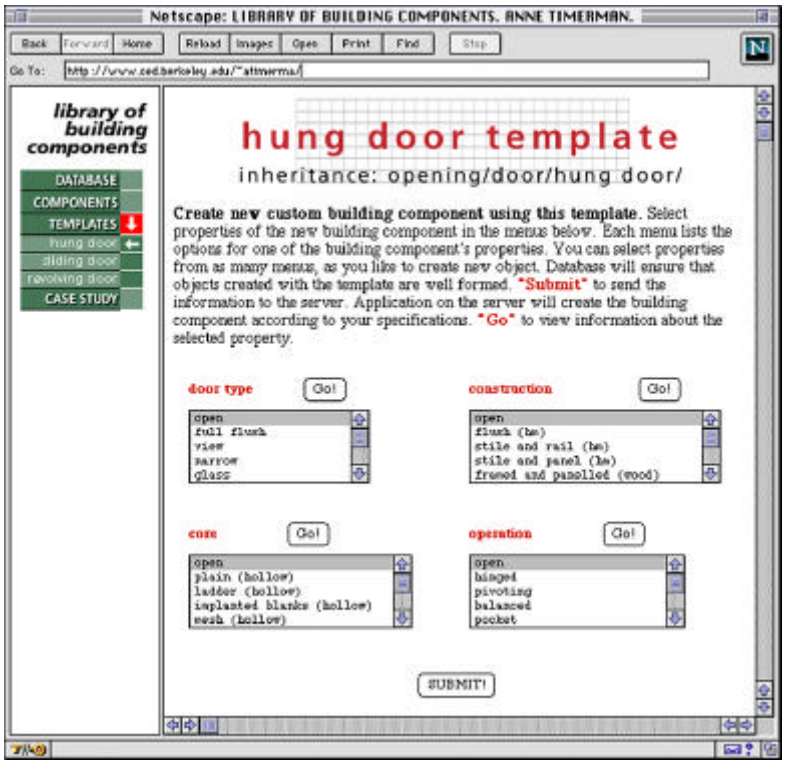

Figure 7 – The new Generic object template that can be used to create customized building components.

5.3.2. *Catalog objects*

Catalog objects link the generic object hierarchy to the manufacturers' own product catalogs. Many manufacturers already have their product descriptions available on the Web (see, for example, *www.steelcraft.com*). Several sites even provide search engines for construction products and components according to keywords as well as Masterformat numbers and titles. Since these sites do not include a taxonomy of building products, nor provide independent information that helps to compare products, their connection to the Generic Objects of the ODB enhances the semantic meaning of the product catalogs.

6. PROJECT DATABASE (PDB)

The PDB is responsible for linking all the components of the building into the appropriate assembly, and storing information about the geometrical location of components with respect to each other. Detailed information about the actual components, when needed, can be obtained from the respective ODBs. Unlike the ODBs, the PDB is unique to each building project.

The PDB carries the semantics of assembly—which parts connect to what parts, and where are they in relation to each to each other. This includes the responsibility for ensuring the three conditions of *disjointedness*, *conjointedness*, and *subjointedness*. The first is when objects must not occupy the same space (at the same time), the second is when they are supposed to do so (e.g., pipes in a wall), and the third is when one object must be included in another (e.g., window). The ODB, in contrast, carries other kinds of semantics, such as

what is a door, a window, a wall, etc. In this way, the semantics of both the PDB and the ODBs complement each other.

In developing a suitable data structure for the PDB, one of the key issues that needs to considered is the relationship between the spaces and the structure of the building.

6.1. The Space-Structure Dilemma

Space and structure in a building are not just two different categories of components, they are of a different "substance" altogether. Spaces are abstract entities—they do not have a physical reality. In contrast, the structure of the building is definitely physical—slabs, beams, columns, walls, windows, and so on actually exist and have a shape, size, material characteristics, etc.

The difficulties in building representation arise from the fact that the physical structure and the abstract space are highly interdependent. Depending upon how the building is viewed, the structure defines the space, but the space, in turn, also determines the structure that bounds it. Building professionals such as the engineer and construction manager are primarily concerned with the physical fabric of the building. In contrast, for professionals such as the architect, the spatial qualities of the building are paramount. There are still others such as the energy consultant for whom both are equally important.

We have identified four main challenges for a building representation as far as space and structure are concerned:

1. Both space and structure must be *explicitly* represented, rather than leaving one to be inferred or derived from the other. Most commercial CAD systems represent only the structure, leaving the spaces it encloses to be inferred implicitly by the designers.

2. Both space and structure must be represented on an *equal footing* rather than giving one precedence over the other, since the representation is aimed at serving the needs of *all* the building design professionals rather than just one group over another

3. The representation should be *non-redundant*. Thus, if a wall is represented in the structural assembly of the building, it should not be re-represented in the spatial assembly.

4. The representation must reflect the *strong interdependency* between space and structure. Spaces are bounded by walls and slabs, which are supported by beams, which in turn are supported by columns. If the size and shape of a space is changed, it must bring about a corresponding change in all of these elements. Similarly, any change in the location of a wall or slab, for instance, must affect the shape of the spaces they enclose. In short, the representation should not allow space and structure to be created or modified independently of each other.

No research effort has emerged so far in the area of building representation that has yielded a satisfactory solution to the space-structure problem just discussed. We will now describe our solution to this problem that has been developed by addressing all of the four issues stated above.

6.2. Version 1 of the PDB

The first version of the PDB (Kalay et al 1996) represented in a non-redundant, interdependent, complementary way, the spaces and the partial structure of the building. Slabs and beams were not represented at all, while columns were represented indirectly rather than explicitly. Nevertheless, it served as a good starting point and enabled us to test it with architecture-specific evaluation tools such as spatial layout, adjacencies, path-finding, etc. (Khemlani & Kalay 1997, Lee 1997).

The dominant feature of the first version of the PDB was that it had one key building element around which the representation of all the other entities was built. This was the SEGMENT, defined as that abstract surface that

encloses a space, but which has some semblance of physical reality by being linked to a wall and having the attribute of surface finish associated with it. The "inspiration" for this idea came from the well known Split-Edge Data Structure, which in turn is a variant of the Winged-Edge Data Structure, both of which have been very successfully implemented in geometric modeling systems (Baumgart 1972, Baer et al 1979, Kalay 1987, Kalay 1989). These data structures use a simple and elegant relational scheme to solve the edge-face problem in polyhedra (which is analogous to the space-structure problem in buildings) in a compact, non-redundant, consistent, and efficient manner.

6.3. Version 2 of the PDB

The current version (Version 2) of the PDB extends the earlier representation to explicitly include the structural entities of the building such as columns, slabs, and beams. While the Split-Edge data structure concept is still embodied in the representation, now as a SURFACE rather than a SEGMENT, it is no longer the key element of the representation. The key is now a revamped VERTEX entity which is purely geometric (as opposed to using it to also represent columns as we did in Version 1), but which is used to physically locate and bring together all the various elements of the building. The simplicity and elegance of the Split-Edge data structure, however, continues to "inspire" the development of the representation, and is very much in evidence in the current version.

With the structure being more comprehensively represented, it necessitated a redefinition of what constitutes space, as opposed to what constitutes structure, and how they are inter-related. We found it convenient to adopt the approach illustrated in Figure 8 for our representation. A space is regarded as a volume bounded by surfaces (not walls). Surfaces can have openings associated with them for doors and windows. Columns, slabs, and beams form another group of interconnected elements. Walls may or may not be integrated with the structure. The space and the structure are related to each other in a number of ways—vertical surfaces are linked to walls, horizontal surfaces are linked to slabs, slabs are linked to beams, walls are linked to beams, and surfaces are also linked to columns.

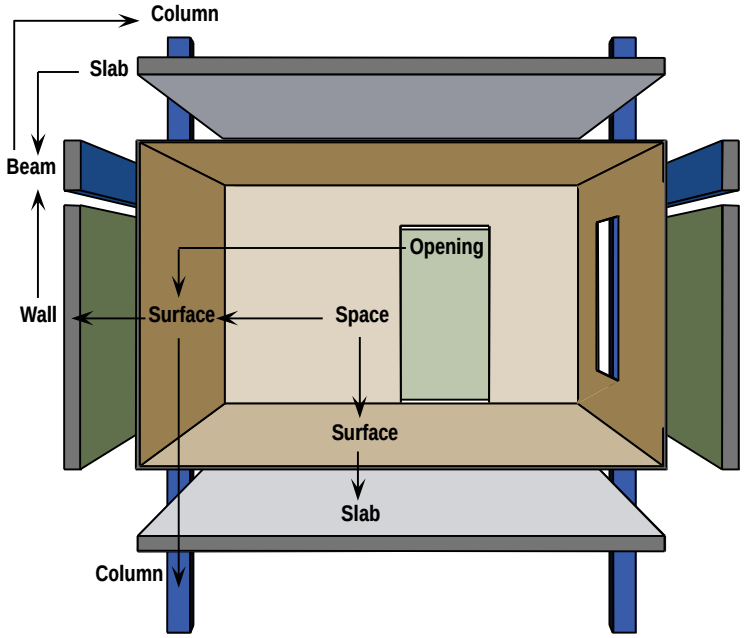

Figure 8 - The approach underlying the PDB representation scheme.

We will now explain the data scheme of the various elements that constitute the PDB, with reference to a simple building as an example, illustrated in Figure 9. Since the database is relational, the data will be represented by means of tables for each element. Where the element is also an object that can be referenced in

an ODB, one field in the table is allocated for storing the tag which will link the element to the corresponding object in the ODB.

For the sake of clarity, un-normalized versions of the relational tables will be presented in this paper. In the actual implementation of the PDB, the tables are normalized so that regular SQL commands such as SELECT, PROJECT, JOIN, and so on, can be used for retrieving information from them.

6.3.1. *Buildings And Levels*

At the top rung of our PDB hierarchy lies the *building project* which can comprise a group of buildings rather than just a single building. We therefore have a BUILDING table that can have multiple entries for buildings as shown in Table 1.

Table 1 - The BUILDING table.

Building ID	First Level	Building ODB Link
BLD1	LVL1	*tag name*
...	...	...

Within a building, the highest category is a *level*. A building can be composed of many levels (Figure 9). For each building in the table however, only the first level is identified. Thereafter, information about subsequent levels of the building can be obtained from the LEVEL entity, which will include a pointer to the next level of that building (see Table 2). If some information about that building type is available in a "Building ODB," there is a place in the BUILDING table for storing that link.

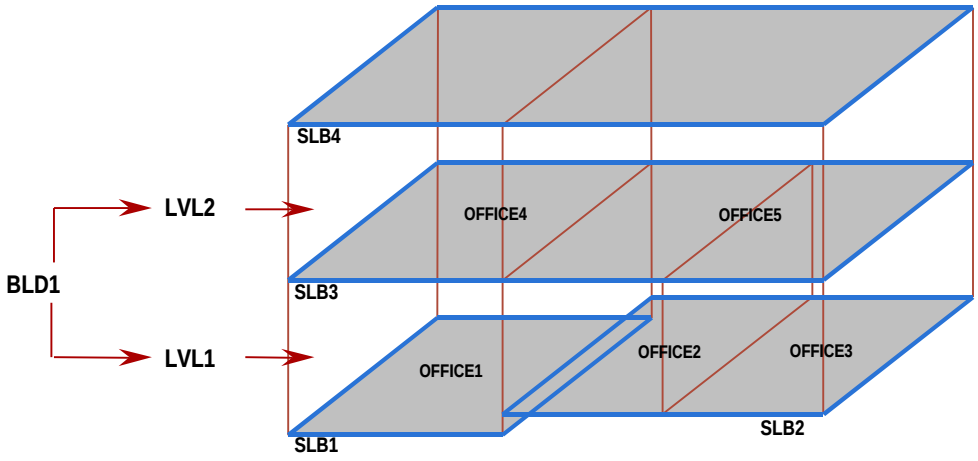

Figure 9 - A sample building for demonstrating the PDB representation.

A level is more of a functional than a physical grouping of spaces, i.e. spaces at different physical heights (from the ground plane) can still be considered as one level. In this respect, the concept of a level is similar to that of a floor plan as defined by the architect—it can accommodate slight changes in floor heights of the spaces that are considered to "belong" to that floor.

A level has two main components—*spaces* and *slabs*. There is no predefined correspondence between these two elements; one cannot be derived from the other. Consider the example of the building BLD1 shown in Figure 9. The level LVL1 consists of three offices on two different floor slabs but with the same ceiling slab. LVL2 consists of two offices with the same floor and ceiling slabs. (We consider stairs and staircases to be a special element, and will therefore ignore them for now.)

For a given LEVEL entity, we would be interested in knowing what spaces it is comprised of, as well as which are the physical slabs that make up that level. Therefore, we store the first space and the first floor slab in the LEVEL table as shown in Table 2. (It does not matter which space or slab is indicated as the first.) The

responsibility of pointing to the next space or slab in the list is delegated, following the same principle mentioned earlier, to the SPACE and SLAB entities respectively.

Table 2 - The LEVEL table.

Level ID	Building	Next Level of Building	First Space	First Floor Slab
LVL1	BLD1	LVL2	OFFICE1	SLB1
LVL2	BLD1	-	OFFICE4	SLB3
...	...	...	...	...

The LEVEL entity also includes a back pointer to indicate the building it belongs to. This information can be derived from the BUILDING table by doing a search, but it is more convenient and efficient to store it with each level.

6.3.2. *Spaces, Surfaces, Walls, and Openings*

One of the two basic components of the LEVEL entity is the space. What defines a space are its *surfaces*, both vertical and horizontal. A vertical surface is essentially a wall surface—there will necessarily be many of these. For a given space, if we list its first surface (again, it does not matter which is designated as the first), we can associate a next surface to each one of the surfaces and thereby collate, in a sequence, all the vertical boundaries of the space (Table 3). For horizontal surfaces, we assume (in this version) that there will be only one floor surface and one ceiling surface associated with every space. The area and volume of the space can be derived once all its surface information has been obtained. Figure 10 illustrates all the surfaces for the two spaces on the second level of the building.

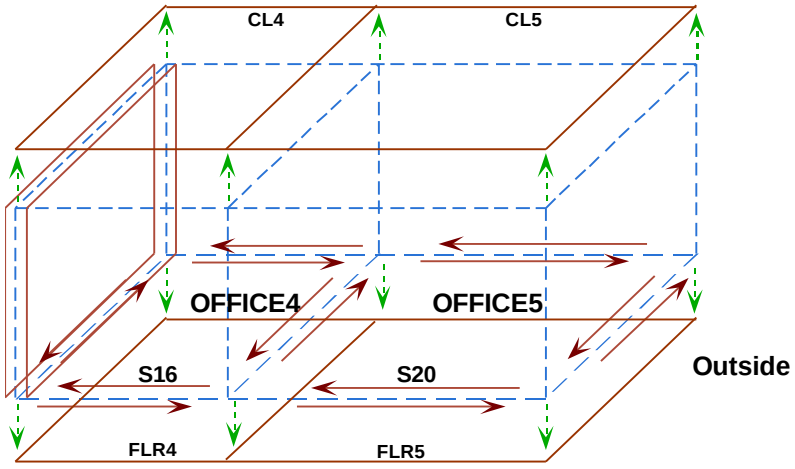

Figure 10 - Defining a space by its surfaces.

Table 3 - The SPACE table. *(Italicized entries in the tables indicate that these are not shown in the illustrations.)*

Space ID	Level	Next Space on Level	First Surface	Floor Surface	Ceiling Surface	Space ODB Link
OFFICE1	LVL1	OFFICE2	*S1*	*FLR1*	*CL1*	*tag name*
...	...	...	...	...	...	...
...	...	...	...	...	...	...
OFFICE4	LVL2	OFFICE5	S16	FLR4	CL4	*tag name*
OFFICE5	LVL2	OFFICE4	S20	FLR5	CL5	*tag name*
Outside	-	-	*S9*	-	-	-

As shown in Table 3, the outside of the building is included as a space since the outside surfaces of a building are also important and need to be defined.

Floor and ceiling surfaces, on their part, have a link to the slabs they rest on or hang off respectively. The only other information they include, as shown in Table 4, is their distance from the associated slab. Their geometric location and configuration can be derived (as the intersection of the vertical wall surfaces with the horizontal slabs) so it does not need to be explicitly stored. It is the slab information associated with the floor and ceiling surfaces that allows the vertical position and the height of a space to be determined.

Table 4 - The FLOOR/CEILING table.

Floor/Ceiling ID	Space	Slab	Distance from Slab	Floor/Ceiling ODB Link
...	...	...	...	...
FLR4	OFFICE4	SLB3	0.25	*tag name*
CL4	OFFICE4	SLB4	1.50	*tag name*
...	...	...	...	...

In Version 1 of the PDB (Khemlani & Kalay 1997), there were two equal and opposite segments (i.e. edges) representing the wall surfaces on either side of a wall. Each segment was defined by a single vertex; the second vertex that came from the opposite segment defined both segments geometrically, and in turn also defined the wall surface which was assumed to be always vertical. The opposite segment information also enabled space adjacency information to be very easily derived.

However, that surface representation did not efficiently deal with a situation such as the one shown in Figure 11 (a different example from the sample building used so far), where a single space on one side of the wall adjoins three spaces on the other side of the wall. It would have broken up the continuous wall surface S1 into several smaller surfaces, both horizontally and vertically. Rather than break up a single surface into many parts in this manner, Version 2 introduces a purely geometrical entity called the edge. An edge, just like the segment of Version 1, has an equal and opposite edge and is defined by a single vertex. A surface is now defined by a *collection of edges* rather than by a single edge, and space adjacency information can still be conveniently obtained by the equal and opposite edge data. A simple function translates the sequence of edges into a physically continuous surface with a specific location and area; the total vertical height of a surface remains the same as the height of the space it is part of.

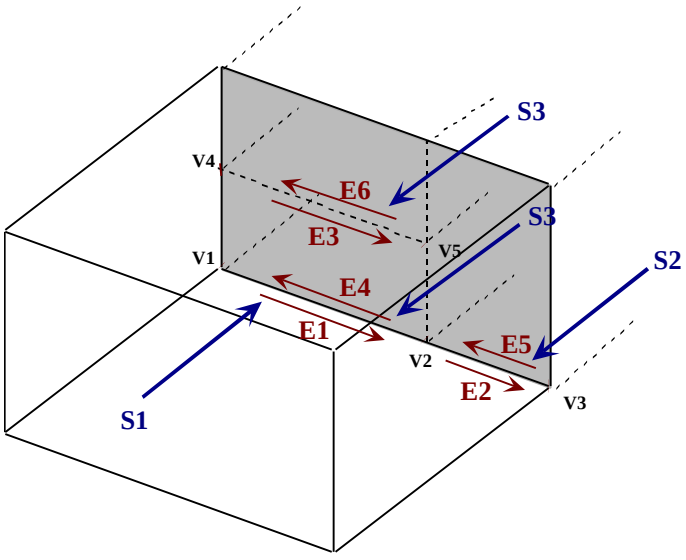

Figure 11 - A situation where a single surface on one side of a wall adjoins three surfaces on the other side.

This situation shown in Figure 11 does not occur in the sample building, therefore each surface is defined only by a single edge as shown in Figure 12.

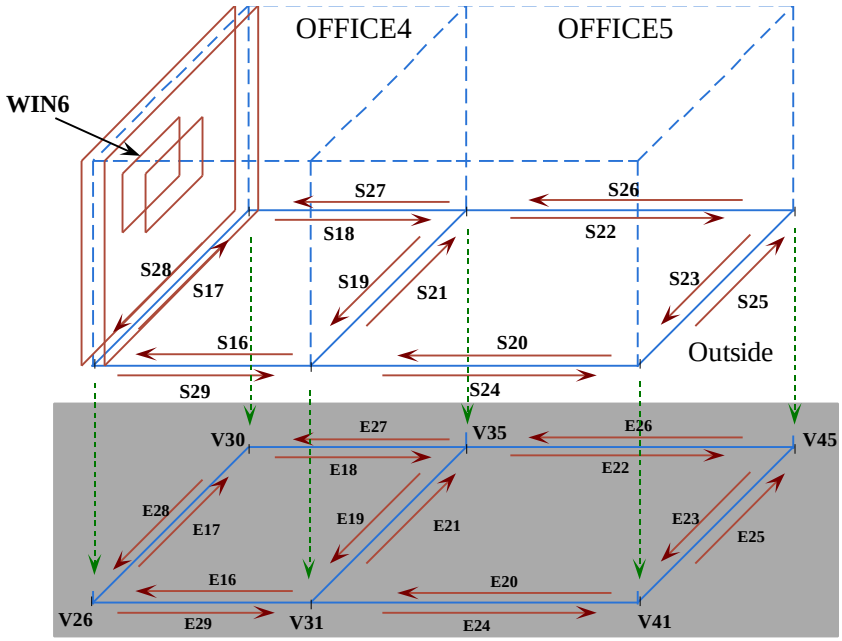

Figure 12 - Using edges to define surfaces in the sample building.

Surfaces do not exist in isolation. They have *openings* for doors and windows, and are also associated with *walls*. Opening information is attached directly to a surface while wall information is associated with it through an edge. The relational tables for the SURFACE and EDGE entities are given in Tables 5 and 6 respectively. Every surface carries information about the space it belongs to, the next surface for that space, a first edge, and a first opening. Because it is also a building element, it has a tag that links it up with the corresponding ODB. Every edge indicates what surface it belongs to, the vertex which locates it geometrically, the next edge for that surface, the opposite edge, and the wall it is attached to (see Figure 13).

Table 5 - The SURFACE table.

Surface ID	Space	Next Surface in Space	First Edge	First Opening	Surface ODB Link
...	...	...	...	...	...
S16	OFFICE4	S17	E16	-	*tag name*
S17	OFFICE4	S18	E17	WIN6	*tag name*
...	...	...	...	...	...
S28	Outside	S29	E28	WIN6	*tag name*
S29	Outside	S24	E29	-	*tag name*

Table 6 - The EDGE table.

Edge ID	Surface	Vertex	Next Edge on Surface	Opposite Edge	Wall
...	...	...	...	...	...
E18	S18	V30	E18	E27	W11
E19	S19	V35	E19	E21	W13
E20	S20	V41	E20	E24	W10
E21	S21	V31	E21	E19	W13
...	...	...	...	...	

Figure 13 shows how the walls are represented. A wall is geometrically located by specifying its two vertices and its height. A single external wall can extend up to several levels, therefore the height of a space cannot be used to determine the height of a wall.

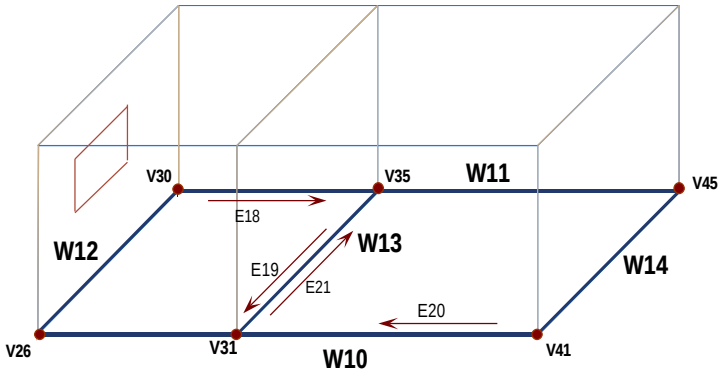

Figure 13 - The walls for the two spaces shown earlier in Figure 10.

Walls are also linked to the structure of the building by indicating if there is a beam running across the top of the wall. It is important to note that there is no one-to-one relationship between walls and beams—not all walls have beams above them, and not all beams have walls below them. If that was the case, we could have used one entity to derive the other. In the absence of such a relationship, we simply list the beam associated with a wall, if any, by having a "Beam" field in the WALL table, as shown in Table 7.

Table 7 - The WALL table.

Wall ID	Vertex 1	Vertex 2	Height	Beam	Wall ODB Link
...	...	...	...	...	...
W10	V26	V41	15	*B36*	*tag name*
W11	V30	V45	15	*B34*	*tag name*
W12	V26	V30	15	*B33*	*tag name*
W13	V31	V35	15	*B37*	*tag name*
W14	V41	V45	15	*B35*	*tag name*

The OPENING entity is relatively straightforward. Each surface carries the information of the first opening located within it; thereafter, each opening indicates which is the next opening on that surface. In this manner, all the openings located within a single surface can be collected. Also, a single opening will necessarily be associated with both surfaces that make up the pair. Thus, the opening WIN6 shown in Figure 12 is referenced by both surfaces S17 and S28 in their table entries. The OPENING table is shown below with the entries for this particular window.

Table 8 - The OPENING table.

Opening ID	Type	Surface	Next Opening	Width	Height	Distance from Surface Vertex	Opening ODB Link
...	...	...	...	...	...	...	...
WIN6	window	S28	-	8	6	7.5	*tag name*
...	...	...	...	...	...	...	...

As we have seen, there will always be a pair of surfaces referencing any given opening. But the single surface listed in the OPENING table determines which direction the door or window opens in. That is why S28 is the surface referred to by WIN6 rather than S17. The "Distance" field locates the opening geometrically on the surface by specifying the distance from the center of the opening to the vertex associated with that surface. The values in the "Width" and "Height" fields would be useful if the opening was not yet linked to a specific door or window object in the ODB, a situation very common in actual practice. Once the linking was done, however, the "Width" and "Height" values would be automatically replaced by those specified in the ODB for that particular object.

6.3.3. *Slabs, Beams and Columns*

The second main component of the LEVEL entity, as introduced earlier, is the slab. A slab can be represented in terms of the *beams* that support the slab. Figure 14 shows the slab SLB3 and the beams that support it. The

boundary of this slab can be obtained by collecting all its peripheral beams. Thus, we can simply define the first beam for every slab in the SLAB table as shown in Table 9, and associate a next beam with every beam in the BEAM table. For each slab, we also have a back pointer to indicate which level it belongs to.

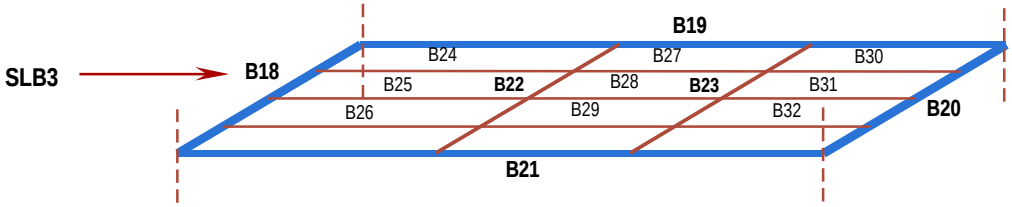

Figure 14 - The relationship between slabs and beams.

Table 9 - The SLAB table.

Slab ID	Level	First Peripheral Beam	Slab ODB Link
...	...	...	...
SLB3	LVL2	B18	*tag name*
...	...	...	...

For the BEAM entity, we can locate a beam geometrically by defining its two vertices as shown in Figure 15.

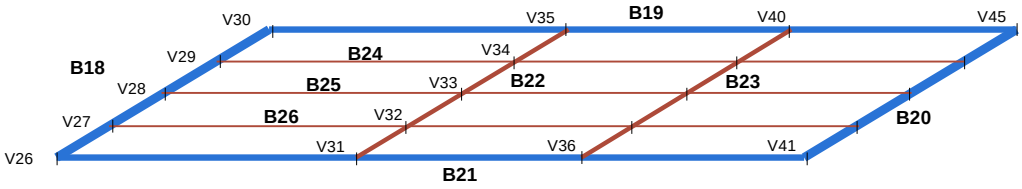

Figure 15 - Using vertices to define beams.

We have seen that we also need to associate a next beam with every beam to define the slab. However, there is a lot more information about the structure in Figure 15 that we need to capture. We would like to have the full picture—the primary beams that rest on columns, secondary beams that rest on the primary beams, tertiary beams that rest on the secondary beams, and so on. Trying to derive this information from the vertex information of the beams alone would be highly inefficient and tedious. However, we can make the BEAM entity carry some more information in addition to which is the next beam along the slab—such as the next parallel beam, and the next continuous beam (see the entries for beams shown in Table 10). It should be noted that all these lists are circular, so that "previous" beams for any category need not be stored.

Table 10 - The BEAM table.

Beam ID	Vertex 1	Vertex 2	Next Beam along Slab	Next Parallel Beam	Next Continuous Beam	Beam ODB Link
...	...	...	...	...	...	...
B18	V26	V30	B19	B22	-	*tag name*
B19	V30	V45	B20	B24	-	*tag name*
B20	V45	V41	B21	B18	-	*tag name*
B21	V41	V26	B18	B19	-	*tag name*
B22	V31	V35	-	B23	-	*tag name*
B23	V36	V40	-	B20	-	*tag name*
B24	V29	V34	-	B25	B27	*tag name*
B25	V28	V33	-	B26	B28	*tag name*
B26	V27	V32	-	B21	B29	*tag name*
...	...	...	...	...	...	...

The final building component of our basic representation is the column. A column can simply be represented as a sequence of *vertices* as shown in Figure 16. Thus, we can associate a starting vertex to every column, and let each vertex carry the information about which is the next vertex in the vertical sequence. (There will be only

one such vertex in the vertical direction unlike in the horizontal direction, where there can be many vertices that can qualify as the next vertex of a particular vertex.)

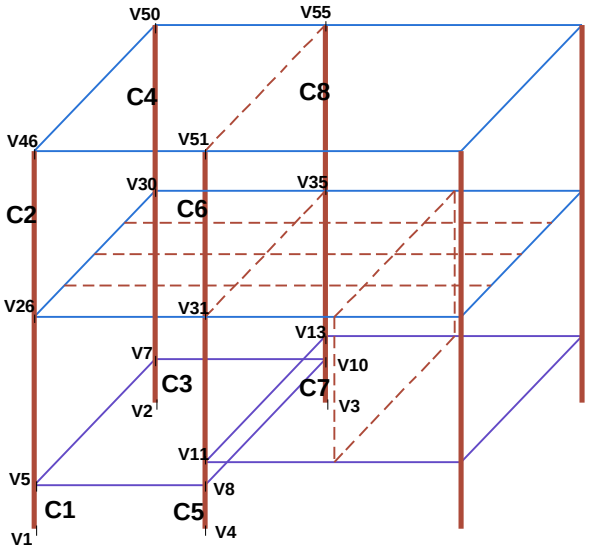

Figure 16 - Columns as defined by vertices.

The only complication arises in the case where one vertex is shared by two columns. For instance, the vertex V5 in Figure 16 is shared by the columns C1 (a foundation column) and C2 (a superstructure column). Therefore, it also become necessary to indicate for every column, the last vertex in addition to the first vertex, as shown in the Table 11 for the COLUMN entity.

Table 11 - The COLUMN table.

Column ID	First Vertex	Last Vertex	ODB LINK
C1	V1	V5	*tag name*
C2	V5	V46	*tag name*
C3	V2	V7	*tag name*
C4	V7	V50	*tag name*
C5	V4	V8	*tag name*
C6	V8	V51	*tag name*
C7	V3	V10	*tag name*
C8	V10	V55	*tag name*
...	...	...	...

6.3.4. *Vertices*

All that remains now is the specification of the vertices. Each vertex carries information about the next vertex in the vertical sequence. Also, if there is a column passing through that vertex, this information is indicated as well. This is in order to link the space and the enclosure more closely together. Since the other entities such as surfaces, walls, and beams all reference a vertex, if the vertex carries the column information with it, the columns can be directly linked to all the other entities. In the case where a vertex is shared by two columns such as V5, it will, as a rule, list the column below it; the column above it can be identified by simply going to its “next” vertex.

The vertex entries for some of the vertices shown in Figure 16 are given in Table 12. For the sake of conceptual clarity, the actual vertex coordinates are listed in a separate table (not shown here).

Table 12 - The VERTEX table.

Vertex ID	Next Vertical Vertex	Column
V1	V5	C1
V2	V7	C3
...	...	...
V5	V26	C1
...	...	...
V26	V46	C2
...	...	...
V46	-	C2
...	...	...

6.3.5. *The Overall Picture*

The overall data schema of the PDB, showing how all the entities are related to each other, is illustrated in Figure 17. (Back pointers between entities are not shown.) All the entities fall into either one of three broad categories: Architectural, Structural, or Geometrical. Wherever appropriate, architectural and structural elements are linked to the corresponding ODBs.

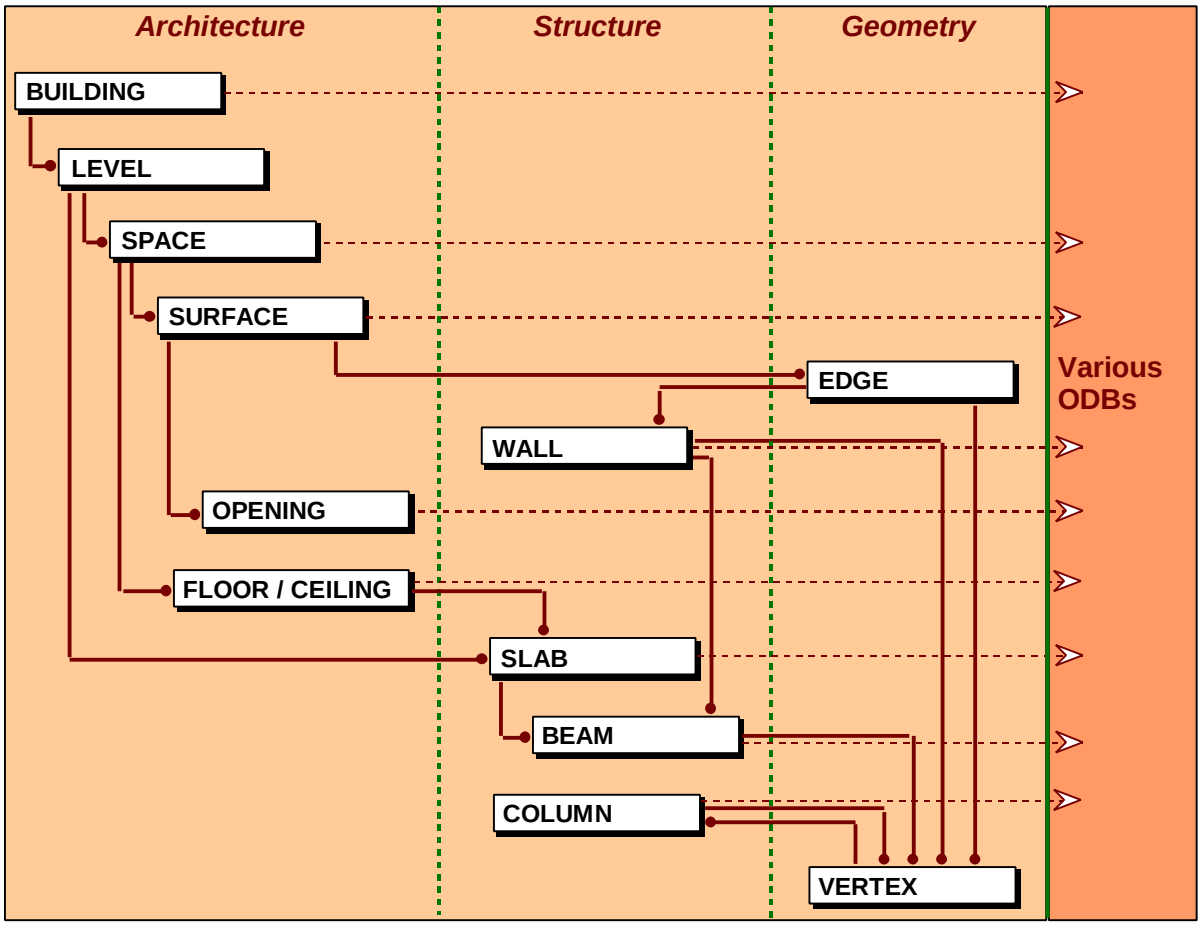

Figure 17 - The overall data schema of the PDB.

6.4. Implementation of the PDB

We had developed a CAD editor which could be used to create a graphical description of the building in Version 1 of the PDB (Figure 18). We are currently updating it to work with Version 2. The attempt was not to re-create a full-blown, sophisticated drafting or modeling software such as those available commercially. Rather, we needed a reasonably efficient and user-friendly representation tool that would enable PDBs of buildings to be quickly generated for testing by various evaluation programs. It was particularly important for the editor to provide the facility of carrying out a variety of semantically-meaningful operations that currently available general-purpose CAD programs do not provide. For example, deletion of the peripheral room of a building should automatically update the walls of the adjoining rooms from being "internal" to "external" walls.

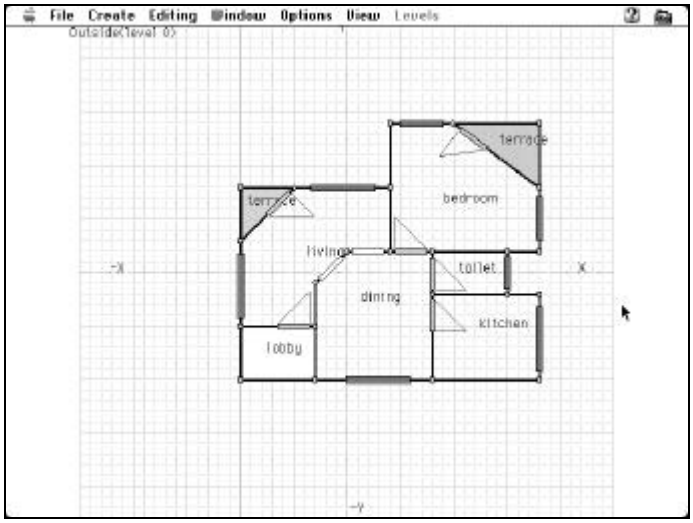

Figure 18 - The customized graphical editor implemented to describe the PDB of a building.

Internally in the editor, all the PDB entities are stored as linear or circular linked lists. The relational tables shown in the preceding sections are generated for query purposes only (i.e., as input to evaluation programs) rather than for update operations.

In the current implementation of the editor, the links from the PDB to the various ODBs are made by associating "types" to each of the PDB entities—levels, spaces, walls, surfaces, columns, and openings. Figure 19 shows this link being made through a dialogue box that opens up by clicking on one of the spaces.

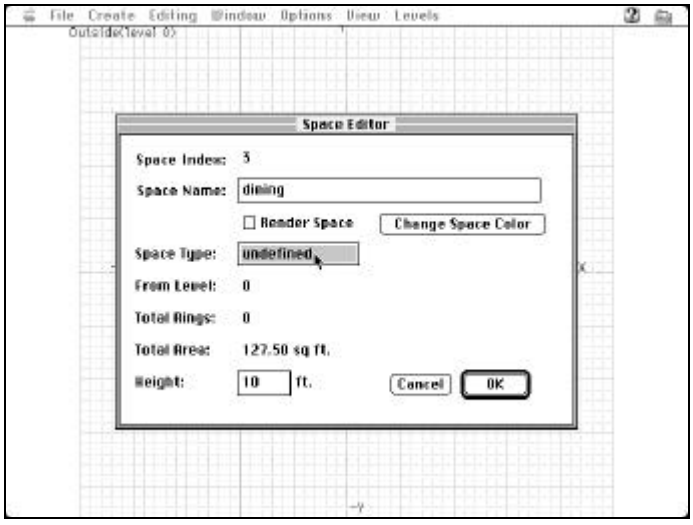

Figure 19 - Linking elements of the PDB to various ODBs by attaching "types" to entities.

In future implementations, we envision the link between the PDB and the ODBs to be made differently. On the graphic editor screen where the PDB representation is being generated, there will be tabbed palettes of instantiable objects (walls, doors, windows, rooms, etc.). Objects will be grouped logically, e.g. classrooms in one palette, offices in another, sanitary equipment on a third, and so on. Each symbol in the palettes will represent and will be linked to one object in an ODB. Users can link a graphic entity in the PDB directly to a generic object located at any level of abstraction hierarchy. In this way all properties of the generic object become attributed to the graphic entity, and can be easily referenced by evaluation tools.

7. CONCLUSION

In conclusion, we will evaluate our representation against the criteria we used to define an "intelligent" building representation for computer-aided building design. Our solution is informationally complete because it can represent both the spatial and the structural aspects of a building. It is general-purpose, because it does not matter what the building type or use is. The data structure of the PDB which connects the elements together into the appropriate assembly ensures well-formedness. And finally, the representation is semantically rich because there is information about the building assembly as well as information about each of the individual building elements. Based on the separation of the building assembly from its components, this representation can adequately support the needs of design development, analysis, and evaluation at the crucial stage of schematic design.

We have already prefaced our work with the disclaimer that it is not targeted for that fuzzy, "pre-evaluable" design phase in which myriad shapes and forms are being conceived, many of which could be quite fantastic and totally unbuildable. Our representation has a fixed assembly structure, which it must have in order to represent a building at a stage where it *is* definitely a building and nothing else. Of course, the majority of buildings which are structurally straightforward could be conceived directly in our representation, using our CAD editor. However, complex buildings (such as the famous Sydney Opera House, for instance) could be conceptualized using some other medium which has no limitations on shape, and then translated in a simplified form to our representation for the purpose of evaluation. Thus, we are not proposing to restrict the designer's imagination in any way by purporting our representation as the only means for generating the design and our editor as the sole design medium.

8. FURTHER WORK

We have already tested the representation with some evaluation tools for architectural design (Khemlani & Kalay 1997, Lee 1997). These tools are just a precursor to more design and evaluation tools for other building-related disciplines that we intend to develop. We see the tool-building process as one of the best ways to ratify and further develop the underlying data structure of the representation. To this end, we plan to invite various disciplinary experts to use our proposed representation as the basis for the development of their design aids.

At the same time, we are continuing to develop further both the PDB and ODB components of the database. In particular, the structural solution for the PDB is being considerably reworked to make the representation more flexible and capable of representing more complex structural systems. It is important to emphasize that what is presented in this paper is work in progress. There is room for evolution and improvement, even substantial change, if inconsistencies or errors in our data schemes are revealed in the course of future implementation.

We are also working on an integrated building environment, which will include not just the PDB and the ODBs, but also a *context* database (CDB) and an *issues* database (IDB). We will develop *utilities* that overlay all these databases and provide application programs with the information they need, so that the creators of these programs do not need to be concerned about the internal data structures of the building representation.

We also intend to focus on the communication and control issues, knowledge representation issues, and user interface issues as the research progresses.

REFERENCES

Amor, R., G. Augenbroe, J. Hosking, W. Rombouts, and J. Grundy. 1995. "Directions in Modeling Environments." *Automation in Construction* 4(3), pp. 173-187.

Armour, G.C. and E.S. Buffa. 1968. "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities." *Management Science* 9(2), pp. 294-309.

Baer, A., C.M. Eastman and M. Henrion. 1979. "Geometric Modeling: a Survey." *Computer-Aided Design* 11(5), pp. 253-271.

Baumgart, B. 1972. "Winged Edge Polyhedron Representation." *Technical Report CS-320*, Stanford Artificial Intelligence Laboratory, Palo Alto.

Bjork, B.C. and J. Wix. 1991. "An Introduction to STEP." *Technical Report* jointly published by VTT Technical Research Center, Laboratory of Urban Planning and Building Design, Finland, and Wix McLelland Ltd., England.

Carrara, G. and Y.E. Kalay. 1994. "Past, Present, Future: Process and Knowledge in in Architetcural Design." *Knowledge-Based Computer-Aided Architectural Design* (G. Carrara & Y.E. Kalay, eds.), Elsevier Science Publishers, Amsterdam, The Netherlands.

Carrara, G., Y.E. Kalay and G. Novembri. 1994. "Knowledge-Based Computational Support for Architectural Design." *Knowledge-Based Computer-Aided Architectural Design* (G. Carrara & Y.E. Kalay, eds.), Elsevier Science Publishers, Amsterdam, The Netherlands.

Coyne, R.D., M.A. Rosenman, A.D. Radford, M. Balachanran, and J.S. Gero. 1989. *Knowledge-Based Design Systems*. Addison-Wesley, Reading, Massachusetts.

Eastman, C.M. and A. Siabiris. 1995. "A Generic Building Product Model incorporating Building Type Information." *Automation in Construction* 3(4), pp. 283-304.

Fenves, S., U. Flemming, C. Hendrickson, M.L. Maher, R. Quadrel, M. Terk, and R. Woodbury. 1994. *Concurrent Computer-Aided Integrated Building Design*, Prentice-Hall, Inc. Englewood Cliffs.

Harfmann, A. and S. Chen. 1993. "Component-based Building Representation for Design and Construction." *Automation in Construction* 1(2), pp. 339-350.

Kalay, Y.E. 1987. "WORLDVIEW: An Integrated Geometric Modeling/Drafting System." *IEEE Computer Graphics & Applications* 2(7), pp. 36-46.

Kalay, Y.E. 1989. *Modeling Objects and Environments*, John Wiley & Sons, New York.

Kalay, Y.E., L. Khemlani and Jin Won Choi. 1996. "An Integrated Model To Support Collaborative Multi-Disciplinary Design Of Buildings." *Conference Proceedings of the First International Symposium on Descriptive Models of Design*, Istanbul, Turkey.

Khemlani, L. and Y. E. Kalay. 1997. "An Integrated Computing Environment for Collaborative, Multi-Disciplinary Building Design." *CAAD Futures 1997*, Munich, Germany.

Lee, Choong-Hoon. 1997. "Evaluation of Social and Cultural Performance of Modern Korean Housing Design." *Masters Thesis*, University of California at Berkeley.

Shaviv, E. and Y.E. Kalay. 1991. "Combined Procedural and Heuristic Method for Energy Conscious Design and Evaluation." *Evaluating and Predicting Design Performance* (Y.E. Kalay, ed.), John Wiley & Sons, New York.

Shaviv, E. and G. Shaviv. 1977. "A Model for Predicting Thermal Performance of Buildings." *WP ASDM-8 Report*, Faculty of Architecture and Town Planning, Technion, Haifa, Israel.

Richens, P. 1977. "OXSYS-BDS Building Design Systems." *Bulletin of Computer-Aided Architectural Design* 25, pp. 20-44.