AN INFORMATION MODEL FOR MULTIPLE VIEWS OF BUILDINGS

Hugues Rivard[1], Steven J. Fenves[2], and Nestor Gomez[3]

Civil and Environmental Engineering Department
Carnegie Mellon University
Pittsburgh, PA, 15213

ABSTRACT: The paper outlines an information model that organizes the wealth of data used and generated during the conceptual design stage. The building is represented as an assembly of entities with relationships among them. Each entity represents a meaningful concept to design participants such as a beam, a room or a structural frame. Each entity contains data about its design aspect, its function aspect and its behavior aspect. Furthermore, each entity stores its geometry, its topological relationships with other entities, its aggregation relationship (made-of and part-of) and a reference to the technology (a set of knowledge and procedures) that is used to derive it. The geometry and topological relationships for the entity are obtained from a non-manifold skeletal geometrical representation common across all views. Multiple views representation is supported by dividing the attributes of an entity into small cohesive subsets, which we call primitives. These primitives are then used as construction blocks to present different views of the entity. The goal of this representation is two fold: to support case-based reasoning and to store the design data as it is generated during the conceptual design.

1. INTRODUCTION

In this paper, we present a proposed information model for the Configuration module of SEED (Software Environment to support the Early phases in building Desig ' currently under development at Carnegie Mellon University [Flemming et al. 93]. This module supports the generation of a 3-dimensional configuration of spatial and physical building components based on schematic layouts [Flemming and Woodbury 95]. The objective of this information model is two-fold. First, it records the design data as it is generated during schematic configuration design. Second, it serves as the foundation for case-based design, allowing building designers to retrieve and adapt previous designs as an aid in solving the current design problem.

Building designers need to be able to retrieve and reuse cases throughout the design process from architectural programming to detail design. The retrieved cases must match the desired level of detail. For instance, in the early design stage, the architect may be interested in an overall summary of building cases while in a more advanced design stage the structural engineer may be interested in the design of a particular floor slab. The retrieval of cases should support different levels of problem definition according to the designer's need.

Several participants are involved in the building design process and each has a different perception of the evolving product. A building information model needs to integrate all the views

1. Research Assistant, hr29@andrew.cmu.edu
2. Sun Company Professor, sf18@andrew.cmu.edu
3. Research Assistant, ng27@andrew.cmu.edu

of the design participants in order to ensure compatibility, reusability and integrity of the data. Such an information model fosters efficient data communication between participants throughout the full life cycle of the building and would have a positive impact on productivity, costs and quality.

2. BUILDING ENTITIES

Buildings are made of entities. An entity is a distinguishable object meaningful to a building designer. An entity can be a system, a sub-system, a component, a part, a feature of a part, a space or a joint [Gielingh 88]. An entity can then be a building itself or a building component at any level of decomposition, from a building wing to a nail. For instance, a building, a building wing, a room, a frame, a beam and a bolt are all entities.

An entity is an object that is to be represented in the database [Date 95]. Information about these entities needs to be recorded for archival purposes, for communicating design decisions, for obtaining approval by the owner and so on. The entities have to be unique across the database. To do so, each entity has a unique identity which uniquely identifies the entity during its lifetime. An entity includes four necessary types of data to fully describe it: attribute-value pairs, geometrical description, relationships and technologies. Zamanian had identified the two first types which he called functional and spatial data [Zamanian 92]. We add relationships to allow the representation of interactions among entities, and technologies to record references to the procedures that are used in designing the entity. Figure 1, below, shows the four types of data and their subcategories. The four types of data are explained in greater detail in the following four sub-sections.

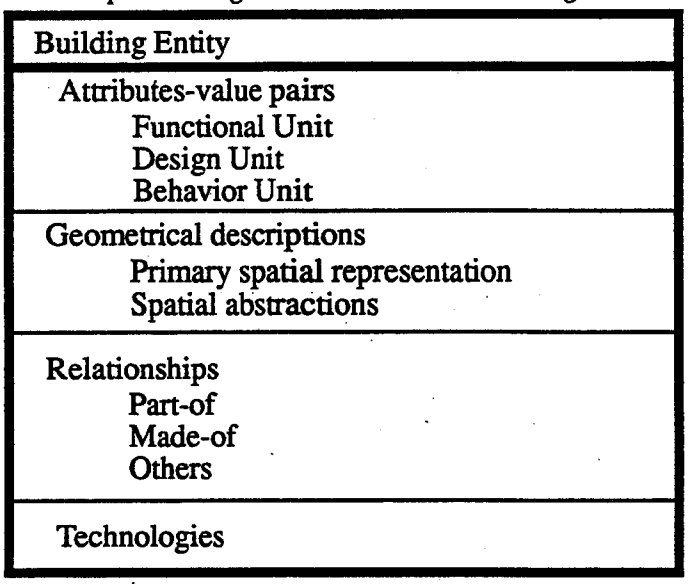

Figure 1. A building entity description.

2.1 Attribute-Value Pairs

Designing involves the generation of design descriptions of a potential entity intended to satisfy the specified qualities to be exhibited by that entity [Coyne et al. 90]. These design descriptions and requirements (i.e. qualities to be exhibited), as well as the behavior of the artifact, can be represented as attribute-value pairs where the value may be an atomic type (i.e. integer, float, string, boolean and so on), a matrix, a derived value (which may depend on other attribute-value pairs), and so on. In this model, a value cannot be a geometrical description, a relationship with another entity nor a reference to a technology.

The attribute-value pairs characterizing an entity are grouped into three subsets: the functional aspect, the design aspect and the behavior aspect. The functional aspect includes the intended purposes, the requirements and the constraints on the entity; this aspect is called the functional unit. The requirements have to be satisfied to realize the intended purpose. The functional unit can be seen as a design-problem statement [Gielingh 88]. The design aspect includes all the physical and spatial characteristics that define the actual design of the entity; this is called the design unit. The design unit can be seen as the solution to the design-problem. The design aspect can be compared with the original requirements to verify compliance. The behavior aspect includes the response to stimulations associated with different design conditions, and is called the behavior unit.

The multiplicity of the three units for a given entity is as follow:

- an entity may have only one functional unit,
- it may have several design units corresponding to different alternatives, but it may only have one current design unit at any given time in a given design state, and
- it may have several behavior unit corresponding to different design conditions.

These three sets of data are necessary to completely define an entity in terms of what it is intended for, what it is and how it responds throughout its working life. This represents a richer data model than the ones supported by current CAD systems which store only the design results, since the reason why a particular design was selected can be understood and justified from its function, behavior and instantiating technologies.

2.2 Geometrical Descriptions

We are following the approach developed by Zamanian for the geometrical description of the entity [Zamanian 92]. This description is classified in two categories: the primary spatial representation of an entity is its high-level geometric description which is used primarily for reasoning about its topological relations with other entities, and the spatial abstractions of an entity are its discipline-specific geometric representations. Each entity has only one primary spatial representation, but it may have several spatial abstractions. This representation scheme relies on the main premise that "topological relations of physical entities are invariant with respect to their discipline-specific spatial abstractions". [Zamanian 92]

The spatial extent of the primary spatial representation of an entity is defined by superior elements. These superior elements are "reference geometric entities which can be linear or curved, arranged in orthogonal or arbitrary directions, or be represented by zero- or higher-dimensional geometric entities" [Zamanian 92]. The superior elements act as grids or boundaries and hence formalize an intuitive and common technique where such elements are used to identify and specify the spatial extent of individual entities or group of entities.

The non-manifold boundary representation scheme is used because topological relations can be investigated without being affected by the various dimensionalities used in representing the geometric entities. Since we often have to deal with line, plane and volume representations at the same time (for beams, slabs and rooms for instance), the selection of this modeling scheme is justified. This scheme has the ability of "modeling and reasoning about mixed-dimensional geometric models in a single, uniform paradigm" [Zamanian 92]. The non-manifold scheme has also the advantage of being able to model "non-solid" objects.

2.3 Relationships

Every entity has some kind of interaction with other entities that need to be represented. "Any dependency between two or more entities is a relationship" [Rumbaugh et al. 91]. Typical

relationships include directed actions (supports, drives), communication (talks to, controls), ownership (has, part of) and so on. "Relationships are just as much a part of the data as are the basic entities" [Date 95], hence they should be represented as well in the information model. Furthermore, relationships should be bidirectional meaning that they must be traversable in either direction.

We isolate the aggregation relationships from others to ensure better access. The aggregation relationship is very important in describing buildings because it captures the link between entities and their components. Building entities are usually complex objects built from component entities which may be complex themselves. The type of relationship which exists between a complex object and its component objects is called an aggregation. The complex object is treated as a unit in many operations, although physically it is made of several component objects. The aggregation relationship can be recognized by the phrase "part-of", which can be used to describe the link between the component and the complex object, and "made-of", which can be used to describe the opposite link between the complex object and the component. Other relationships are frequently needed as well. Examples are: supports, encloses, connects, drives and controls. Topological relationships, such as above, next to, contained in and beside, are not stored explicitly in this information model since they can be obtained directly from the geometric modeler.

2.4 Technologies

We want to record how an entity has been designed in order to be able to make inferences about the processes used to design it. We also want to be able to adapt and redesign an entity rapidly without having to start from "scratch". To support this, we organize the design knowledge into a hierarchically structured technology tree, where each node of the tree represents a known alternative, the constraints that determine its applicability, the computational steps necessary to assign values to the attribute(s) defining that alternative, and the elements of the more detailed alternatives at the succeeding level [Fenves et al. 95]. Then, the design of an entity can be simply described by referring to the technology node that created it.

A technology node is viewed as "a collection of computational mechanisms that creates, details and instantiates entities to satisfy the requirements defined in the functional unit of an entity in a design context based on a specific construction technology or form generation principles" [Woodbury and Fenves 94]. Technology nodes are organized in the form of a tree. In the structural design domain, the technology tree represents the various alternative structural system, subsystem and component types available to the designer. The root of a tree operates on an abstract building as a whole, while succeeding levels of nodes operate on more and more specific building elements. Hence, the technology tree may deal with elements ranging from the most abstract (e.g., a full 3-D building for which a tube structure may be an alternative structural system) to the most specific elements (e.g., individual beams or even connections, reinforcement, etc.). Each node in the technology tree contains constraints on its applicability. If the constraints are satisfied, the node defines procedures either to assign attributes and attribute values to the design unit of a current entity, or to subdivide an entity into constituent entities. Hence, nodes in the technology tree may be categorized in two types: a technology refines another if it provides additional level of detail by adding new attribute-value pairs to the design unit of the current entity, while a technology elaborates another if it subdivides the current entity into constituent sub-units by creating new entities and linking them to the current entity through the "Made-of" relationship slot.[Fenves et al. 95].

A sample technology tree segment for slabs supported on steel framing elements is depicted in Figure 2. Each child technology of "Decking" shows two sets of constraints (the maximum and minimum spans and the maximum and minimum loads) which determines the technology's range of applicability. If the constraints are satisfied, the technology node generates an alternative which may be selected later by the designer. The constraints shown in the figure were taken from

current product catalogs; if a designer disagrees with the constraints provided, he or she can modify them. The technology node "One-Way-Deck-on-Joists" is an example of an elaborating

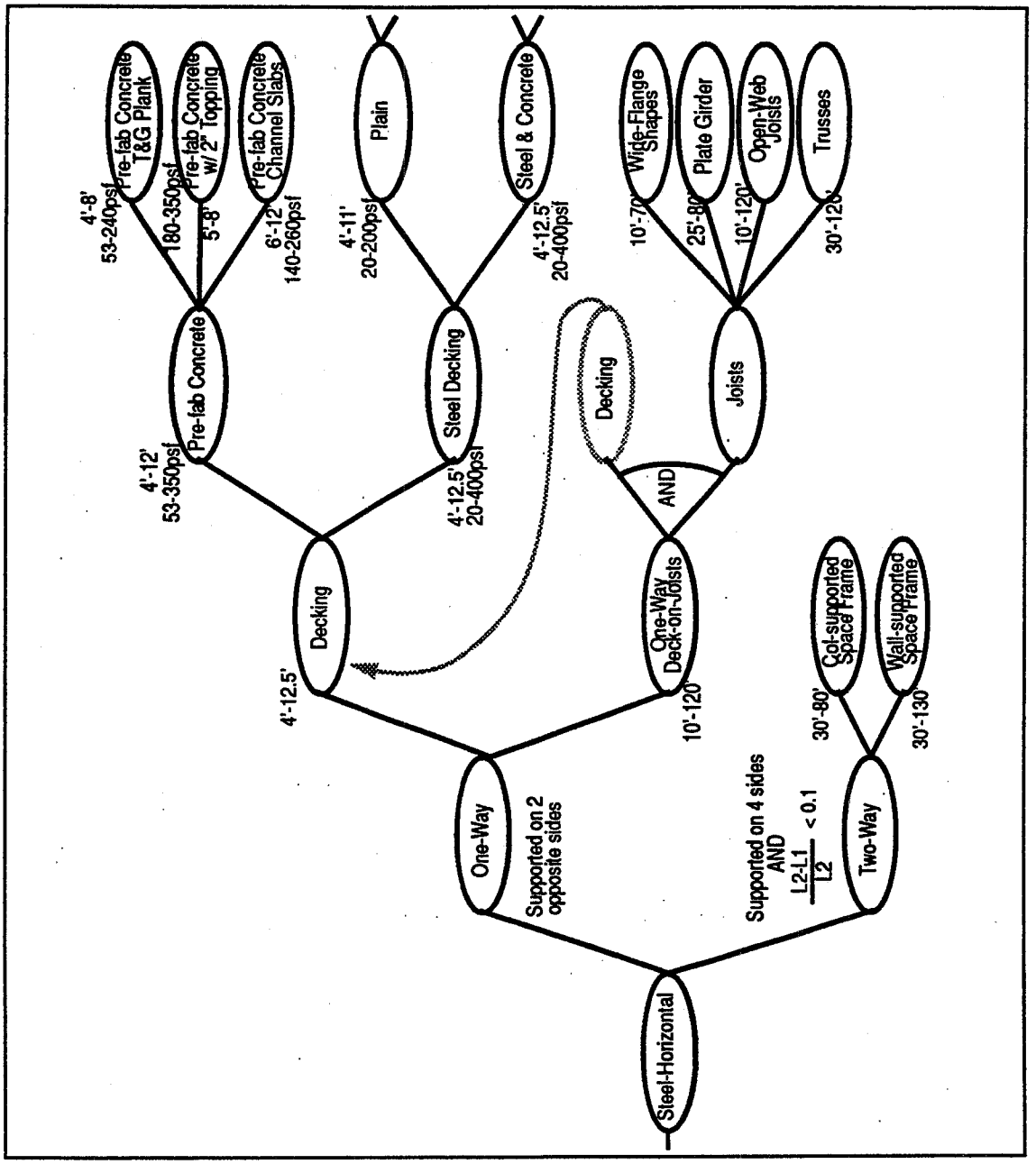

Figure 2. A sample partial structural technology tree [Fenves et al. 95]

technology which subdivides a slab entity into its constituent decking and joists entities. The dashed node "Decking" shown as one elaboration component of "One-Way-Deck-on-Joists" indicates the same subtree as "Decking", the child node of "One-Way". Thus, subtrees in the technology can be used as elements of other technologies.

3. HIERARCHICAL DECOMPOSITION

When faced with a complex design problem, a designer usually solves it by reducing it into a set of smaller more manageable sub-problems. These sub-problems are, in turn, such that a

solution can be easily determined. This "divide-and-conquer" strategy is typical for most design-processes [Gielingh 88]. The information model must therefore be able to support the decomposition of design problems. Such hierarchical decompositions are used to divide and conquer problems and to distribute tasks and responsibilities as well [Gielingh 88].

A building can be considered as composed of four major systems [Rush 86]: structure, enclosure, services and interior. These systems can be further decomposed into more detailed hierarchical levels of subsystems, components, etc. all the way down to distinct materials. Figure 3 shows the hierarchical decomposition of the building structural system. The structure of a building may sometimes be broken down into independent sub-structures depending on the building's complexity. These sub-structures occur in buildings with expansion joints or in buildings made of several independent structural systems. The self-standing structure is decomposed into 2-D structural elements such as frames, walls and slabs. The 2-D structural elements are further decomposed into 1-D elements or 2-D sub-elements (i.e. beams, columns and slab elements). Note that the figure does not show all the relationships which may exist among the entities shown (i.e. support, connected to and so on). The black circle at the end of a link indicates that many entities may be linked to the entity at the other end of the link.

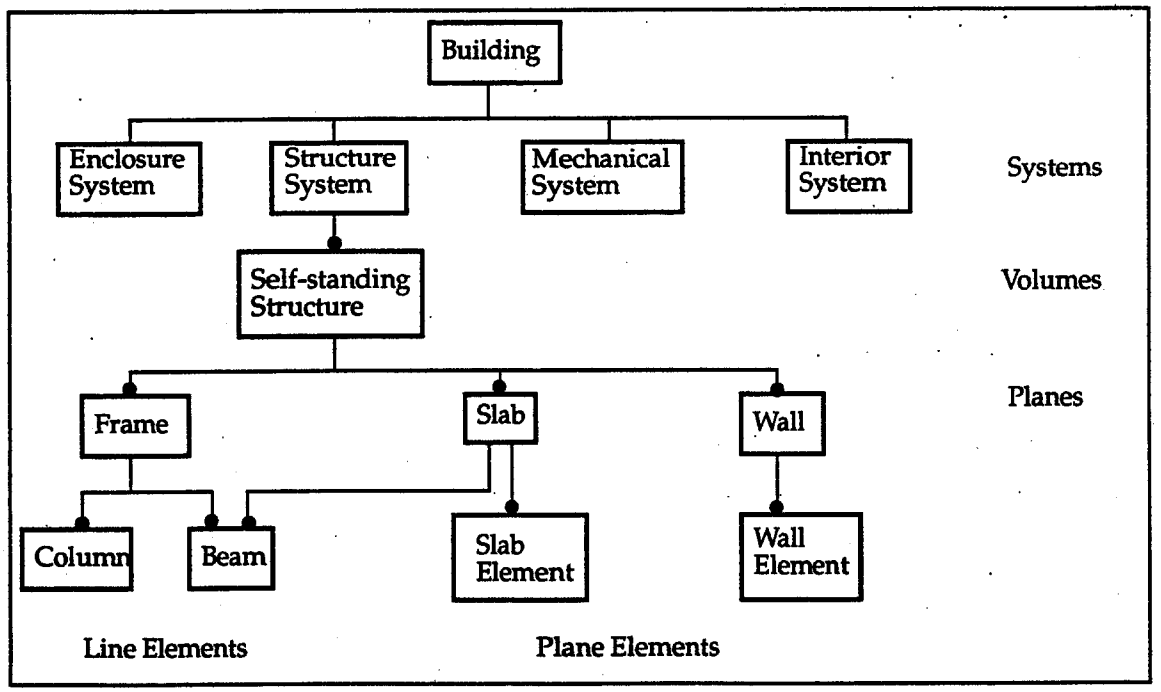

Figure 3. Hierarchical decomposition of the building structure.

This hierarchical decomposition of the structural system follows the total-system approach promoted by T. Y. Lin and S. D. Stotesbury where the designer focuses first on the three-dimensional implications of architectural space-form options, then on the more-or-less planar sub-systems which make up the structure, and finally on the elaboration and refinement of individual elements and connection details [Lin and Stotesbury 81]. Hence, a complex structural problem is decomposed into simpler sub-problems that can be considered in a semi-independent fashion. This "approach reflects the organic concept that the whole (of a design scheme) should give rise to the need for details and not vice versa" [Lin and Stotesbury 81].

A similar hierarchical decomposition has been developed for the enclosure system by Rivard et al. [Rivard et al. 95]. The enclosure system of a building is decomposed into envelope planes such as roofs, exterior walls, slab on grade and cantilevered floors. Each envelope plane can be

subdivided into envelope areas, each of which has one envelope section and corresponds to one indoor space, and can be pierced by openings. An envelope section is a sequence of envelope

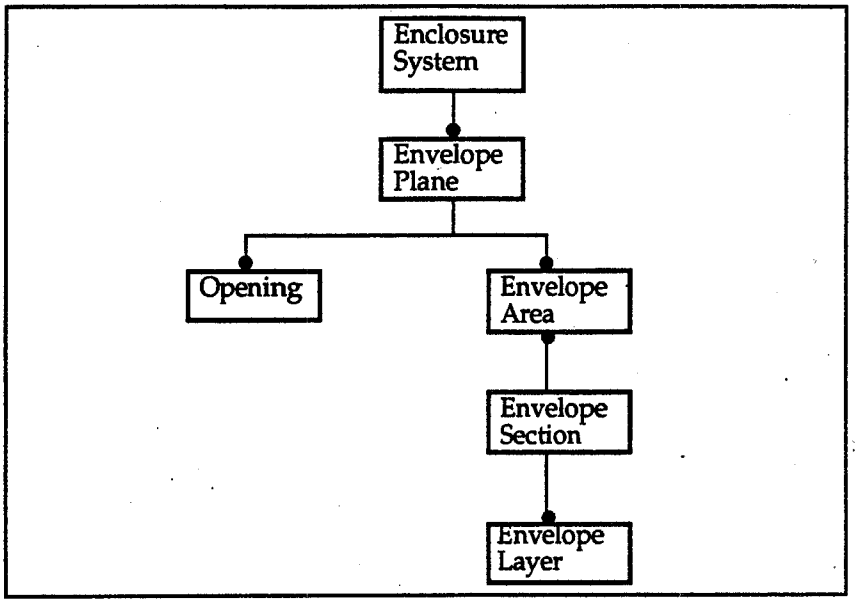

Figure 4. Hierarchical decomposition of the enclosure system

layers (i.e. construction products) such as cladding, membrane, insulation and finishing.

The information model, presented here, supports such hierarchical decompositions. Systems and subsystems can be modeled as entities linked together by aggregation relationships. The result of the hierarchical decomposition is a tree of entities. The designer can look at the system at any level of abstraction simply by going to the corresponding depth in the tree. For instance, he or she can look at the structure as a whole or as planes or as beams, columns and slab elements.

The information model also supports the design process as it unfolds by allowing designers to populate the database in an intuitive manner from global system entities, to sub-system entities all the way down to component entities. The root of a hierarchical decomposition tree is recorded first, followed by the system entities and so on. New entities are added to the existing ones, through aggregation relationship, as the design proceeds and becomes more and more detailed.

4. INTEGRATION OF MULTIPLE VIEWS

The entities presented in the previous section represent a logical manner for organizing the building data and correspond to how designers see the building. But sometimes, it may happen that an entity occurs in two different hierarchical decompositions. For instance, a load bearing exterior wall is both an envelope plane in the enclosure system and a wall in the structural system. Hence, there is a need to represent the multiple views of an entity. In this section, we look at a mechanism to record the set of attribute-value pairs in order to be able to provide different views.

The attributes of an entity may be organized as follows:

- the collection of all attributes defining an entity may be grouped into one flat structure,
- the attributes may be divided into small cohesive subset, or
- each attributes may be represented as a distinct structure.

The first and last approach correspond to the two extremes of a scale. The first approach leads to the creation of exceedingly complex entities which are difficult to understand (for a single specialist), to maintain and to extend [Howard et al. 92]. In the third approach, at the other

extreme, each attribute is stored separately. The attributes are accessed by looking at their name. This approach leads to complex naming conventions.

The representation approach that we are investigating is located between these two extremes. We intend to divide the attributes of an entity into small cohesive subset each of which we call a primitive. This approach, called the Primitive-Composite Approach (or P-C Approach), was originally developed by Phan and Howard (1993). It is a data model and a structured methodology for modelling facility engineering processes and data to achieve integration. It has the following advantages: it supports multiple views, schema evolution and data integration.

Cohesion is the only criterion used in decomposing the entities and it is defined as a measurement that shows how closely the attributes of an entity relate to one another [Phan and Howard 93]. They characterized cohesion into five specific criteria: the data attributes are stored in one location (access-cohesive), related to the same concept (concept-cohesive), not derived from each other (source-cohesive), instantiated at the same time (time-cohesive) and used at the same time (use-cohesive). A functional and data flow analysis of the building design process is needed to evaluate the cohesion of the attributes of each entity.

Our definition of a primitive is slightly different. We keep only three of the five original criteria. Hence, a primitive is defined to be a group of closely related attributes which are found together in a repository (access-cohesive), which are instantiated at the same time (time-cohesive) and which corresponds to the same concept (concept-cohesive). The access-cohesive criterion ensures that attributes from different views are not put together (i.e. data found in structural drawings are not mixed with data found in construction estimates). The time-cohesive criterion ensures that if a user tries to access some data and sees that the corresponding primitive exists, he or she can assume that all the attributes in it have a value. If the primitive does not exist, it means that the data has not been generated yet, and he or she may create a new primitive. This criterion implies that each primitive is generated by only one technology node. The concept-cohesive criterion divides the attributes in at least three broad classes: function, design (form) and behavior which were presented earlier in section 2.1. This criterion may subdivide the attributes further if more concepts are considered.

The use-cohesive criterion is not considered for efficiency. It is difficult to predict all possible uses of an attribute, and hence the corresponding attributes may be subdivided too much. It does not really matter if all data attributes of a primitive are used at the same time or not. We do not agree with the source-cohesive criterion. We think that dependent data can be recorded in the same primitive. Methods could be used, from within the primitive, to compute the dependent information.

The primitive representation provides an abstraction of the entity to the designer. Views hide the actual complexity of the entity by providing only the relevant information while hiding the unnecessary details. Figure 5, below, shows a wall entity decomposed in a set of primitives. Three different views are shown as referring to a subset of the primitives. Two of the primitives are shaded to show the sharing of information between the different views. Hence, this model supports the integration of the various views required by the design participants. This characteristic ensures compatibility, reusability and integrity of the data. It also fosters efficient data communication between participants.

The definition of the primitives are inherited from a common structure which is shown in Figure 6. A primitive class should have a unique name. Each primitive stores the name of the person who is responsible for its creation. It stores the initial time when it is created and when it is last modified. It also stores a reference to the technology node that was used to instantiate it and a status which could take one of three values: candidate (alternative is not explored yet), explored (alternative has been explored but not selected) and committed (alternative represents the current

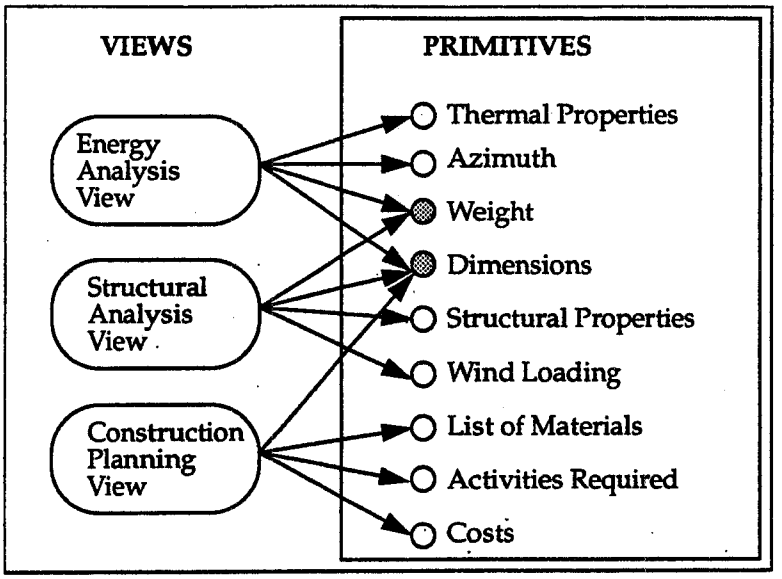

Figure 5. Multiple views of a wall entity [Rivard 94].

design). A primitive should include a reference to all the entities to which it belongs. The fact that a given primitive may be referenced by several entities demonstrates the possibility for reusing the same data (e.g. the type of concrete should be defined once and referenced by every concrete member). Each primitive should have a built-in help mechanism that would allow the user to obtain a description of the data stored in it.A primitive can have both attributes and

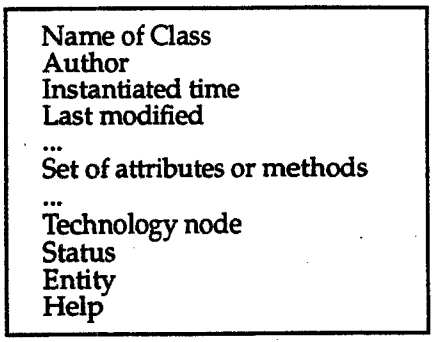

Figure 6. A generic primitive.

methods (or procedures). Methods are used to compute values based on other attribute-value pairs. For instance, the density of an object can be computed from its mass and volume. It is redundant to store the density data explicitly if it can be computed. The use of methods provide support for dependencies among data.

An entity contains a reference to the technology node that created it. As the entity is refined, design unit primitives are added to the entity. Figure 7 shows the relationship between an entity, its design unit primitives and a technology tree. This representation supports the generation of solutions in staged steps so as to allow backtracking and generating different states within the design space, representing alternative design solutions for the same subproblem. The hierarchical structure of the technology tree (and thus the knowledge base) serves to meet these goals [Fenves et al. 95]. The design process proceeds as follows: the child technology nodes of the current technology test the entity against their own constraints and determine whether it is applicable or not. If a technology node is applicable, it instantiates the appropriate design unit primitives and assigns corresponding design attributes. The designer selects one of the candidate primitives to

expand further with the technology tree. The selected primitive is automatically incorporated into the entity.

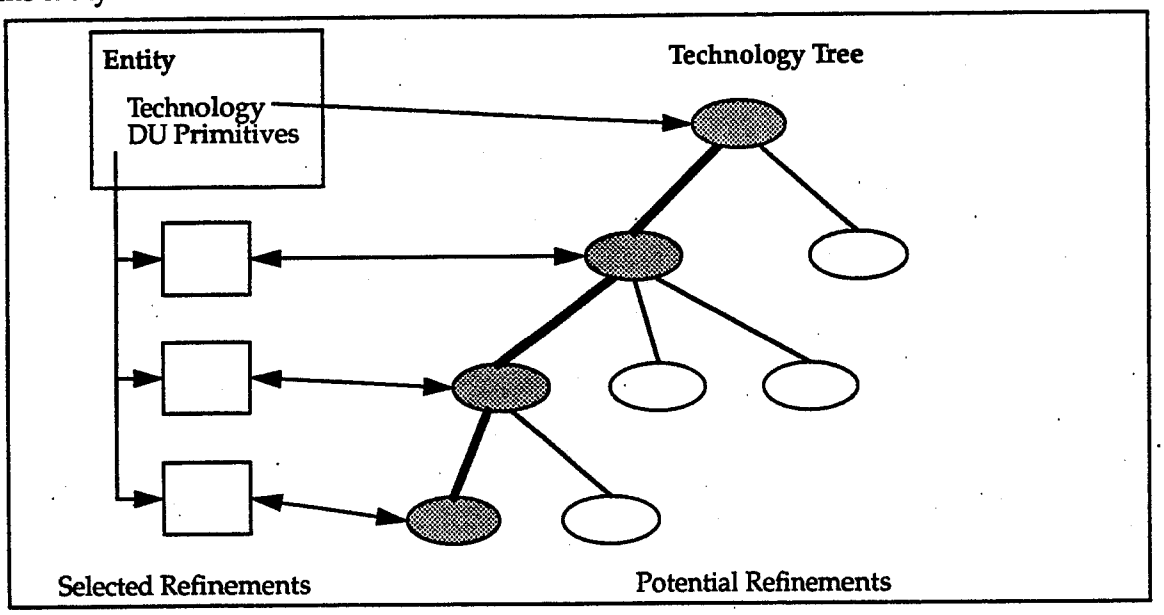

Figure 7. An entity , its design unit primitives and a technology tree

5. CASE-BASED REASONING SUPPORT

Case-based reasoning is an analogical reasoning method which uses previously stored solutions as a means to solve a new problem, to warn of possible failures, and to interpret a situation [Kolodner 1993]. A design system based on this artificial intelligence methodology would help the designer to remember previous and appropriate cases. The designer can use these cases as sources of inspiration, or as drafts on the basis of which a more relevant solution to the current problem can be developed. It is in the human nature to remember previous experiences in order to develop solutions for new problems. Designers use previous designs because they save time and effort and because the concept has been proven effective in a previous situation. Case-based reasoning is an attempt to implement this natural design process in computers as a tool for designers. The strength of computer programs augment the human abilities as follows:

- cases originating from different designers can be made available,
- cases are retrieved quickly and are not forgotten,
- a retrieved case can be used as a starting point to generate a new design, and
- the system learns as new cases are added to the case-base.

We have presented in this paper an information model that can be used both for case representation and for recording design data. Since the representation is identical for the two, no translation is required to store the design data into a case. This simplifies the implementation of the premise stated in Flemming's paper [Flemming 94] that cases are accumulated as a side-effect of a firm's normal design activities.

Hierarchical decomposition provides a mean of extracting cases at different level of details or abstraction. It also provides the capability for retrieving solutions to any level of detail [Flemming 1994]. Hence, a case can be retrieved at different levels: from the system level (e.g. the structure of a building wing) to the component level (e.g. a roof truss).

Cases are searched based on the current functional unit, the current hierarchical decomposition and the problem context. When an entity is retrieved, all its constituent sub-entities

are evaluated to see whether they also satisfy the new problem context. All the sub-entities that are satisfactory are retrieved to a depth specified by the user. The unsatisfactory sub-entities are pruned from the retrieved solution. Once a case is retrieved and approved by the user, it can be added to the current design. It can then be modified, augmented and reduced.

Sub-entities are pruned using the technology tree referenced by the retrieved entity (or case) and the current design context. If the design of the retrieved sub-entity is still within the range of applicability of the technology node once it is set in the new design context, its attributes are re-evaluated and matching continues at the successor level of the technology tree. If the design of the sub-entity falls outside the range of applicability, it is removed from the design state together with all its subsequent refinements and/or elaborations. The designer can then proceed to redesign the eliminated sub-entities. Designers have two options to replace the pruned sub-entities: they can execute a new case retrieval at the new, more detailed level; or they can complete the design by themselves or with the help of a technology tree.

The primitive representation, discussed in Section 4, supports the retrieval of cases (or entities) with multiple functions (or views). Multifunction entities are frequent in building design and must be supported by a case-based reasoning system. Whenever a case is retrieved for a particular function and it is found to have more than one function, the designer has the choice to keep those extra functions or to strip them from the case. Furthermore, searches for multi-functional entities are supported. Hence, a designer is able to retrieve an entity that complies with two or more functions. Here are a few illustrating examples:

- a case retrieved for an enclosure design problem may also be found to satisfy the structural requirements of that entity;
- the case base may be searched for a case satisfying the requirements of more than one view (e.g. the enclosure and the structural views of an exterior wall);
- a multi-function entity may be stripped of one of its design aspects if it is deemed useless (e.g. one may remove the enclosure aspect of an exterior load-bearing wall case to be used indoors).

The recording of a reference to a node of the technology tree in a case provides several advantages. It records both the results of the design as well as the design process itself. By referring to the technology nodes that were used in designing an entity, we are also recording a reference to the knowledge and process used in designing it. This is as important to the designer as the design descriptions. It allows the designer to reuse the same design process. It is also possible to limit the search of a case to a given technology (or one of its children) or to exclude a given technology from the search.

6. CONCLUSION

In this paper, we presented an information model for the preliminary design stage. This model decomposes buildings into hierarchies of entities which provide different levels of abstraction. Each building entity contains:

- attribute-value pairs organized into three subsets: function aspect, design aspect and behavior aspect;
- a high-level geometrical description which is used to reason about its topological relations with other entities, and discipline specific geometric information;
- relationships with other entities; and
- references to the computational mechanisms (or technologies) used in designing it.

The attribute-value pairs of an entity are further grouped into primitives in order to integrate multiple views, to facilitate data exchange between design tasks, to improve communication between designers and to support the growth of data as the design process unfolds.

We believe that this information model has the potential to support case-based reasoning and to record design data as it is generated during the design process. We intend to implement this information model in an object-oriented database management system. Subsequent validation with end users will show to what extent the approach is appropriate in parts or in whole.

7. REFERENCES

Coyne, R. D., M. A. Rosenman, A. D. Radford, M. Balachandran, and J. S. Gero (1990). *Knowledge-Based Design Systems*, Addison-Wesley Publishing Co., Reading, MA.

Date, C. J. (1995). *An Introduction to Database Systems*, Sixth Edition, Addison-Wesley Publishing Co., Reading, MA.

Fenves, S. J., H. Rivard, N. Gomez, and S.-C. Chiou (1995). "Conceptual Structural Design in SEED." To be published in the Journal of Architectural Engineering of ASCE in the fall of 1995.

Flemming, U. (1994). "Case-Based Design in the SEED System." *Knowledge-Based Computer-Aided Architectural Design*, G. Carrara and Y. Kalay (Editors), Elsevier, New-York, NY, pp. 69-91.

Flemming, U., R. Coyne and R. Woodbury (1993). "SEED: A Software Environment to Support the Early Phases in Building Design" in ARECDAO93, Proceedings of the IVth Int. Conference on Computer Aided Design in Architecture and Civil Engineering, Barcelona, Spain, pp. 111-122.

Flemming, U. and R. Woodbury (1995). "A Software Environment to Support the Early Phases in Building Design (SEED): An Overview." To be published in the Journal of Architectural Engineering of ASCE in the fall of 1995.

Gielingh, W. (1988). "General AEC Reference Model." ISO TC 184/SC4/WG1 doc 3.2.2.1, TNO Report BI-88-150.

Howard, H. C., J. A. Abdalla and D. H. D. Phan (1992). "Primitive-Composite Approach for Structural Data Modeling", Journal of Computing in Civil Eng., ASCE, Vol. 6, No. 1, pp. 19-40.

Kolodner, J. L. (1993). *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

Lin, T. Y. and S. D. Stotesbury (1981). *Structural Concepts and Systems for Architects and Engineers*, John Wiley & Sons Inc., New-York, NY.

Phan, D. H. D. and H. C. Howard (1993). "The Primitive-Composite (P-C) Approach - A Methodology for Developing Sharable Object-Oriented Data Representations for Facility Engineering Integration", Technical Report 85 (A and B), CIFE, Stanford University.

Rivard, H. (1994). "Integration of the Building Envelope Design Process", Master Thesis, Centre for Building Studies, Concordia University, Montreal, Canada.

Rivard, H., C. Bedard, K. H. Ha and P. Fazio (1995). "An Information Model for the Building Envelope", ASCE, Proceedings of the 2nd Congress of Computing in Civil Engineering, Atlanta, GA, June 5-8, pp. 302-309.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen (1991). *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ.

Rush, R. D., Editor (1986). *The Building Systems Integration Handbook*, The American Institute of Architects, Butterworth-Heinemann.

Woodbury, R. and S. J. Fenves (1994). "SEED-Config Requirements Analysis", Internal Document, Carnegie Mellon University.

Zamanian, K. M. (1992). "Modeling and Communicating Spatial and Functional Information about Constructed Facilities." Phd thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.