1995

An information model for the preliminary design of buildings

Hugues Rivard
Carnegie Mellon University

Steven J.(Steven Joseph) Fenves

Nestor Gomez

Carnegie Mellon University.Engineering Design Research Center.

Published In
.

An Information Model for the Preliminary Design of Buildings

Hugues Rivard, Steven J. Fenves and Nestor Gomez

EDRC 12-71-95

An Information Model for the Preliminary Design of Buildings

Hugues Rivard, Research Assistant,

Steven J. Fenves, Sun Company University Professor, and

Nestor Gomez, Research Assistant.

Department of Civil and Environmental Engineering
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA, 15213

November 14,1995

Abstract

The report outlines an information model that organizes the wealth of data used and generated during the conceptual design stage of buildings. 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 containment relationships (made-of and part-of), a reference to the technology (knowledge and procedures) that is used to derive it, and a set of classifiers. The geometry and topological relationships for the entity are obtained from a non-manifold skeletal geometrical representation common across all views. Representation of multiple views is supported by dividing the attributes of an entity into small cohesive subsets, which we call components. These components are then used as construction blocks to present different views of the entity. The goal of this representation is twofold: to store the design data as it is generated during the conceptual design and to support case-based reasoning.

Contents

List of Figures

Chapter 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 Design) currently under development at Carnegie Mellon University [Hemming et al. 931. This module supports the generation of a 3-dimensional configuration of spatial and physical building components based on schematic layouts [Hemming and Woodbury 951. 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 designers to retrieve and adapt previous designs as an aid in solving the current design problem.

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 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 hill life cycle of the building and would have a positive impact on productivity, costs and quality.

The implementation environment envisioned for this information model is an object-oriented database management system. Object-oriented database management systems combine the richness of representation of object-oriented languages with the practical features of database management systems and have a good potential for modelling complex applications. This technology is chosen because of its ability to model and manage complex data types, its capacity to model a problem domain more naturally and its suitability for implementing systems that are more closely related to the model [Rivard 94].

The information model presented here addresses the early design stages and supports design evolution. This contrast with the efforts for developing a Standard for the Exchange of Product Model Data (STEP) which addresses a later stage of design and provides snapshots of the evolving product but does not support design evolution (for a general overview of STEP see [Burkett and Yang 951; for an overview of current European A/E/C standardization works see [Wix and Bloomfield 95]).

Chapter 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.

An entity is an object that is to be represented in the database [Date 95]. Information about 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. An entity includes the six following categories of information: functional unit components; design unit components; evaluation unit components; relationships; technologies; and classifiers. Figure 1 shows the six categories of information and their subcategories. The six categories are described in greater detail in the following sub-sections.

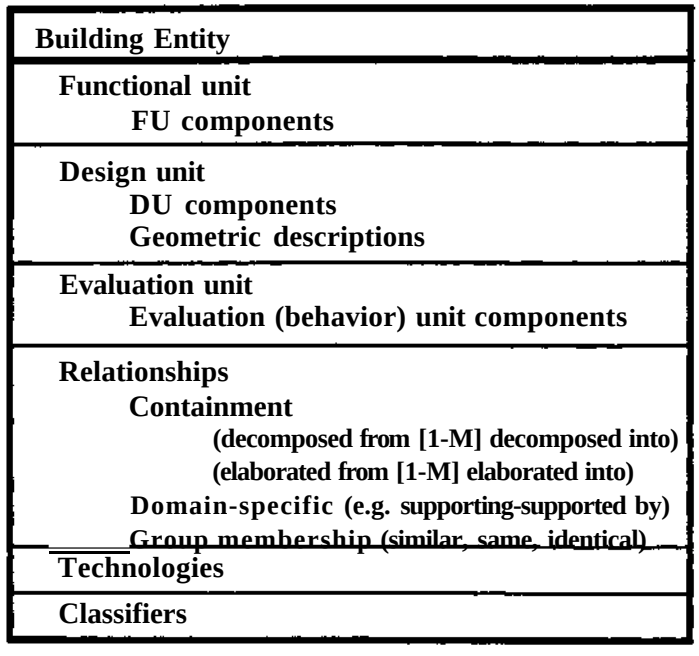

Figure 1. A building entity description.

In order to address SEED-Config's data requirements, the underlying data model for the SEED project, which consists of a functional unit - design unit - technology triad [Flemming and Woodbury 95], is extended as follows:

- the evaluation unit is added to the triad in order to record the results of the frequent evaluations required;
- the relationships are extracted from the design unit and functional unit and are added as a separate category for ease of access;

- classifiers are added to allow the classification of the entities without the need for inheritance based classification; and
- the object's information is encapsulated by a container called an entity.

2.1 Functional Unit, Design Unit and Evaluation Unit

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. 901. 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 (e.g., integer, float, string and boolean), 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 data (i.e. 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 881. 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 a 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 evaluation unit.

The cardinality of the three units for a given entity is as follows:

- 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 evaluation units per design unit, corresponding to the 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. Thus, the reason which a particular design was selected can be understood and justified from its function, behavior and instantiating technologies. This represents a richer data model than the ones supported by current CAD systems which store only the design results.

1.2 Geometrical Descriptions

We are following the approach developed by Zamanian for the geometrical description of entities [Zamanian 921. 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; while secondary representations of an entity are its discipline-specific geometric representations. Each entity has only one primary spatial representation, but it may have several secondary representations. This representation scheme relies on the premise that topological relations of physical entities are invariant with respect to their discipline-specific secondary representations.

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.

A 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.

Since the geometric descriptions are part of the design definition, their logical position in the entity is in the design unit category.

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 containment relationships from others to ensure better access. The containment relationship (also known as aggregation relationship) is very important in describing buildings because it captures the link between an entity and its components. Building entities are usually complex objects built from component entities which may be complex themselves. The complex object is treated as a unit in many operations, although physically it is made of several component objects. The containment 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 inverse link between the complex object and the component. Other relationships are frequently needed as well. For instance, the supporting-supported by

relationship is used to define a load path among the structural entities. Topological relationships, such as next to, above, spatially contained in and adjacent, 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 design knowledge into a hierarchically structured technology tree. Then, the design of an entity can be simply described by referring to the technology node that created it.

A technology 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 941. Technologies are organized in the form of a tree. In the structural design domain, the technology tree represents the various alternative structural systems, subsystems 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. These constraints may arise from: functional requirements obtained from the functional unit; geometric limitations obtained from the geometric description; or from relationships of the entity to other entities. 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 contained entities. Hence, a technology node refines another if it provides additional level of detail by adding new components to the design unit of the current entity, or a technology node 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 containment relationship slot [Fenves et al. 951.

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 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 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.

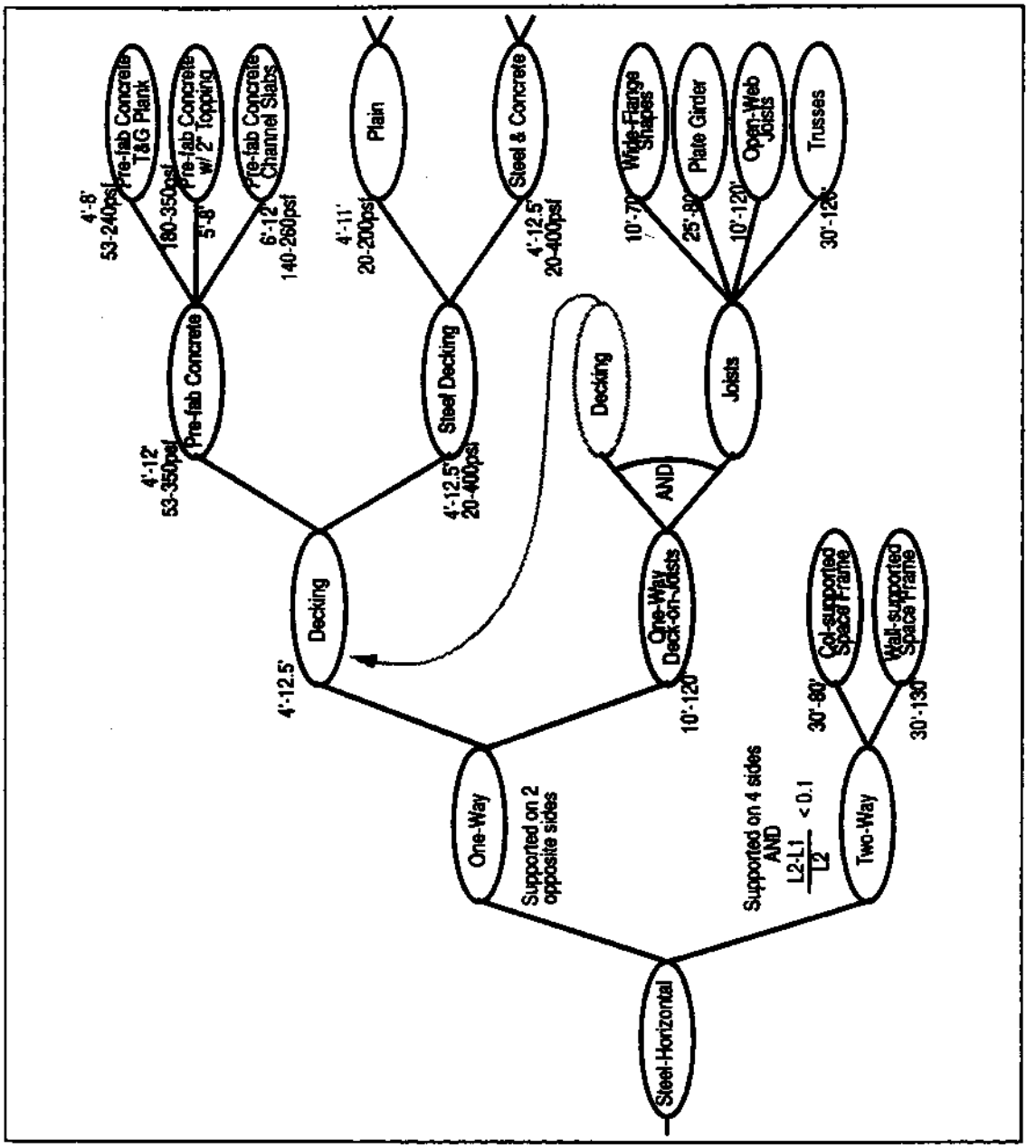

Figure 2. A sample partial structural technology tree [Fenves et al. 951

There are three categories of technologies: **decomposition technologies** are used to break down a complex design problems into sub-problems; **elaboration technologies** are used to subdivide the current entity into constituent sub-entities; and **refinement technologies** are used to provide additional detail to an entity. The first two categories are implemented in the same manner with respect to the information model. Both categories create new entities and link them to the current entity through the containment relationship slot. Refining technologies add new components to the design unit of the current entity and can operate on both abstract or detailed

entities. Figure 3 shows the relationship between an entity and the three types of technologies and Figure 4 illustrates the creation of a hierarchy of entities using the three types of technologies.

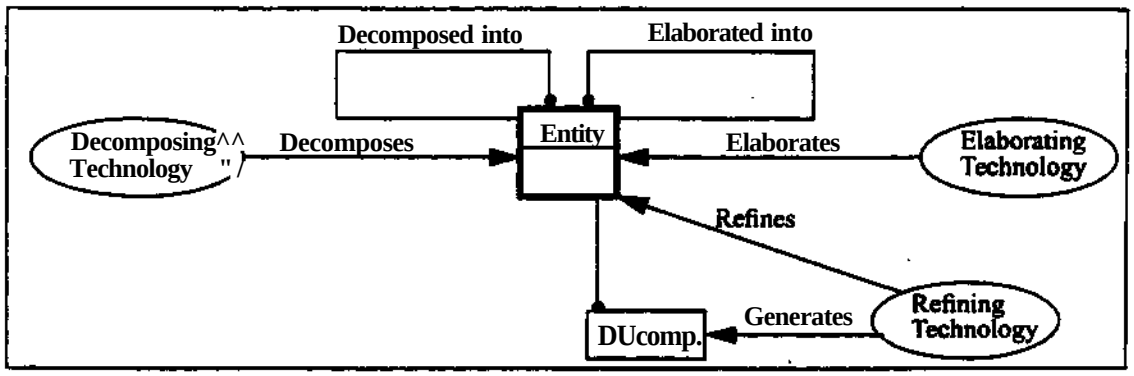

Figure 3. Object model of the relationship between entity and technologies.

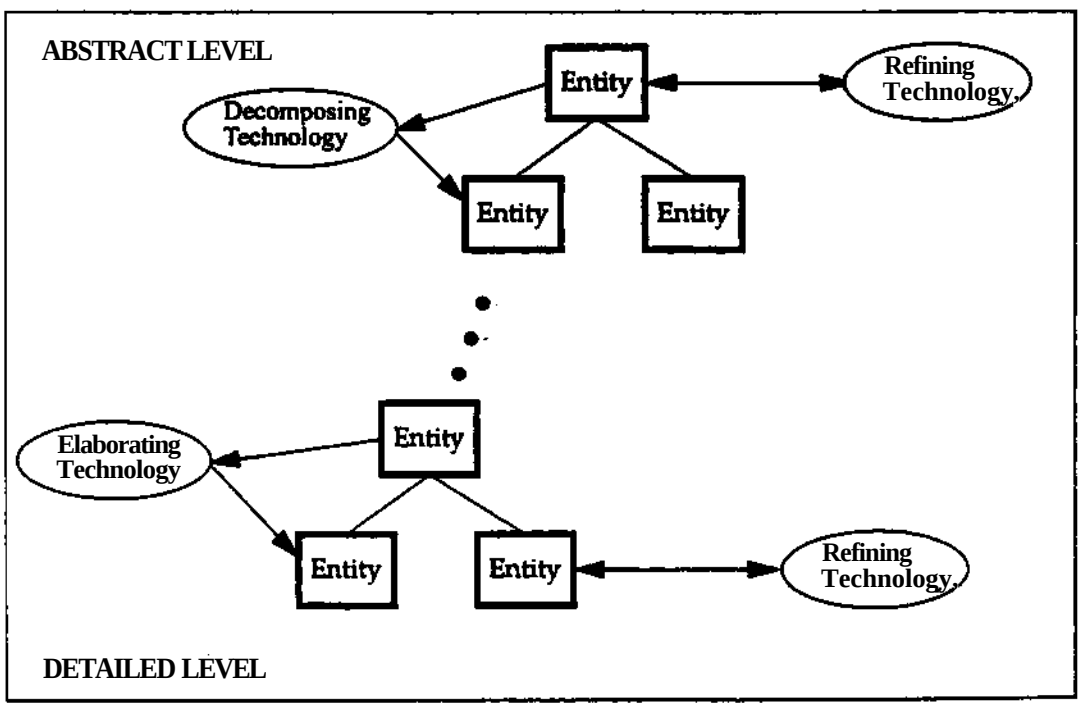

Figure 4. Use of technologies in defining a hierarchy of entities.

2.5 Classifiers

Classification is the process of systematically arranging entities into groups [Harris and Wright 81]. Classification is an important aspect of problem-solving tasks [Rich and Knight 91]. It can be used to identify a given entity or to query the set of all entities for a given group. Building

entities are classified through the use of classifiers. A classifier is an instance of a class which can be part of a class hierarchy. Hence, a class can be the subclass of a more general class. An entity is not limited to one classifier, but may have several classifiers. This approach for classification is similar to the one used in the Context-Oriented Model [Kiliccote 92]. It circumvents many anomalies and ambiguities that arise from the use of multiple inheritance in object-oriented programming [Flemming et al. 951. Figure 5 shows an example of an entity being classified as a framed tube. The fact that the entity's classifier is an instance of the class "Framed Tube" means that the entity can be generally classified as a framing system, specifically classified as a tube framing system, or even more specifically as a framed tube framing system.

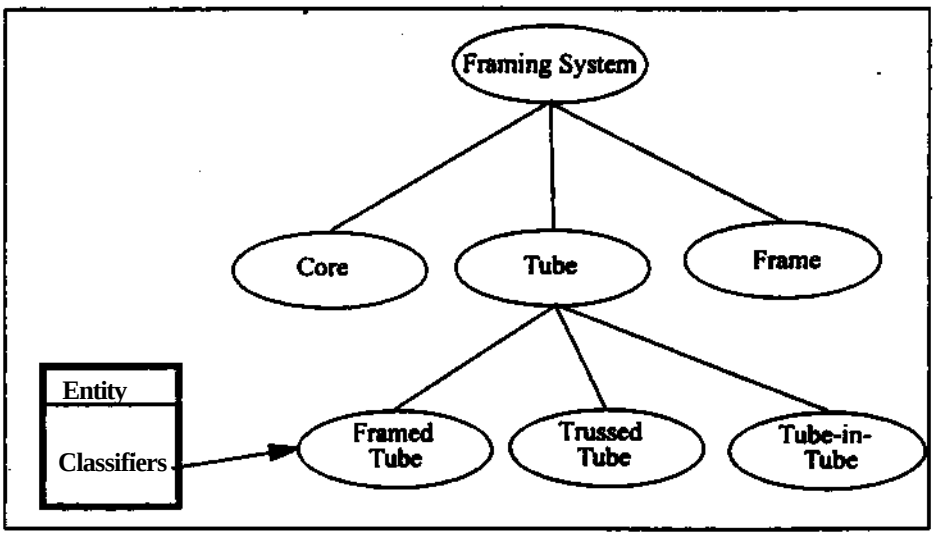

Figure 5. Classifiers and class hierarchy.

Chapter 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, decomposed such that a solution can be easily determined. This "divide-and-conquer" strategy is typical for most design processes. The information model must therefore be able to support the decomposition of design problems. Such hierarchical decompositions are also used to distribute tasks and responsibilities among designers [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 6 shows the hierarchical decomposition of the building into four systems with the structural system further decomposed. The structure of a building may sometimes be broken down into independent sub-systems depending on the building's complexity. These sub-systems, which we call 3-D structural systems, occur in buildings with expansion joints or in buildings made of several independent structural systems. The 3-D structural systems are 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 (e.g., beams, columns and slab elements). Note that the figure does not show all the relationships which may exist among the entities shown (e.g., supports and connected to). 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.

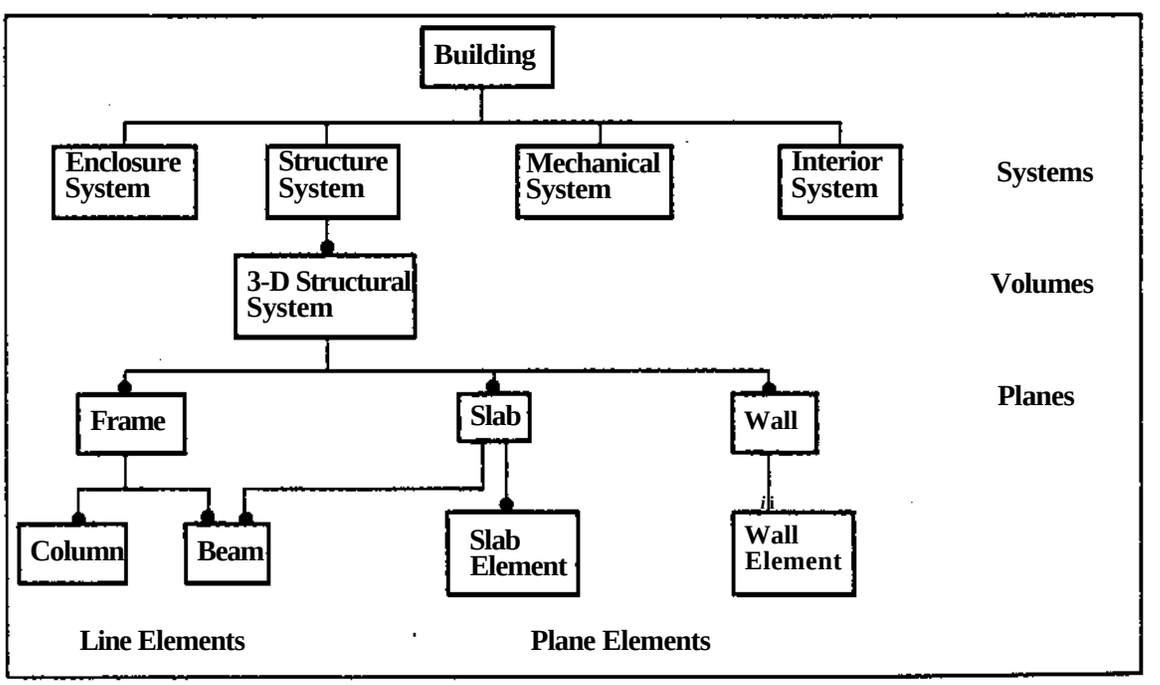

Figure 6. Hierarchical decomposition of the building structure.

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 811. 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 and is shown in Figure 7. 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 layers (i.e. construction products) such as cladding, membrane, insulation and finishing [Rivard et al. 95].

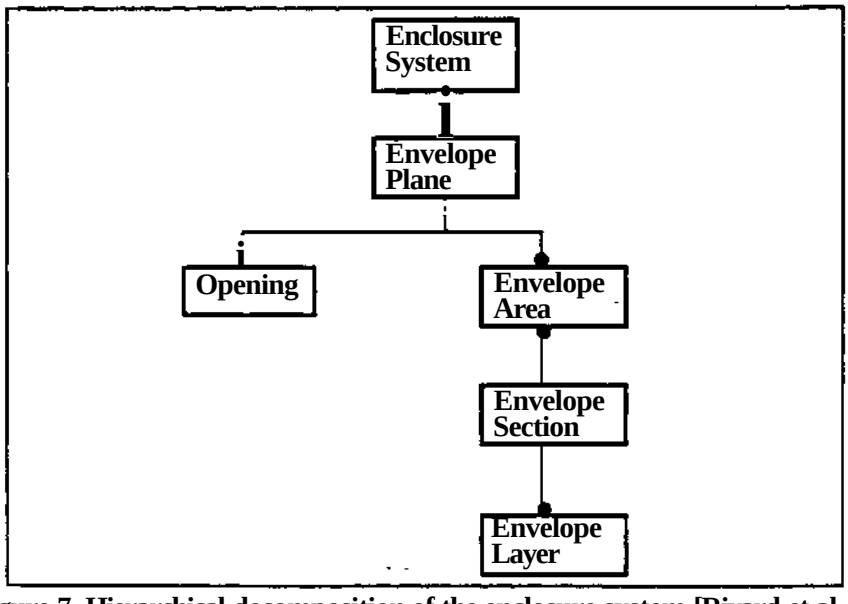

Figure 7. Hierarchical decomposition of the enclosure system [Rivard et al. 951

The information model presented here supports such hierarchical decompositions. Systems and subsystems can be modeled as entities linked by containment 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.

The information model also supports the design process as it unfolds by allowing designers to populate the design space in an intuitive manner from global system entities through sub-system entities ail 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, created from elaborations of more abstract entities, are added to the existing ones, through containment relationship, as the design proceeds and becomes more and more detailed.

It is important to note that the hierarchical decompositions described in this section are examples only. It does not mean that the hierarchical decomposition for the structure is fixed and unchangeable. The power of this information model is that any hierarchical decomposition can be supported. Hence, additional entities could be incorporated between the ones shown in the examples, and different alternative design solutions could incorporate different hierarchical decompositions.

Chapter 4

Integration of Multiple Views

The entities presented in the previous section represent a logical manner for organizing the building data and correspond to the way designers see the building. It may often 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 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.

4.1 Component Representation

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 subsets; 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 attribute values are accessed by attribute names. 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 subsets, each of which we call a component. This approach, called the Primitive-Composite Approach (or P-C Approach), was originally suggested 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 advantage that it supports multiple views, schema evolution and data integration.

Cohesion is the only criterion used in decomposing entities. It is defined as a measure 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 component is slightly different. We keep only three of the five original criteria. Hence, a component 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 (e.g., 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 component exists, he or she can assume that all the attributes in it have a value. If the component does not exist, it means that the data has not been generated yet, and he or she may create a new component. This criterion implies that each component 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 overly subdivided. It does not really matter if all data attributes of a component 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 component. Methods could be used from within the component to compute the dependent information.

The component representation provides an abstraction of the entity to designers. Views hide the actual complexity of the entity by providing only **the** relevant information while hiding the unnecessary details. Figure 8 shows a wall entity decomposed into a set of components. Three different views are shown as referring to a subset of the components. Two of the components 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.

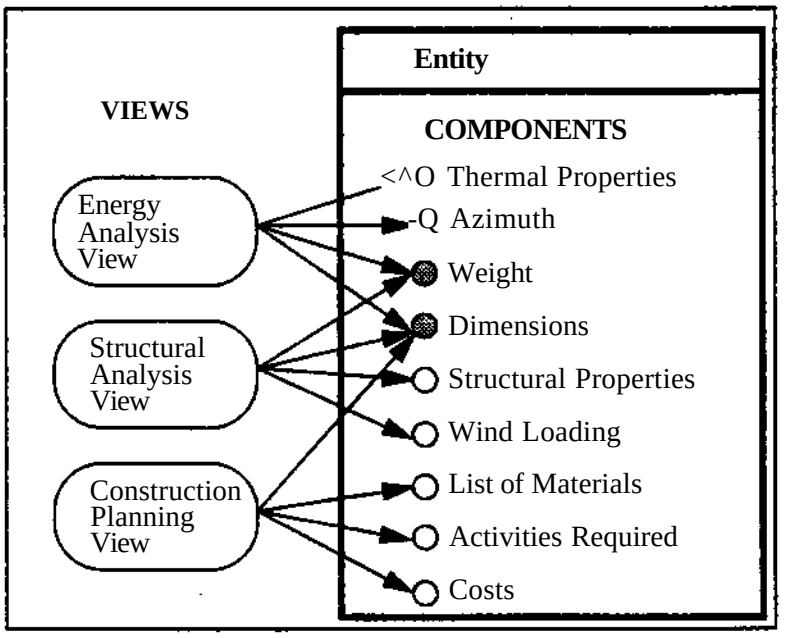

Figure 8. Multiple views of a wall entity [Rivard 941.

4.2 A Generic Component Class

The definitions of the components are inherited from a common superclass, which is shown in Figure 9. A component class should have a unique name. Each component 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 design). A component should include a reference to all the entities to which it belongs. The fact that a given component 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 component should have a built-in help mechanism that would allow the user to obtain a description of the data stored in it A component can have both attributes and methods (or procedures). Methods are used to compute values based on other components. The use of methods provide support for dependencies among data.

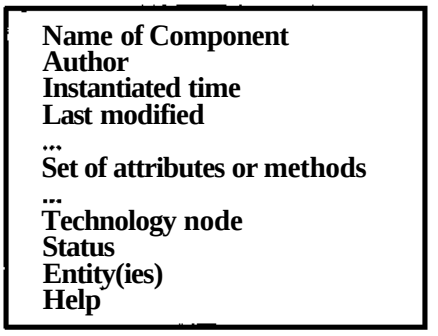

Figure 9. A generic component class.

An advantage of encapsulating data within component objects is that the type of data stored could be of multimedia type. The component has the appropriate methods to display, edit and input its data. Therefore, a component could contain images, sounds, texts and even video. For instance, an evaluation unit component could contain the bending moment diagram of a structural beam.

Concurrent use has not been considered at this stage. Since changes are usually localized to the component levels, only the components would need to be locked. The system could allow the display of components created by other users but protect them from modification if a user does not have write permission.

4.3 Relationship Between Components, Entities and Technologies

An entity contains a reference to the technology node that created it. As the entity is refined, design unit components are added to the entity. Figure 10 shows the relationship between an entity, its design unit components 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. 951. The design process proceeds as follows: the child technology nodes of the current technology test the entity against their own constraints and determine whether they are applicable or not. If a technology node is applicable, it instantiates the appropriate design unit component(s) and assigns corresponding design attributes. The designer selects one of the candidate components to expand further with the technology tree. The selected component is automatically incorporated into the entity.

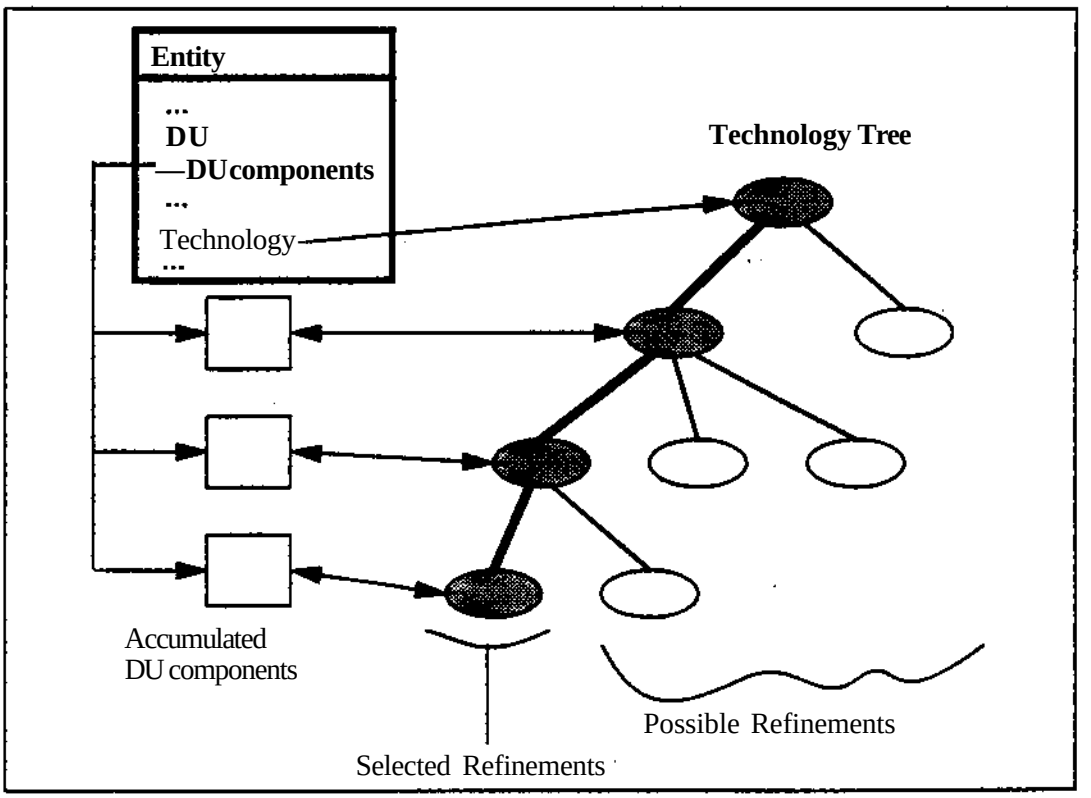

Figure 10. An entity , its design unit components and a technology tree

4.4 Integration Across Hierarchical Decompositions

The primary spatial representation of a building entity, described in section 2.2, can be used to identify an entity in the geometrical model and then to add appropriate components to the entity. For instance, a structural function may be enforced to a building envelope entity (such as an exterior wall), found in the geometrical model, by adding the proper relationships, FU components and DU components. The designer could access the entity through the geometric model and add it to the structure hierarchical decomposition. Hence, the exterior wall becomes part of two hierarchical decompositions: the enclosure and the structure. The decomposition hierarchies become lattices.

4.5 Search Mechanism for Components Within Building Entities

The building entity's information is encapsulated within components. The information is accessed through a search mechanism. First, the requester (which may be a user or a computer application) must specify one of the subsets where the search is to be made: either functional unit, design unit or evaluation unit. Second, the requester must provide the name of the component that is of interest. For example, say someone is interested in checking the water-ratio of a concrete structural element, s/he would access the corresponding building entity and provide the following request: design-unit (concrete characteristics). The system would look among the components stored in the design-unit and return the one with the matching name. Once the requester has found access to the component, s/he can fire any of its methods such as display or edit attributes. In our example, the person would request the display of the water-ratio attribute.

The building entities have a mechanism to display all the attributes of their functional units, design units and evaluation units for quick reference. Whenever someone requests that facility for one of the subsets of data, the building entity displays all of its attributes and their values under headings corresponding to their component names.

Chapter 5

Grouping Entities

To facilitate the design process, a set of entities can be grouped together and designed simultaneously. We have identified three different types of groups: "same" means that all the entities are assigned the same design components according to the most constraining entity of the group; "similar" means that all the entities are designed with the same technology but may have design components with different attribute values; and "identical" means that all the entities in the group have the same dimensions and are subject to the same conditions and hence can be designed just by looking at one of the entities. As for "same", the entities grouped by "identical" are assigned the same design components. While the two groups "same" and "similar" are assigned by the user, the group "identical" is assigned by default by elaborating technologies.

As an example of the use of a "same" group, a structural engineer may want to assign the same concrete characteristics to a group of entities. The engineer selects the group of entities and the refining technology that assigns the concrete characteristics; the technology would check that the structural material of each entity is concrete and would assign the specified concrete by adding the same DU component to all entities. Having only one DU component for a group of entities ensures that any changes will be applied to all entities (i.e. there is no unnecessary redundancies). This example is illustrated in Figure 11 below. For a group of "similar" entities, the technology would add a new DU component to each entity.

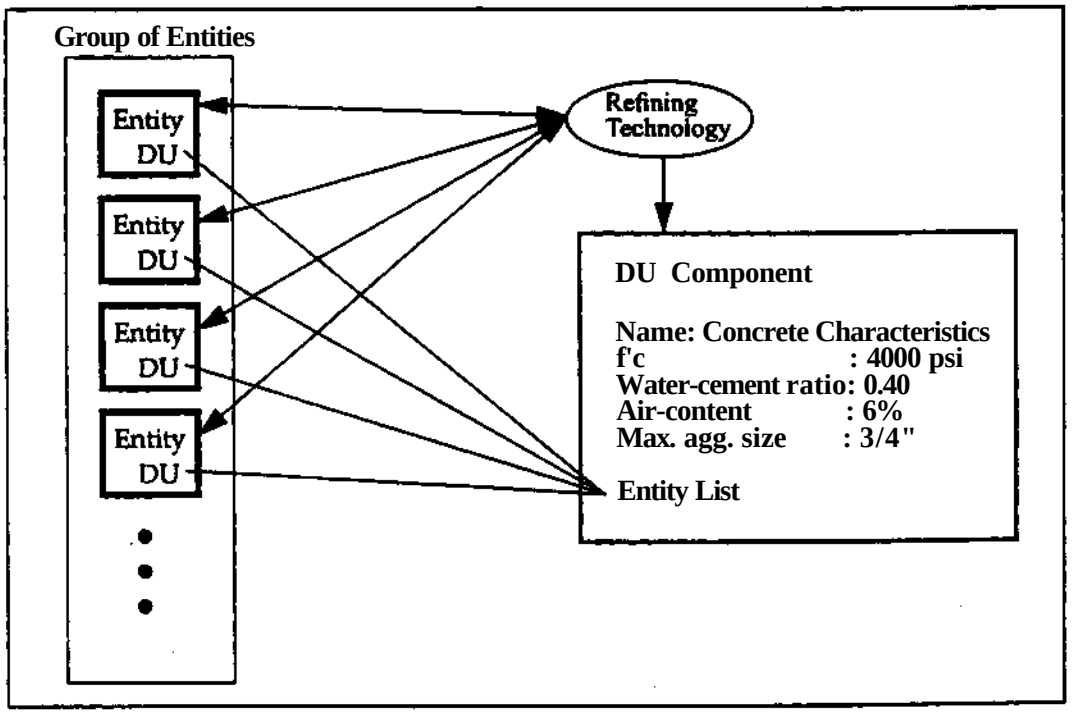

Figure 11. Designing a group of "same" entities.

As an example of an "identical" group, when elaborating a structural slabs into beams and deckings, the technology creates a number of beam-entities and a number of deck-entities. All these entities are associated to the slab-entity through the containment relationship. Each distinct sub-entity is needed because each one has its own distinct primary spatial representation. But to simplify design, all the beam-entities are automatically grouped into an "identical" group and thus ensure consistency between the beams. When a technology refines the beam-entities, it assigns the same DU components to all of the entities in the group. The same process applies for the deck-entities. Figure 12 below illustrates the creation and design of the beam entities.

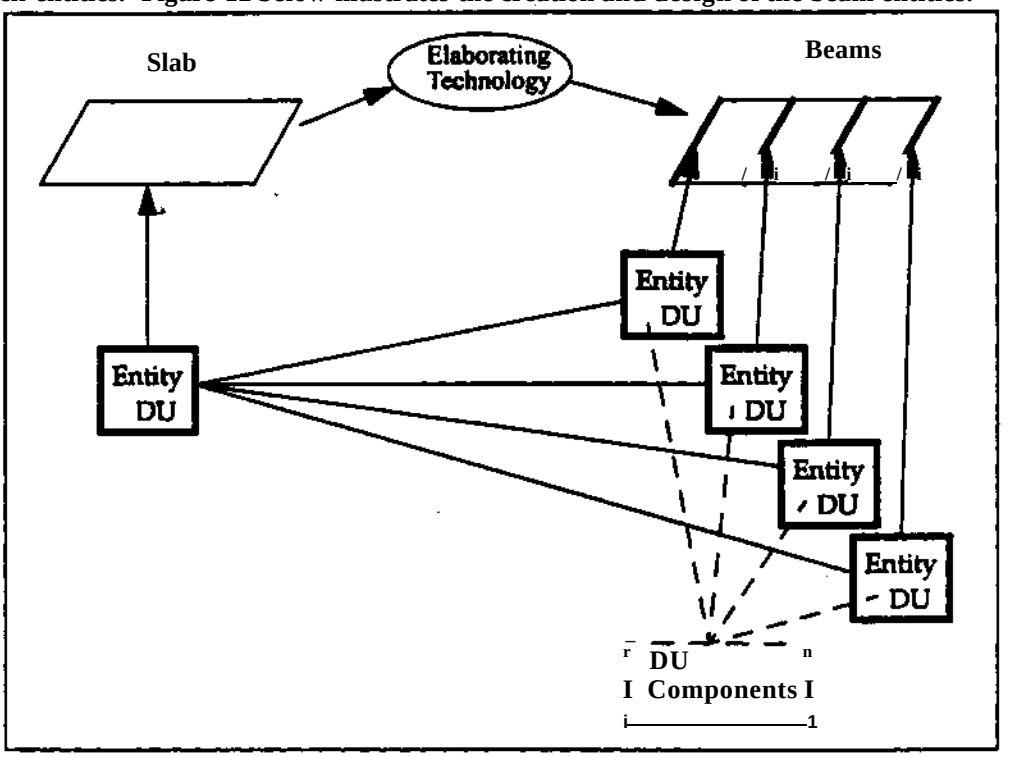

Figure 12. The creation of an "identical" group of entities.

Chapter 6

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 ieams as new cases are added to the cast

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 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 94]. 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 component 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.

Flemming's case retrieval mechanism for a hierarchy of objects would work with this information model [Flemming 94]. The only differences are that instead of traversing a hierarchy of functional units, we are traversing a hierarchy of building entities; and that an additional parameter is necessary when searching for an attribute: the name of the component which contains the attribute of interest.

The fact that a component refers back to the entities it belongs to allows a case search to be done from bottom up. All the components that satisfies a search criteria could be used as a basis to find matching cases. The name of the components could be used as an index to limit the search to a particular type of component. Once cases (or building entities) are retrieved through the matching components, they can be pruned based on other search requirements. The classifiers are another mean for retrieving cases. They support case retrieval from general to specialized entities.

Chapter 7

Conclusion

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

- data 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;
- references to the computational mechanisms (or technologies) used in designing it; and
- a set of classifiers.

The data of an entity are further grouped into components 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 record design data as it is generated during the design process and to support case-based reasoning. 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.

References

Burkett, W. C. and Y. Yang (1995). "The STEP Integration Information Architecture." Engineering with Computers, Springer, Vol. 11, No. 3, pp. 136-144.

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.

Flemming, U., Z. Aygen, R. Coyne and J. Snyder (1995). "Case-Based Design in a Software Environment that Supports the Early Phases in Building Design." To be published in "Issues and Applications of Case-Based Reasoning to Design", Maher, M. L. and Pu, P. (eds), Lawrence Erlbaum Associates.

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

Harris, J. R., and R. N. Wright (1981). "Organization of building standards: systematic techniques for scope and arrangement." Washington: U.S. Government Printing Office.

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.

Kiliccote, H. (1992). "The Context-Oriented Model: A Hybrid Approach to Modeling and Processing Design Standards." M. Sc. Thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.

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.

Rich, E. and K. Knight (1991). "Artificial Intelligence." 2nd Edition, McGraw Hill, New-York.

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.**

Wix, J. and D. P. Bloomfield (1995). "Standardisation in the Building Industry: The STEP Building Construction Core Model." Publication 180, International Council for Building Research Studies and Documentation (CIB), Stanford, California, pp. 184-195.

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 Infonnation about Constructed Facilities." Phd thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.

An Information Model for the Preliminary Design of Buildings

Hugues Rivard, Research Assistant,

Steven J. Fenves, Sun Company University Professor, and

Nestor Gomez, Research Assistant.

Department of Civil and Environmental Engineering
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA, 15213

November 14,1995

Abstract

The report outlines an information model that organizes the wealth of data used and generated during the conceptual design stage of buildings. 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 containment relationships (made-of and part-of), a reference to the technology (knowledge and procedures) that is used to derive it, and a set of classifiers. The geometry and topological relationships for the entity are obtained from a non-manifold skeletal geometrical representation common across all views. Representation of multiple views is supported by dividing the attributes of an entity into small cohesive subsets, which we call components. These components are then used as construction blocks to present different views of the entity. The goal of this representation is twofold: to store the design data as it is generated during the conceptual design and to support case-based reasoning.

Contents

List of Figures

Chapter 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 Design) currently under development at Carnegie Mellon University [Hemming et al. 931. This module supports the generation of a 3-dimensional configuration of spatial and physical building components based on schematic layouts [Hemming and Woodbury 951. 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 designers to retrieve and adapt previous designs as an aid in solving the current design problem.

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 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 hill life cycle of the building and would have a positive impact on productivity, costs and quality.

The implementation environment envisioned for this information model is an object-oriented database management system. Object-oriented database management systems combine the richness of representation of object-oriented languages with the practical features of database management systems and have a good potential for modelling complex applications. This technology is chosen because of its ability to model and manage complex data types, its capacity to model a problem domain more naturally and its suitability for implementing systems that are more closely related to the model [Rivard 94].

The information model presented here addresses the early design stages and supports design evolution. This contrast with the efforts for developing a Standard for the Exchange of Product Model Data (STEP) which addresses a later stage of design and provides snapshots of the evolving product but does not support design evolution (for a general overview of STEP see [Burkett and Yang 951; for an overview of current European A/E/C standardization works see [Wix and Bloomfield 95]).

Chapter 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.

An entity is an object that is to be represented in the database [Date 95]. Information about 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. An entity includes the six following categories of information: functional unit components; design unit components; evaluation unit components; relationships; technologies; and classifiers. Figure 1 shows the six categories of information and their subcategories. The six categories are described in greater detail in the following sub-sections.

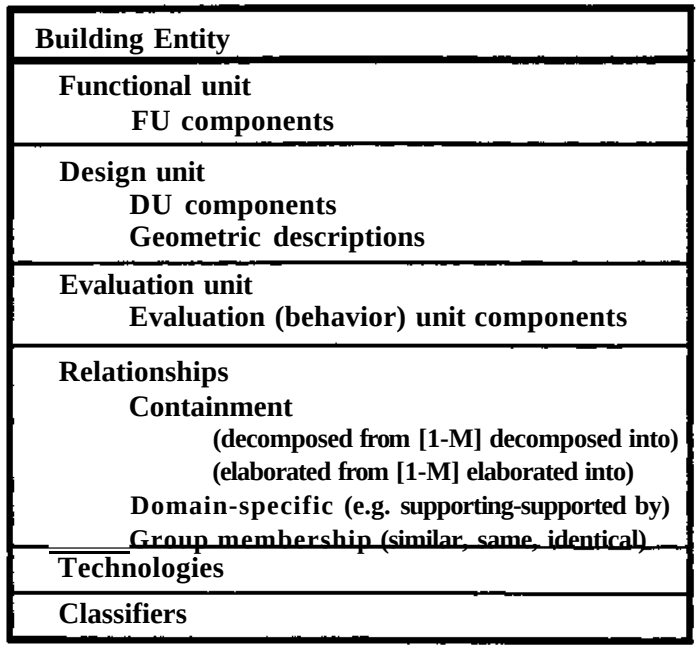

Figure 1. A building entity description.

In order to address SEED-Config's data requirements, the underlying data model for the SEED project, which consists of a functional unit - design unit - technology triad [Flemming and Woodbury 95], is extended as follows:

- the evaluation unit is added to the triad in order to record the results of the frequent evaluations required;
- the relationships are extracted from the design unit and functional unit and are added as a separate category for ease of access;

- classifiers are added to allow the classification of the entities without the need for inheritance based classification; and
- the object's information is encapsulated by a container called an entity.

2.1 Functional Unit, Design Unit and Evaluation Unit

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. 901. 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 (e.g., integer, float, string and boolean), 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 data (i.e. 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 881. 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 a 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 evaluation unit.

The cardinality of the three units for a given entity is as follows:

- 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 evaluation units per design unit, corresponding to the 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. Thus, the reason which a particular design was selected can be understood and justified from its function, behavior and instantiating technologies. This represents a richer data model than the ones supported by current CAD systems which store only the design results.

1.2 Geometrical Descriptions

We are following the approach developed by Zamanian for the geometrical description of entities [Zamanian 921. 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; while secondary representations of an entity are its discipline-specific geometric representations. Each entity has only one primary spatial representation, but it may have several secondary representations. This representation scheme relies on the premise that topological relations of physical entities are invariant with respect to their discipline-specific secondary representations.

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.

A 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.

Since the geometric descriptions are part of the design definition, their logical position in the entity is in the design unit category.

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 containment relationships from others to ensure better access. The containment relationship (also known as aggregation relationship) is very important in describing buildings because it captures the link between an entity and its components. Building entities are usually complex objects built from component entities which may be complex themselves. The complex object is treated as a unit in many operations, although physically it is made of several component objects. The containment 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 inverse link between the complex object and the component. Other relationships are frequently needed as well. For instance, the supporting-supported by

relationship is used to define a load path among the structural entities. Topological relationships, such as next to, above, spatially contained in and adjacent, 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 design knowledge into a hierarchically structured technology tree. Then, the design of an entity can be simply described by referring to the technology node that created it.

A technology 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 941. Technologies are organized in the form of a tree. In the structural design domain, the technology tree represents the various alternative structural systems, subsystems 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. These constraints may arise from: functional requirements obtained from the functional unit; geometric limitations obtained from the geometric description; or from relationships of the entity to other entities. 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 contained entities. Hence, a technology node refines another if it provides additional level of detail by adding new components to the design unit of the current entity, or a technology node 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 containment relationship slot [Fenves et al. 951.

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 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 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.

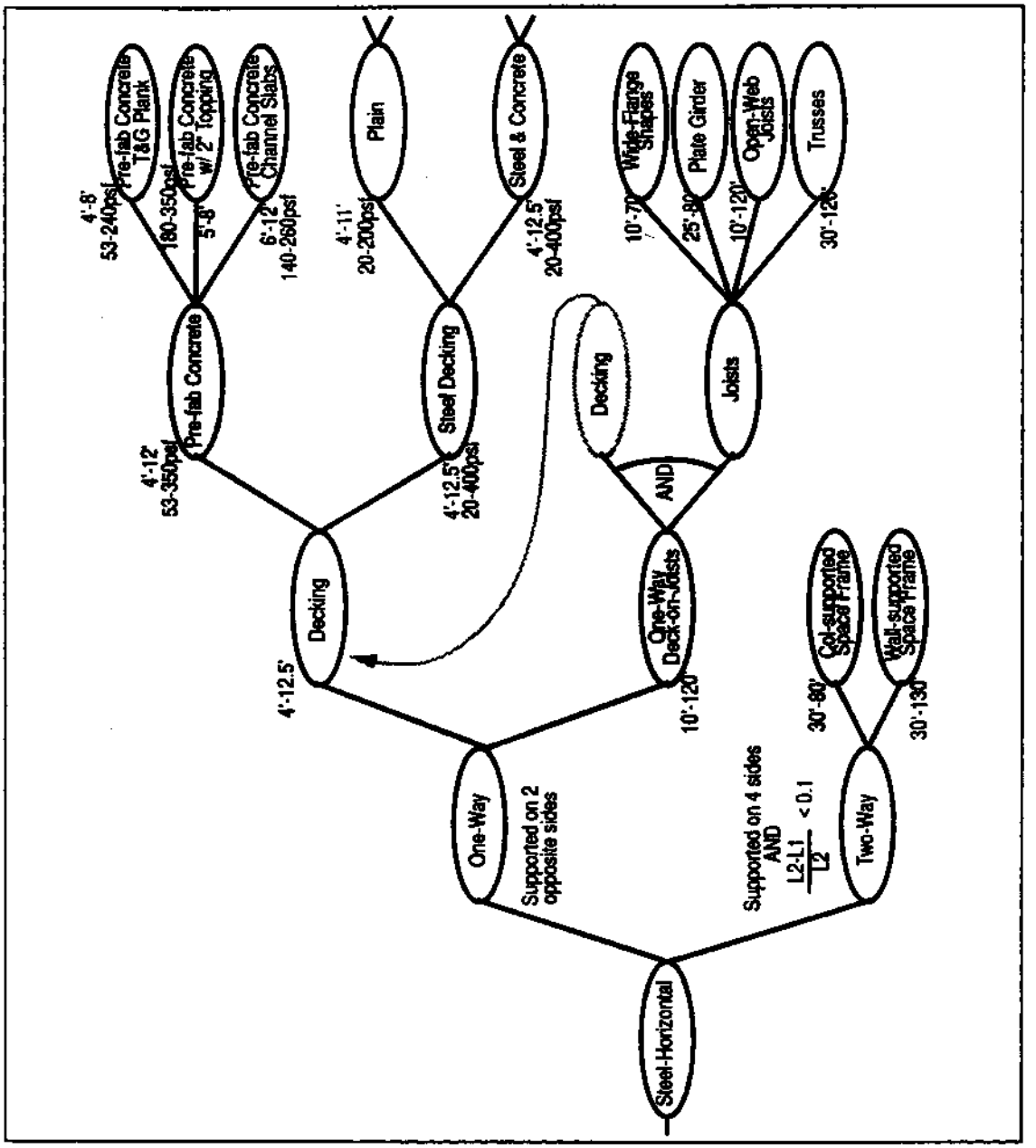

Figure 2. A sample partial structural technology tree [Fenves et al. 951

There are three categories of technologies: **decomposition technologies** are used to break down a complex design problems into sub-problems; **elaboration technologies** are used to subdivide the current entity into constituent sub-entities; and **refinement technologies** are used to provide additional detail to an entity. The first two categories are implemented in the same manner with respect to the information model. Both categories create new entities and link them to the current entity through the containment relationship slot. Refining technologies add new components to the design unit of the current entity and can operate on both abstract or detailed

entities. Figure 3 shows the relationship between an entity and the three types of technologies and Figure 4 illustrates the creation of a hierarchy of entities using the three types of technologies.

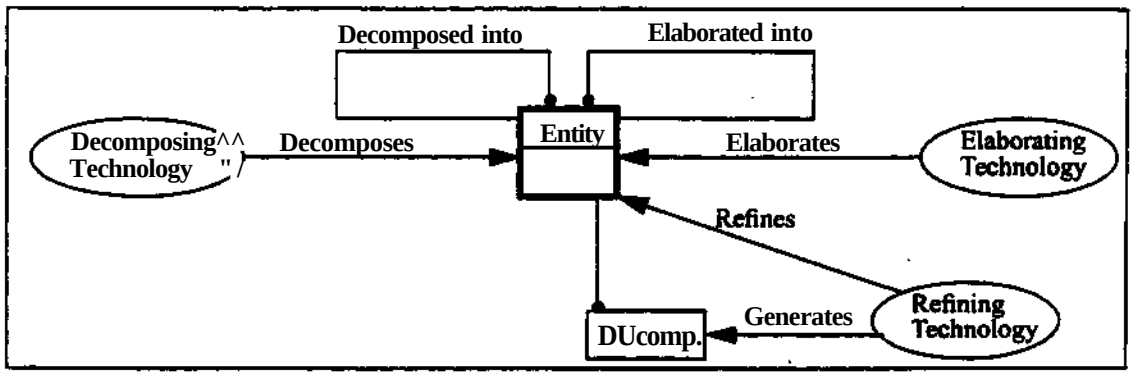

Figure 3. Object model of the relationship between entity and technologies.

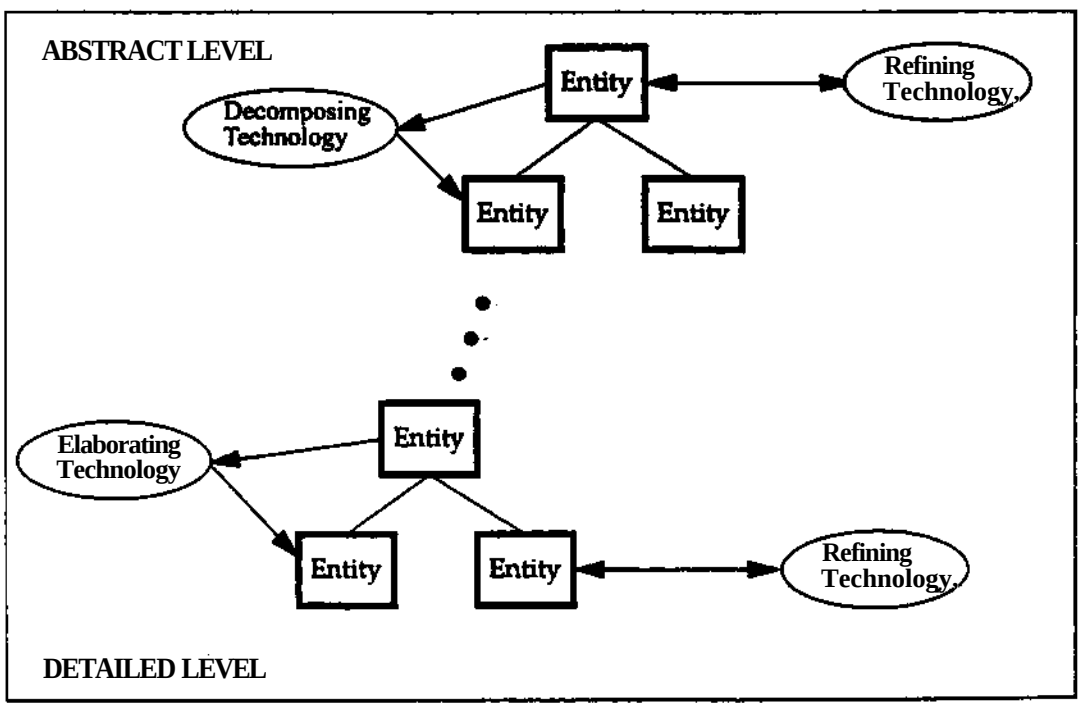

Figure 4. Use of technologies in defining a hierarchy of entities.

2.5 Classifiers

Classification is the process of systematically arranging entities into groups [Harris and Wright 81]. Classification is an important aspect of problem-solving tasks [Rich and Knight 91]. It can be used to identify a given entity or to query the set of all entities for a given group. Building

entities are classified through the use of classifiers. A classifier is an instance of a class which can be part of a class hierarchy. Hence, a class can be the subclass of a more general class. An entity is not limited to one classifier, but may have several classifiers. This approach for classification is similar to the one used in the Context-Oriented Model [Kiliccote 92]. It circumvents many anomalies and ambiguities that arise from the use of multiple inheritance in object-oriented programming [Flemming et al. 951. Figure 5 shows an example of an entity being classified as a framed tube. The fact that the entity's classifier is an instance of the class "Framed Tube" means that the entity can be generally classified as a framing system, specifically classified as a tube framing system, or even more specifically as a framed tube framing system.

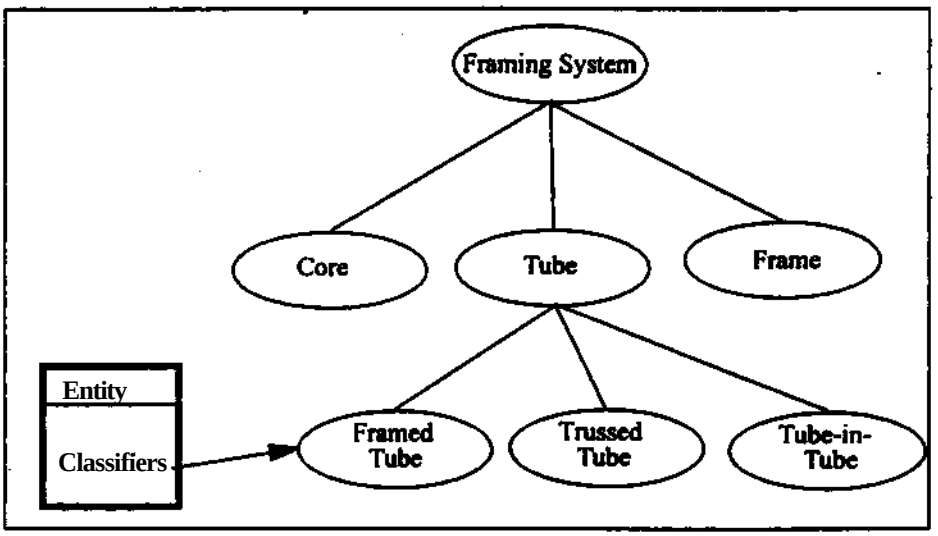

Figure 5. Classifiers and class hierarchy.

Chapter 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, decomposed such that a solution can be easily determined. This "divide-and-conquer" strategy is typical for most design processes. The information model must therefore be able to support the decomposition of design problems. Such hierarchical decompositions are also used to distribute tasks and responsibilities among designers [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 6 shows the hierarchical decomposition of the building into four systems with the structural system further decomposed. The structure of a building may sometimes be broken down into independent sub-systems depending on the building's complexity. These sub-systems, which we call 3-D structural systems, occur in buildings with expansion joints or in buildings made of several independent structural systems. The 3-D structural systems are 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 (e.g., beams, columns and slab elements). Note that the figure does not show all the relationships which may exist among the entities shown (e.g., supports and connected to). 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.

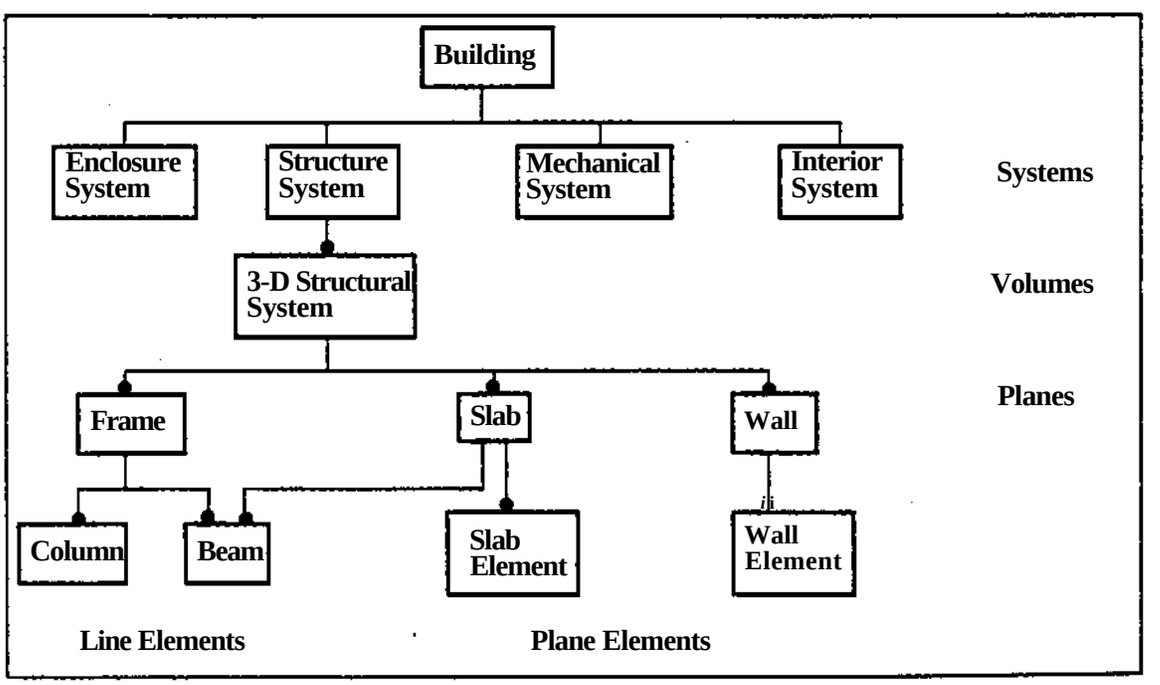

Figure 6. Hierarchical decomposition of the building structure.

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 811. 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 and is shown in Figure 7. 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 layers (i.e. construction products) such as cladding, membrane, insulation and finishing [Rivard et al. 95].

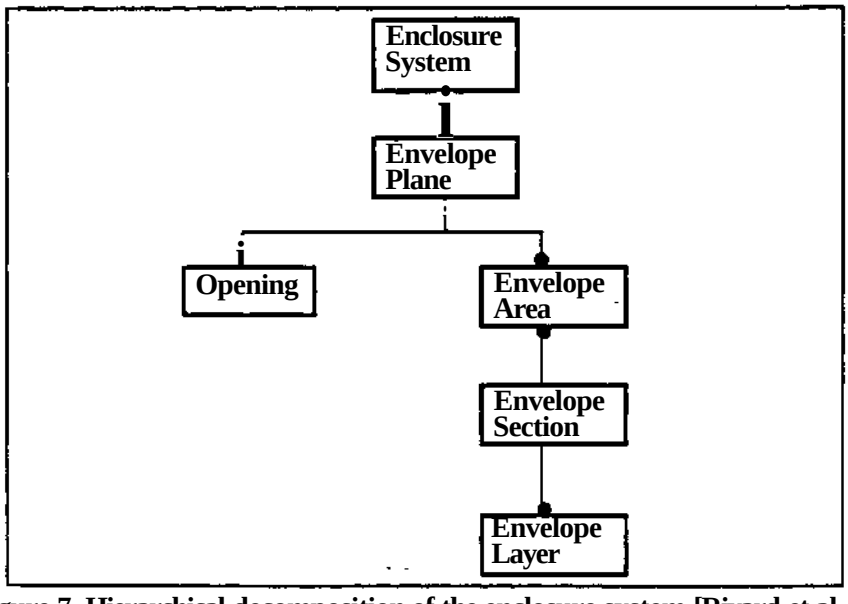

Figure 7. Hierarchical decomposition of the enclosure system [Rivard et al. 951

The information model presented here supports such hierarchical decompositions. Systems and subsystems can be modeled as entities linked by containment 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.

The information model also supports the design process as it unfolds by allowing designers to populate the design space in an intuitive manner from global system entities through sub-system entities ail 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, created from elaborations of more abstract entities, are added to the existing ones, through containment relationship, as the design proceeds and becomes more and more detailed.

It is important to note that the hierarchical decompositions described in this section are examples only. It does not mean that the hierarchical decomposition for the structure is fixed and unchangeable. The power of this information model is that any hierarchical decomposition can be supported. Hence, additional entities could be incorporated between the ones shown in the examples, and different alternative design solutions could incorporate different hierarchical decompositions.

Chapter 4

Integration of Multiple Views

The entities presented in the previous section represent a logical manner for organizing the building data and correspond to the way designers see the building. It may often 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 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.

4.1 Component Representation

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 subsets; 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 attribute values are accessed by attribute names. 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 subsets, each of which we call a component. This approach, called the Primitive-Composite Approach (or P-C Approach), was originally suggested 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 advantage that it supports multiple views, schema evolution and data integration.

Cohesion is the only criterion used in decomposing entities. It is defined as a measure 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 component is slightly different. We keep only three of the five original criteria. Hence, a component 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 (e.g., 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 component exists, he or she can assume that all the attributes in it have a value. If the component does not exist, it means that the data has not been generated yet, and he or she may create a new component. This criterion implies that each component 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 overly subdivided. It does not really matter if all data attributes of a component 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 component. Methods could be used from within the component to compute the dependent information.

The component representation provides an abstraction of the entity to designers. Views hide the actual complexity of the entity by providing only **the** relevant information while hiding the unnecessary details. Figure 8 shows a wall entity decomposed into a set of components. Three different views are shown as referring to a subset of the components. Two of the components 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.

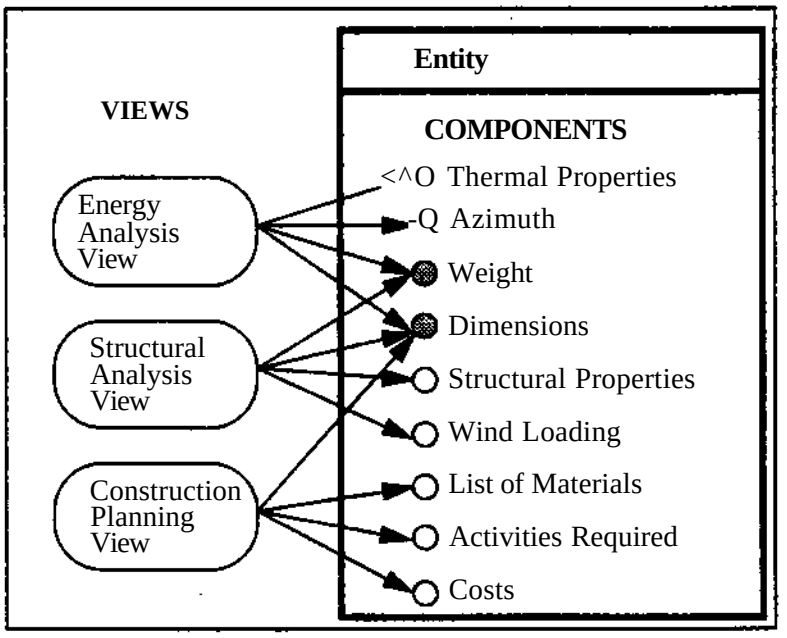

Figure 8. Multiple views of a wall entity [Rivard 941.

4.2 A Generic Component Class

The definitions of the components are inherited from a common superclass, which is shown in Figure 9. A component class should have a unique name. Each component 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 design). A component should include a reference to all the entities to which it belongs. The fact that a given component 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 component should have a built-in help mechanism that would allow the user to obtain a description of the data stored in it A component can have both attributes and methods (or procedures). Methods are used to compute values based on other components. The use of methods provide support for dependencies among data.

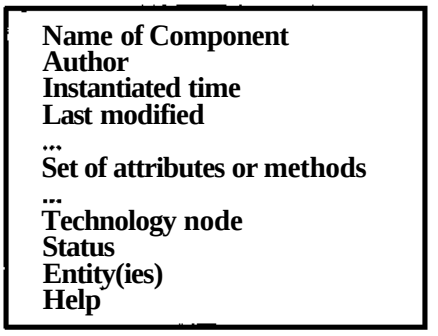

Figure 9. A generic component class.

An advantage of encapsulating data within component objects is that the type of data stored could be of multimedia type. The component has the appropriate methods to display, edit and input its data. Therefore, a component could contain images, sounds, texts and even video. For instance, an evaluation unit component could contain the bending moment diagram of a structural beam.

Concurrent use has not been considered at this stage. Since changes are usually localized to the component levels, only the components would need to be locked. The system could allow the display of components created by other users but protect them from modification if a user does not have write permission.

4.3 Relationship Between Components, Entities and Technologies

An entity contains a reference to the technology node that created it. As the entity is refined, design unit components are added to the entity. Figure 10 shows the relationship between an entity, its design unit components 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. 951. The design process proceeds as follows: the child technology nodes of the current technology test the entity against their own constraints and determine whether they are applicable or not. If a technology node is applicable, it instantiates the appropriate design unit component(s) and assigns corresponding design attributes. The designer selects one of the candidate components to expand further with the technology tree. The selected component is automatically incorporated into the entity.

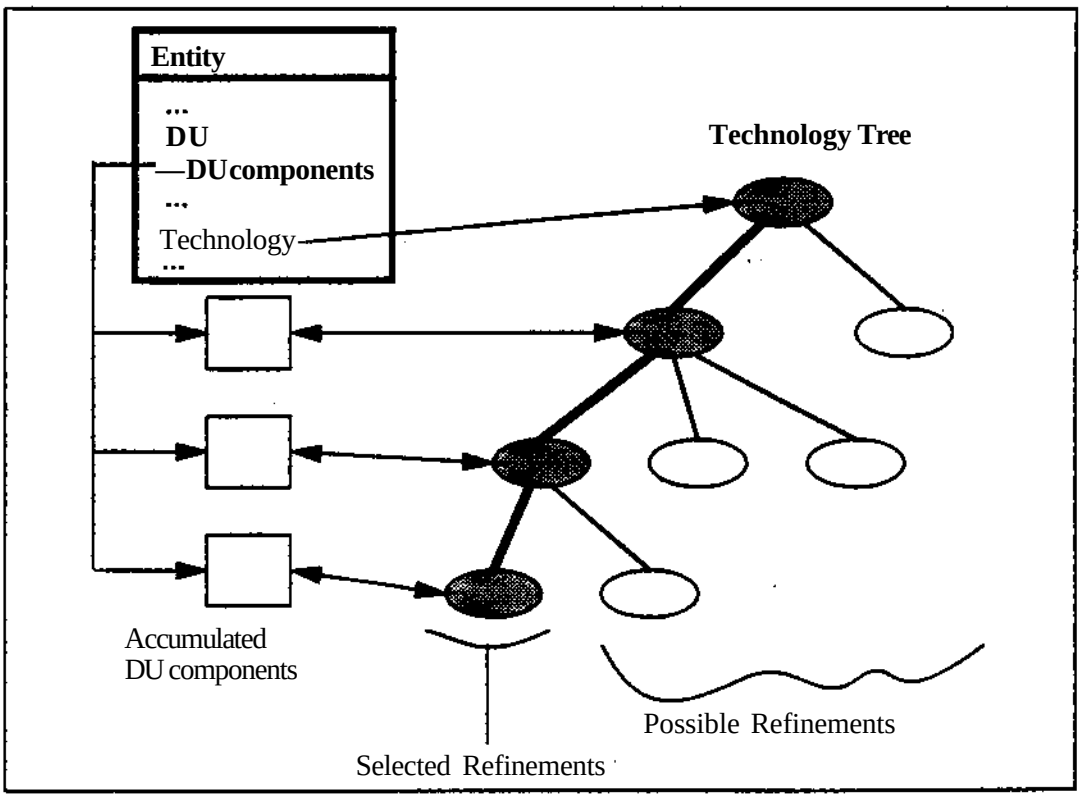

Figure 10. An entity , its design unit components and a technology tree

4.4 Integration Across Hierarchical Decompositions

The primary spatial representation of a building entity, described in section 2.2, can be used to identify an entity in the geometrical model and then to add appropriate components to the entity. For instance, a structural function may be enforced to a building envelope entity (such as an exterior wall), found in the geometrical model, by adding the proper relationships, FU components and DU components. The designer could access the entity through the geometric model and add it to the structure hierarchical decomposition. Hence, the exterior wall becomes part of two hierarchical decompositions: the enclosure and the structure. The decomposition hierarchies become lattices.

4.5 Search Mechanism for Components Within Building Entities

The building entity's information is encapsulated within components. The information is accessed through a search mechanism. First, the requester (which may be a user or a computer application) must specify one of the subsets where the search is to be made: either functional unit, design unit or evaluation unit. Second, the requester must provide the name of the component that is of interest. For example, say someone is interested in checking the water-ratio of a concrete structural element, s/he would access the corresponding building entity and provide the following request: design-unit (concrete characteristics). The system would look among the components stored in the design-unit and return the one with the matching name. Once the requester has found access to the component, s/he can fire any of its methods such as display or edit attributes. In our example, the person would request the display of the water-ratio attribute.

The building entities have a mechanism to display all the attributes of their functional units, design units and evaluation units for quick reference. Whenever someone requests that facility for one of the subsets of data, the building entity displays all of its attributes and their values under headings corresponding to their component names.

Chapter 5

Grouping Entities

To facilitate the design process, a set of entities can be grouped together and designed simultaneously. We have identified three different types of groups: "same" means that all the entities are assigned the same design components according to the most constraining entity of the group; "similar" means that all the entities are designed with the same technology but may have design components with different attribute values; and "identical" means that all the entities in the group have the same dimensions and are subject to the same conditions and hence can be designed just by looking at one of the entities. As for "same", the entities grouped by "identical" are assigned the same design components. While the two groups "same" and "similar" are assigned by the user, the group "identical" is assigned by default by elaborating technologies.

As an example of the use of a "same" group, a structural engineer may want to assign the same concrete characteristics to a group of entities. The engineer selects the group of entities and the refining technology that assigns the concrete characteristics; the technology would check that the structural material of each entity is concrete and would assign the specified concrete by adding the same DU component to all entities. Having only one DU component for a group of entities ensures that any changes will be applied to all entities (i.e. there is no unnecessary redundancies). This example is illustrated in Figure 11 below. For a group of "similar" entities, the technology would add a new DU component to each entity.

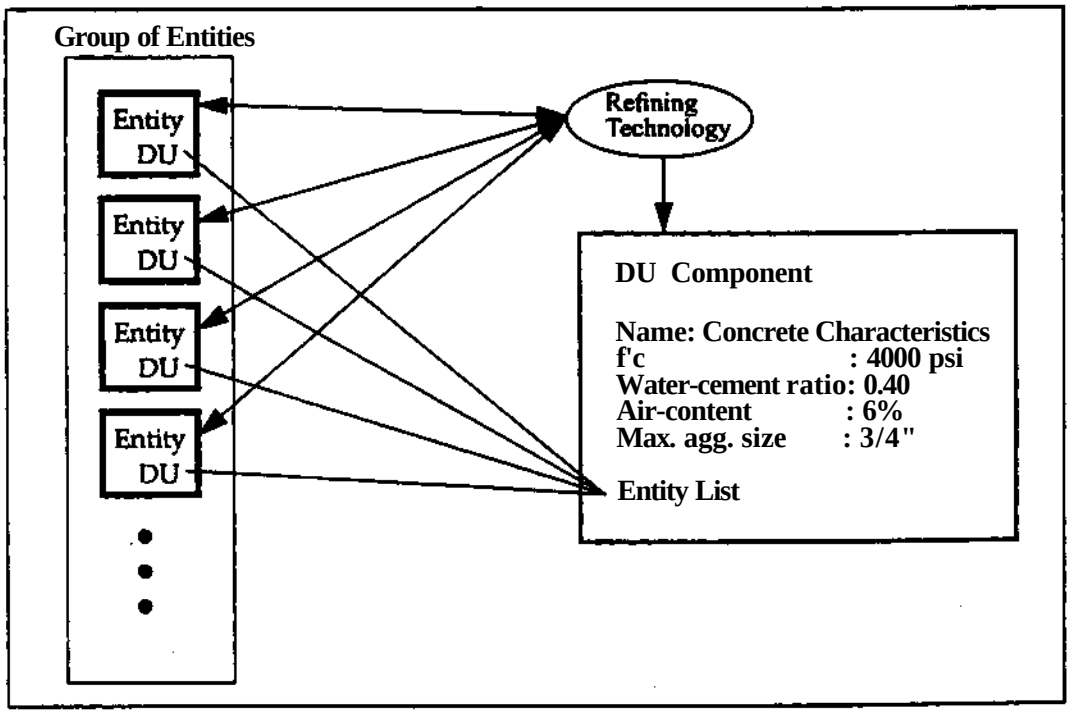

Figure 11. Designing a group of "same" entities.

As an example of an "identical" group, when elaborating a structural slabs into beams and deckings, the technology creates a number of beam-entities and a number of deck-entities. All these entities are associated to the slab-entity through the containment relationship. Each distinct sub-entity is needed because each one has its own distinct primary spatial representation. But to simplify design, all the beam-entities are automatically grouped into an "identical" group and thus ensure consistency between the beams. When a technology refines the beam-entities, it assigns the same DU components to all of the entities in the group. The same process applies for the deck-entities. Figure 12 below illustrates the creation and design of the beam entities.

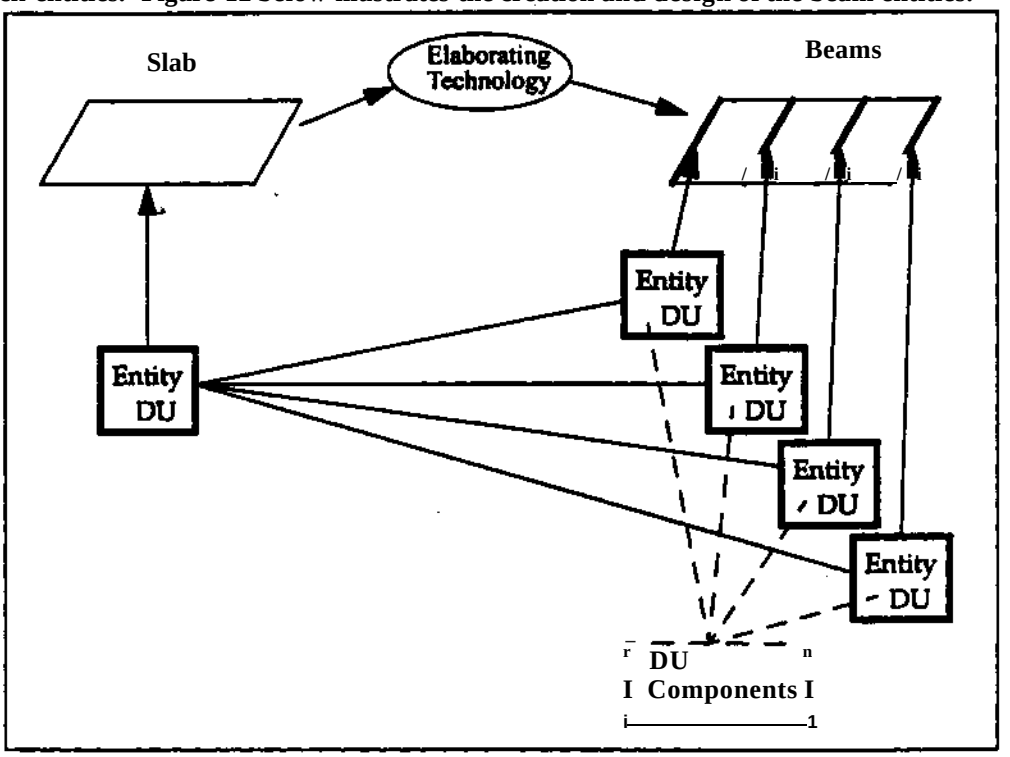

Figure 12. The creation of an "identical" group of entities.

Chapter 6

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 ieams as new cases are added to the cast

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 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 94]. 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 component 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.

Flemming's case retrieval mechanism for a hierarchy of objects would work with this information model [Flemming 94]. The only differences are that instead of traversing a hierarchy of functional units, we are traversing a hierarchy of building entities; and that an additional parameter is necessary when searching for an attribute: the name of the component which contains the attribute of interest.

The fact that a component refers back to the entities it belongs to allows a case search to be done from bottom up. All the components that satisfies a search criteria could be used as a basis to find matching cases. The name of the components could be used as an index to limit the search to a particular type of component. Once cases (or building entities) are retrieved through the matching components, they can be pruned based on other search requirements. The classifiers are another mean for retrieving cases. They support case retrieval from general to specialized entities.

Chapter 7

Conclusion

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

- data 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;
- references to the computational mechanisms (or technologies) used in designing it; and
- a set of classifiers.

The data of an entity are further grouped into components 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 record design data as it is generated during the design process and to support case-based reasoning. 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.

References

Burkett, W. C. and Y. Yang (1995). "The STEP Integration Information Architecture." Engineering with Computers, Springer, Vol. 11, No. 3, pp. 136-144.

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.

Flemming, U., Z. Aygen, R. Coyne and J. Snyder (1995). "Case-Based Design in a Software Environment that Supports the Early Phases in Building Design." To be published in "Issues and Applications of Case-Based Reasoning to Design", Maher, M. L. and Pu, P. (eds), Lawrence Erlbaum Associates.

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

Harris, J. R., and R. N. Wright (1981). "Organization of building standards: systematic techniques for scope and arrangement." Washington: U.S. Government Printing Office.

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.

Kiliccote, H. (1992). "The Context-Oriented Model: A Hybrid Approach to Modeling and Processing Design Standards." M. Sc. Thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.

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.

Rich, E. and K. Knight (1991). "Artificial Intelligence." 2nd Edition, McGraw Hill, New-York.

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.**

Wix, J. and D. P. Bloomfield (1995). "Standardisation in the Building Industry: The STEP Building Construction Core Model." Publication 180, International Council for Building Research Studies and Documentation (CIB), Stanford, California, pp. 184-195.

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 Infonnation about Constructed Facilities." Phd thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.