Adjacency Structures as Mappings Between Function and Structure in Discrete Static Systems

Steven Meyer, Steven J. Fenves

EDRC 12-49-92

Adjacency Structures as Mappings Between Function and Structure in Discrete Static Systems

A Working Paper

Steven Meyer & Steven J. Fenves
Department of Civil Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

One view of the design process is that design is a mapping from functional requirements to artifact description. This article presents initial work on a method for mapping between functional requirements and a description of the physical structure of discrete static systems. The representation consists of a set of atomic elements, a hierarchy of compound components from the domain, and the composition of a graph of adjacent atomic elements. Through forward or backward chaining, this method may be used in a parsing mode to discover the behavior and function of a given system, or in a generative mode to suggest instances of systems which can be used to satisfy the desired functionality. Parsing discovers the behavior of the system in terms of the compound components by matching on subgraphs within the overall adjacency graph. Generation hierarchically instantiates subgraphs which satisfy the initial functional requirements and the requirements propagated by previously instantiated components. The graph is composed from a geometric model, but the method is independent of the specific representation used by the geometric modeler. We focus on the domain of structural systems in buildings to describe this method.

This work has been sponsored by EDRC, the Engineering Design Research Center at Carnegie Mellon University, an NSF-sponsorcd Engineering Research Center.

1 Introduction

In many domains a finished design consists of a description of the physical artifact's structure.[1] The artifact*s structure is a description of its topology, geometry, material and manufacturing details. However, this description does not normally contain a full specification of the artifact's desired functionality. In contrast, at the start of the design process the design mandate consists almost entirely of specifications of the artifact's desired functions. Between the initiation of the design process—when no physical structure has been given to the projected artifact—and the beginning of the manufacturing process—when a complete physical description must be specified—the design process consists of an iteration of mappings between function and structure. That is, a structure for a system or subsystem of the design is suggested which is thought to meet the given functional requirements through its expected behavior. Then, this structure is evaluated to determine whether it meets the behavioral and functional requirements and what its side effects might be. This loop continues until the current structure can be mapped to the required functions, and vice versa. Thus, in an evaluation-synthesis-analysis design loop, the mapping from function to structure is a high level view of the synthesis stage, the mapping from structure to behavior is a view of the analysis stage, and the mapping from behavior to function is a high level view of the evaluation stage. We describe a method for performing the two-way mappings between structure and function within discrete static systems based on the adjacency structure representation, a representation that extends the concept of adjacency graphs by adding geometric, material and behavioral information to each node.

This paper is organized into 6 sections. Section 2 presents a review of some of the related research and defines discrete static systems—the design domain which motivates this research. Section 3 presents adjacency structures in terms of their elements and composition. The introduction to adjacency structures begins with an example to give the reader a feel for the information we would like to express using this representation then describes the atomic and compound elements of adjacency structures. Section 3 also presents three aspects of the construction of adjacency structures: composition operations, the inclusion of constraints in the composition of adjacency structures and a method of generating adjacency structures from a geometric model. The next two sections present the potential uses of an adjacency structure representation in design processes, presenting the two directions of the function-structure mapping. Section 4 presents the parsing of a given adjacency structure, describing how the function and behavior of a given structure are discovered, whereas Section 5 presents the use of adjacency structures in the generation process, describing how a set of functional requirements and a minimal geometric description can be used to synthesize adjacency structures in a behavior-oriented derivation of structure. The paper ends with a summary and brief discussion of the use of adjacency structures within a larger synthesis system.

---

[1] In this paper we adopt the terminology of [Gero 91] using the *term function* to describe the intentions and purposes of the artifact being designed; the term *behavior* to describe how the functions are achieved in the design; and the term *structure* to describe the physical components specified by the design. An example of these distinctions in structural engineering is that the function of a building is to resist lateral and gravity loads within a prescribed limit of deflection, vibration etc.; the structure of the building may be a set of beams, columns and slabs of a particular material arranged in a specific configuration; the behavior of the building may be that it resists the applied loads through the flexural strength of the components.

2 Background and Motivation

In the domains of **interest, the** form variables—the topological and geometric aspects of the design which are to be determined **during the** design process—are given values in order to satisfy functional requirements—the function variables. **The** form variables may also generate additional function variables or help to complete the specification of existing function variables. These function variables are generally at a higher level of abstraction than the form variables. A useful design method and representation must be able to mediate between the multiple levels of abstraction used during the design process, tracking the detailed representation of geometry as well as the abstract representation of function and behavior. The adjacency structure method we propose may be viewed as a geometric modeling representation that focuses on design components at a higher level of granularity than typical geometric modelers. A boundary representation geometric modeler, for example, represents a design component such as a rectangular beam as a graph containing vertex nodes, edge-half nodes, face nodes, etc. As the number of design components increases the representation becomes extremely cumbersome to manipulate. Adjacency structures represent an individual design component such as a beam as a single node, and form a graph of adjacent nodes to represent the system of connected components in a design. Additionally, the geometric model is a purely syntactic representation having few facilities for expressing the non-geometric aspects or the domain dependent semantics of the objects being represented. The inclusion of material and behavioral information in the nodes of the graph allows the expression and propagation of functional and behavioral aspects of the system of components.

The representation of design elements within a particular domain using graph structures has been the subject of considerable investigation, notably for the specification of languages in computer science [Knuth 68, Nagl 79], in architectural research [Mitchell 76, Krishnamurti 78, Hemming 86a], and in dynamic systems design [Karaopp 68, Ulrich 87, Finger 89]. Each of these areas has provided domain or theoretical background for the ideas presented in this paper. The relevant contributions will be briefly sketched in the remainder of this section.

2.1 Graph Grammars in Computer Science

One method of specifying the logical structure of programming languages and algorithms is through the use of graphs. The best known method may be the traditional flow chart but more formal methods exist, particularly for the specification of compilers [Culik73, Schneider 75]. An operational definition of a program can be given concisely by a program graph with the nodes of the graph representing data structures and operations on **the** data and the arcs of the graph representing data flow. Program graphs are an abstract notation that facilitates data-flow analyses in a manner which textual algorithm specification methods are unable to provide. An interpreter for such a program specification can be described as a graph rewriting system. Substantial research on the formal characteristics of graphs has been performed to ensure desirable properties in a system expressed as a graph rewriting system [Ehrig 86, Nagl 86]. Also, a number of different graph rewriting systems, or grammars, have been characterized to operationalize the development of specific languages expressible using graph representations [Rozenberg 86, Deransart 88, Bunke 79]. These rewriting systems differ in the definition of the vocabulary elements and in the details of the transformation mechanisms. The formality of rewriting systems operating over graphs, and their utility in design through their expression

of complex structural interrelationships at multiple levels of abstraction, is attractive from the viewpoint of both the development of system logic and its implementation. The ability of graphs to represent complex structural relationships between any type of object being modeled by the nodes of the graph allows a hierarchical and multidimensional system to be formally evaluated Additionally, the similarity between the representation of the program's specification language and the internal representation of the program's data structures simplifies the implementation of the logic invested in the language.

The application of graph grammar variants has begun to receive attention in the engineering design field [Finger 89, Pinilla 89, Rinderle 91]. The necessity for representing both the geometric and non-geometric aspects of partial designs, and the need for basing the transformations in the design process on both aspects of the representation, focuses our attention on the attribute grammar formalism begun by Knuth [Knuth 68]. As described more fully by Deransart, et al. [Deransart 88], an attribute grammar begins from a context-free grammar specified by the four-tuple:

$$G = \{N,T,P,Z).$$

Where

- ***N:*** The finite set of non-terminal symbols,
- ***T:*** The finite set of terminal symbols,
- ***P:*** The finite set of productions of the form $XQ \rightarrow Xi.XiyX^{\wedge}, ...X^{\wedge}$ $y/ithX_oeN.$
- **Z:** The starting symbol of the form *afli* with */3* € */V,* a, 7 € *{N* U 7}*, the set of all finite length strings composed of symbols from *{N* U T}.

An attribute grammar associates an *attribute system* to the underlying grammar. Each non-terminal symbol except the starting symbol has two finite sets associated with it: a set of inherited attributes and a set of synthesized attributes. The inherited attributes of a symbol can be evaluated from the attributes of its parent symbol in the graph, whereas the synthesized attributes can be evaluated from the children symbols' attributes. Therefore, the start symbol has only synthesized attributes and the lowest level symbols have only inherited attributes.[2] Each production/? £ *Pis* given a set of*semantic rules* defining how the attributes *Syn(Xo)* and *lnh(Xj),* $1 \leq j \leq n_p$ *art* computed for the elements of *AttriXi),* $0 \leq * \leq n_p$. Each semantic rule is a function over the production, the symbols in *N* and the attributes. Within a production ah attribute associated with the i^{th} position of symbol *X* is called an attribute occurrence *a(i).* More specifically:

for each Xi e N, $AttrtX) = Inh(X)$ U $Syn(X)$,
for all X,r e N_9 $Inh(X)$n$Syn(Y) = 0$

$$a(i) = f_{p,a,i}(a_j(i_j),\ldots a_k(i_k))$$

$a(i)$ € $Syn(Xo)$ for / = $OJ = I$ *and* $k=n_p$
$a\{i)$ € $Inh\{Xi)$ for $1 \leq i \leq n_p$ and; = $k = 0$

[2]TypicaUy, the terminal symbols have no attributes, but this is not a necessary property of the formalism. Assumedly terminal symbols do not have, or need, a semantic interpretation. More accurately, the whole tree is used to derive the semantics of the terminal string.

The semantic rules associated with a production *p* induce an order of computation, the *local dependency relation* $D(p)_9$ on the set of all attribute occurrences in *p*. Properties of the local dependency relation reveal the attendant properties of the grammar. An attribute grammar is well-formed **iff** for every derivation tree *t* of the grammar G» **the** collective dependency relation *Rd(f)* is cycle free. For a well-formed grammar an evaluation order exists and every attribute's value may be determined.

Attribute grammars have been used in computer science for compiler and program specification, for data-flow analysis and for database specification. Two interrelated points are worth noting about the standard definition of attribute grammars and their applicability to engineering design:

1. There are essentially two coupled grammars; a symbol grammar and an attribute grammar, with a strict partitioning of the total vocabulary. For a compiler it is relatively easy to partition the vocabulary into programming language symbols and machine code "attributes". For an engineering design grammar it is not as simple to partition the relevant vocabulary into a syntactic set and a semantic set.

2. The influence between symbols and attributes is unidirectional. The attributes are used to present a second view of the derivation tree, but take no part in guiding the productions towards their final state. In the original definition, the attributes form a semantic interpretation of the syntactic state. This approach parallels the unending debate within linguistics of the relative primacy of syntax versus semantics. In engineering design the interaction of syntax and semantics necessitates bidirectional influence between the two descriptive systems.

The partitioning of the vocabulary into two grammars is itself a difficult issue when syntax and semantics are fuzzy definitions for a domain. It may seem natural to associate the form variables to the grammar symbols and the function variables to the attributes. However, if the semantic rules are a mapping to the attributes of a symbol only, the function variables can have no impact on the form variables. If the opposite choice is made, associating the function variables to the symbols and the function variables to the attributes, the form variables have no feedback to the function variables. Either partitioning tactic reduces the interplay between form and function variables. This is a severe restriction on the expressiveness of the transformations and therefore on the expressiveness of the grammar for engineering design.

2.2 **Orthogonal Structures in Configuration** Studies

In configuration studies, Flemming's orthogonal structures represent the topological relations of a set of rectangular shapes placed within a bounding rectangle. This representation has formed the basis of a two-stage method for exploring the possible non-overlapping placements of component rectangles within a bounding rectangle [Flemming 86b]. Two classes of constraints are successively employed: topological constraints which specify the spatial relationship between rectangles and geometric constraints which restrict dimensional properties (e.g., maximum or minimum area of a rectangle). The topological description specifies a class of solutions containing all the geometric instantiations. This approach is exemplary in abstracting the topological and geometric aspects of a rectangular dissection in order to develop a formal representation of the topological aspect as a graph whose nodes represent the rectangles and whose directed arcs represent one of two spatial relationships, above or to the right. A well-formed solution is assured by

proving that the representation is closed and complete under the application of a small set of generative rules. Thus, the representation is used to prove theorems about the solutions, placing the approach on a firm theoretical basis.

Two aspects of orthogonal structures discount their use in representing the structural systems of buildings. First, the rules for generating orthogonal structures are embedded in a design method in which the precise number of elements to be used must be known beforehand and the elements are added one at a time while meeting a set of a *priori* adjacency constraints. In a structural system the number of elements employed may change from one potential solution to the next; the exact number of elements is rather unimportant. Likewise, the adjacency requirements on the elements of the design are a function of the partial solution rather than a part of the problem statement. Secondly, we would like to be able to employ elements whose edges are not necessarily parallel to an orthogonal grid. That is, we would like to be able to use diagonal elements such as those in trusses or braced frames. Therefore, we are inspired by orthogonal structures and their formation of the basis of a design method, but we are searching for a representation more appropriate to our domain.

23 Bondgraphs for Dynamic Systems

Bondgraphs are a formal representation for describing the transformation of energy in lumped-parameter systems [Paynter 61]. The applicable domains are classified according to the type of their energy transfer electrical, fluid, mechanical translation and mechanical rotation. A bondgraph is composed of ports and bonds; the nodes and vertices of the graph. There is one type of bond and four types of ports: sources, 1-ports, 2-ports and N-ports. Each bond represents a path for the flow of power which may be described by the combination of a flow variable and an effort variable [Ulrich 87]. Thus, power flow in the mechanical translation domain, for example, is the product of force and velocity which are the effort and flow variables, respectively.

Source nodes describe elements of a system which specify an effort or flow. Lumped-parameter elements are represented with 1-port nodes which constrain the effort-flow relationship on its associated bond. For example, a 1-port representing a mass constrains the derivative of the velocity of the mass and the force on that mass to be related by the mass parameter embodying Newton's Second Law. Elements of the domain which transform efforts or flows are represented as 2-ports. The transformation may be between an effort and a flow within the same domain, or into the same variable type in another domain. N-ports represent junctions of bonds where all the bonds share a common flow and whose effort sums to zero, or where all the bonds share a common effort and their flows sum to zero.

There are many attractive features of the bondgraph concept. First, bondgraphs represent the important components of a system along with their behavioral attributes in a computable form. By explicitly representing the the effort-flow relationships among all the elements of the graph the qualitative representation of bondgraphs can be used to derive a quantitative description of the system's behavior. Second, bondgraphs may represent subsystems or complete systems, and therefore two complete bondgraphs may be used to compose a larger bondgraph using an n-port and its bonds to connect the subgraphs. Therefore, the concept of prototypes[Gero 90, Gero 88] is readily extended to bondgraphs through the use of standardized subgraphs in the composition of system level bondgraphs. Finally bondgraphs are a formal language with a well-defined

grammar allowing methods for ensuring well-formed graphs.

Although structural systems are often represented using mass-spring systems during analysis, this is an overly cumbersome representation for the design of systems with a large number of components. Also, bondgraphs arc not applicable for systems with components whose behavior is context-dependent. For example, the reduction ratio of a gear is independent of the rotation direction or applied torque, but the deflected shape of a column is dependent on its support conditions. Furthermore, it may be argued that by the time a building can be modeled as a lumped-parameter system, the configuration problem, and therefore a major part of the structural design problem as a whole, has been completed. While the analogy between the specification of the power variables of effort and flow and the work variables of force and displacement for the description of a "static" system may lead to a quantitative description within our qualitative adjacency structures, we contend that the geometry of a building is a crucial element of this domain and must have a major role in the representation and design process, an aspect which bondgraphs ignore. Nevertheless, another inspiration in the development of adjacency structures is the schematic description of systems using bondgraphs.

2.4 Discrete Static Systems

In discrete static systems there is no action at a distance. There are no magnetic or electrical fields. There is no transmission of electric energy between components of the system. In a discrete static system, such as the structural system of a building, all communication of forces occur through the physical, **virtual** "pushing and pulling" of adjacent members of the system. Systems are composed of adjacent members arranged in a particular orientation. This orientation may be with respect to other members of the system or with respect to the environmental conditions (loads and displacements) the system is meant to counteract. A common informal language for describing the behavior of a building's structural system is in terms of load paths. These paths consist of networks of adjacent members and subsystems which communicate the loads from their sources to their supports (sinks). The introduction of any physical discontinuity in the network may obviate a load path. The introduction of a new continuous path from load source to sink within the network can introduce a new load path dependent on the geometry of that new path. The new path may quantitatively change the distribution of forces within the system or qualitatively change its behavior. The confirmation of expected load paths, and the discovery of new load paths and their quantitative assessment is one view of behavioral evaluation. The circulation and communication of building occupants between the architectural volumes of a building is another example of a functional requirement of a system predicated on the adjacency of elements (rooms, corridors, floors etc.) of the building.

Of course, building structures are not completely static—they bend, compress and shear in response to lateral, gravity and temperature loads. Yet, the members remain in a relatively fixed position and orientation in relation to other members in the system. Also, it is a complex process to quantitatively describe the motion and forces within a given building as it responds to these loadings. Many mathematical methods such as matrix and finite element methods have been used to quantitatively describe the transmission of forces within structural systems. However, during the design process we would also like to have a more qualitative and less computationally expensive method of describing-the interrelation of physical members. Furthermore, we would like to be able to describe the potential interrelationships of physical members before the complete

geometry (e.g. member sizes) of the system has been determined. Nevertheless, we take another major inspiration for adjacency structures from the assembly step within the matrix and finite element analysis methods.

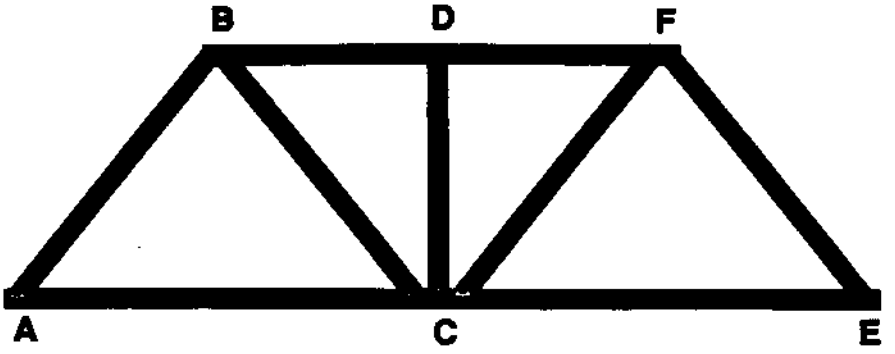
3 Adjacency Structures

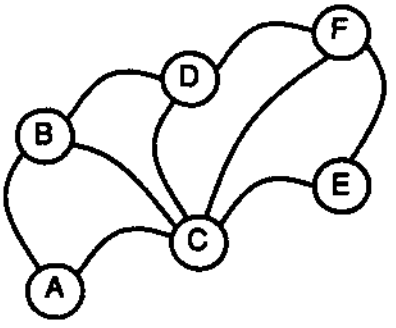
This section begins by introducing adjacency structures and their expressiveness using an intuitive example of a truss. Then, we begin a detailed presentation of the adjacency structure concept by describing the atomic and primitive elements that are used to compose adjacency structures. Next, the types of composition operations for developing higher-level adjacency structures are presented, after which a few of the possible compound components from the domain are described. Finally, the types of constraints incorporated into the non-atomic components are described and a method for discretizing the geometric model to produce a canonical translation to and from adjacency structures is presented. The translation of the complete geometric model into an uninterpreted adjacency structure will be referred to as the *overall adjacency structure.* As typically used, an adjacency graph is a purely topological description of a system. Each element of the system is represented as a node in the graph and each directed or undirected arc represents an adjacency relation between the two elements it connects. Our extension of adjacency graphs to adjacency structures adds basic geometric and material information to each node in a graph with undirected arcs.

We begin the presentation of adjacency structures by considering a truss. Figure 1 shows a simple nine-bar truss and three graphs. Graph *a* represents the truss joints as the graph's nodes and the truss bars as its arcs. This representation is adequate for describing the spatial relationships of the bars. It can be seen that each node is a member of a minimal cycle with a length of three. Furthermore, if each node contains its coordinates in three-space it could be determined whether or not the truss lies in a single plane. However, since this representation contains no information about the elements connecting the nodes it is inadequate for answering questions about the functional adequacy of the truss such as its resistance to applied loads or its resistance to buckling. Graph *b* represents the bars of the truss as nodes, with the arcs representing adjacencies among the bars. It can be seen that the proper cycle information has been lost. For example bar *BC* is a part of a cycle of length three containing bars *AB* and *BD*, three bars adjacent at node *B* but which do not form a triangle. This representation fails on its inability to represent the topological nature that characterizes a truss. Graph *c* contains two types of nodes. Triangular nodes represent the truss bars and circular nodes represent the truss joints. The arcs of graph *c* simply represent the adjacency relations between the bars and joints of the truss. In this representation each node is a member of a minimal cycle of length six composed of three pairs of alternating bar and joint nodes. Furthermore, the representation of joint nodes allows for a simple detennination of the planarity of the truss and the attributes associated with the bar nodes allows the determination of the functional adequacy of the truss. The remainder of this section presents a more specific description of the composition of adjacency structures.

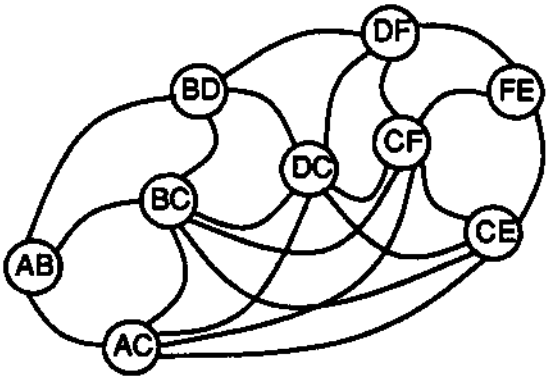
3.1 Elements of Adjacency Structures

The physical objects to be represented by the adjacency structures described in this section are those objects representable in any geometric modeling system. Section 3.4 briefly describes the translation requirements

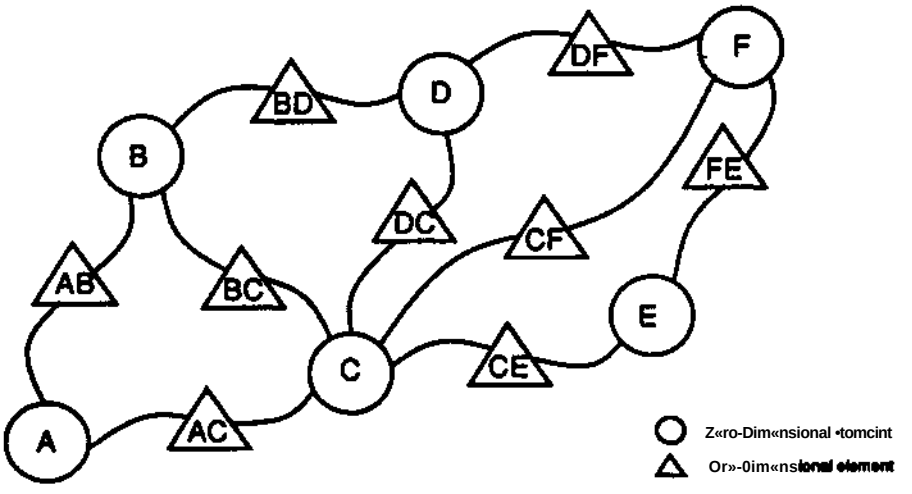




a. Nodal Adjacencies



b. Member Adjacencies



C. Adjacency Structure

Figure 1: Nine-bar truss and associated adjacency graphs.

of various geometric modeling schemes. In this section the two lowest levels of a representation hierarchy are presented. **The** classification begins with the types of nodes in adjacency structures and the objects they may represent, i.e. the leaf nodes of the hierarchy. Next, two primitive elements of the structural engineering domain are described. Operationally, the hierarchy may be composed through a graph grammar, and Section 3.2.1 begins by presenting the three types of graph composition operations before presenting a number of system components from the representation hierarchy.

3.1.1 Atomic Elements of Adjacency Structures

The leaf nodes, or *atomic elements,* of the representation hierarchy are the indivisible nodes of the graph structure. These atomic elements of adjacency structures are translations of the objects in the geometric model, and are classified according to their gross dimensional proportions. We allow four classes of nodes or *atomic elements* based on the gross dimensionality of the element; we admit zero-, one-, two- and three-dimensional nodes.

A zero-dimensional node has position but no size or form. This node may represent an interface between two adjacent elements of a geometric model such as a joint in a truss or frame.

A one-dimensional **node** has a length greater than both its cross-sectional dimensions and may be used to represent a bar of a truss, a column or a beam. Its geometric information may be represented by its two end points.

A two-dimensional node has a breadth and depth much greater than its height, and may represent a wall or floor plate. The geometric information of a two-dimensional polygonal node may be represented by an ordered list of its vertices.

A **three-dimensional node** has each dimension of roughly the same scale, may represent an architectural volume or a foundation footing, and may have its geometric information represented as a nested list of the vertices of its bounding faces.

Each node is represented using a common data structure regardless of its dimensionality. The requirements of the data structure include the ability to represent non-physical attributes which express aspects of the node's behavior and the ability to model the environmental conditions such as loads and displacements for which we are constructing load paths. A single data structure is used to represent both physical or *member objects* as well as *virtual objects. Member* objects encompass those physical objects being modeled in the geometric modeler, e.g., truss bars and floor slabs, whereas virtual objects include the loads and displacements imposed on a system as well as the interface between adjacent member objects. The data structure contains five attribute fields: dimensionality, geometry, magnitude, composition and stiffness. There are two additional fields, one for a node identifier and another for a list of pointers to other data structures. Additional fields may be used for other domains, but these fields provide a compact yet expressive representation of the geometric and non-geometric aspects of a single design component in the domain of tall building design. This representation is shown in Table 1.

	Objects			
Fields	Member	Interface	Load	Displacement
Identifier	String	String	String	String
Dimensionality	6{1A3}	€{0,1,2}	€{0,1,2,3}	€{0,1,2,3}
Geometry	**Vertex** Coordinates	Vertex Coordinates	Vertex Coordinates	Vertex Coordinates
Magnitude	X-Sect. Dimensions	X-Sect. Dimensions	Load Vector	Displ. Vector
Composition	€{R.G, Steel}	**0**	"Load"	"Displacement"
Stiffness	{E,I}	**0**	**0**	**0**
Arcs	List of Pointers	List of Pointers	List of Pointers	List of Pointers

Table 1: Data structure for adjacency structure nodes.

3.1.2 Primitive Elements of Adjacency Structures

Aggregations of atomic elements are organized into a hierarchy based on topological and behavioral distinctions. The hierarchy begins with lower level primitive elements which are, in turn, used to construct higher level system components. A *primitive element* consists of a specific number of nodes arranged in a specific topology. The exact geometry associated with each node in a primitive element is not specified by the component's definition; the geometry of each node is merely constrained to a specified relationship with other nodes in the primitive. A *system component* is composed of an indefinite, but finite, repetition of primitive elements also constrained to specified geometric relationships. A system component class is defined in terms of its constituent primitives. The system class retains the constraints of its primitives and subsystems, and uses additional constraints to define their composition into a system. A system class may be composed of a repetition of a single primitive element, or it may be composed of more than one type of primitive. We will call these two types of systems *uniform systems* and *composite systems,* respectively.

The atomic elements of adjacency structures represent the syntactic elements of all discrete static systems. Any classification of aggregations of nodes gives a domain-dependent relevance to certain subgraph structures composed of these nodes. Therefore, the definition of non-atomic elements and system components identifies relevant syntactic structures and attaches a particular semantic importance to them. For this reason, we will call the union of all the non-atomic components specified as being semantically relevant to the domain the *semantic templates* of that domain. The formal specification of these templates may form the basis of an algorithm for discovering the semantics of the purely syntactic overall adjacency structure. In this section we present an informal specification of two primitive elements from the domain of building structures.

Truss Panel. A truss panel is represented as a graph whose nodes form a cycle of length six composed of three pairs of alternating zero- and one-dimensional nodes. The elements define a single plane parallel to the orientation of the loads it resists (if these loads exist). Also, any applied loads present must be applied only at the zero-dimen$ional nodes.

Bent, A bent is represented as a graph whose nodes form a non-cyclic series of five nodes. The series is composed of three one-dimensional nodes each of which is separated by a zero-dimensional node. As in the truss panel, the elements are arranged in a single plane parallel td the orientation of the

loads the bent resists. However, loads may be applied to either the zero- or one-dimensional nodes. **Additional constraints on the relative angles and absolute** orientations of the one-dimensional nodes must **be included in the** graph template.

3*2 System Components

The expressive richness of the adjacency structure representation hierarchy may be defined by the union of all potential semantic templates over the domain of interest. A representation hierarchy should express the elements and systems relevant to the domain because these elements and systems are the building blocks of the parsing and generation processes. The composition operations described in this section facilitate the composition of systems from elements and the decomposition of systems into elements. A representation hierarchy for design must also be flexible enough to accommodate additions to the design vocabulary; it should allow the easy composition of new templates from lower-level components. A collection of composition operations over the set of atoms and primitives may be able to generate all relevant systems, but if the generation process is unguided it will also generate many irrelevant systems. The generation and even the "discovery" of new elements of the vocabulary is not the topic of this paper, but is an interesting part of future research with adjacency structures. The purpose of this paper is to provide a discussion of the adjacency structure representation and its use within a design process.

3.2.1 Template Composition Operations

There are three types operations useful for combining subgraphs into higher level graphs. One operation combines two separate graphs by unifying nodes which have the same dimensionality and the same location in both graphs. This is particularly useful for systems, such as *plane trusses,* which are defined in terms of components—triangular panels—which share atomic elements—a one-dimensional node and its two adjacent zero-dimensional nodes. A second operation connects two geometrically adjacent subgraphs without unifying nodes in the two subgraphs. This second operation connects geometrically distinct nodes in the two separate subgraphs, but because arcs denote physical adjacency in the graph structure and because an interface is a virtual object represented by a node, this operation adds a 'bridge' composed of two arcs separated by a node between the two subgraphs being connected. The dimensionality and geometry of the inserted node is the dimensionality and geometry of the intersection of the nodes being connected. This operation is useful for combining two systems which do not share components, e.g., two orthogonal shearwalls. The third type of operation embeds one graph within another by replacing arcs (and possibly nodes) of the host graph with new arcs into the immigrant graph. Embedding is useful for rearranging the components of **a** system being combined, e.g., when combining a frame and a shearwall by removing columns at the intersection and rcattaching the beams to the shearwall.

The three types of composition operations, joining two graphs $G\backslash$ and Gi to produce graph G3 are constructed by partitioning each graph $G_{,;z} \cdot = 1, 2$ into two sets of nodes, U_t which are involved in the composition operation, and t/| which are copied directly into G3. The arcs between nodes wholly in'£/) are also copied directly into G3. The difference between the operations lies in how the two sets of nodes £/,*, i = 1,2, and the arcs attendant to these nodes are transformed before being inserted into G3.

- **Merge:** unify one or more nodes in $G\backslash$ and G* The sets $U\!\!\rangle i$ = 1,2 are all nodes which are geometrically **and** dimcnsionally equivalent in graphs $G\backslash$ and G_2. For each pair of nodes to be unified, Mi € $U\backslash$ and **112** € I/2 from $G\backslash$ and G2 respectively, copy node $u\backslash$ into G3 and attach to this node all the arcs it possessed in $G\backslash$ plus all the arcs possessed by $u\%$ in G_2.

- **Abut:** $G\backslash$ and G_2 arc adjacent, but there are no geometrically equivalent nodes in $G\backslash$ and G_2. The sets *Ui, i* ss 1,2 are all those nodes in graph $G\backslash$ which are adjacent to nodes in graph G_2. Copy graphs $G\backslash$ and G_2 into G3 and add one or more 'bridges[9] between $U\backslash$ and £/$_2$. For each node $u\backslash$ *e* $U\backslash$ and *U2* € *U2* insert a node, u*, into G3 with the dimensionality and geometry of the intersection of $u\backslash$ and *U2f* and insert into G3 an arc connecting *Ub* to $u\backslash$ and an arc connecting u& to u_2.

- **Embed:** insert $G\backslash$ into the middle of G_2. There are no dimensionally and geometrically equivalent nodes in $G\backslash$ and G_2. The sets $U\!\!\rangle i$ = 1,2 are specified to achieve a particular behavior. $U\backslash$ may be transformed, possibly by removing some of its nodes, before being inserted into G3. The embedding begins by copying G2 into G3, removing all the arcs between nodes *Ui* now in G3, copying $G\backslash$ into G3, and adding arcs to $\{U\backslash$ U f/$_2\}$ as specified to achieve the desired behavior in the combined G_3.

These three types of composition operations are used to describe the transformation of graphs composing the representation hierarchy. The parsing and generation processes require these complex transformations because the graphs are not simply split when parsing or connected with a few arcs when generating. When parsing orthogonal frames into two sets of plane frames, for example, the single column at each frame intersections must be copied twice when forming the graphs representing the two frames. Therefore, the specification of the representation hierarchy requires an operational definition of the transformation of one level of templates into the templates of another level. These operations are used in describing the system components presented below.

3.2.2 **Planar** System Components of Adjacency Structures

Each of the components described in Section 3.1.2 is represented by a planar graph. The components are planar in graph-theoretic terms as well as representing physical objects which are relatively two-dimensional. This section describes groupings of these compound components into larger graphs which continue to be planar. The next section will extend these compound systems into non-planar graphs and into physical objects which are highly three-dimensional.

Unifonn compound systems are composed by repeating a single compound component within a set of prescribed geometric constraints. For example, a *truss,* in order to be a *plane truss,* must have each of its panels constrained to a single plane. In contrast, a *space truss* may be composed of the same type and number of compound components, but by using different geometric constraints the semantic template of a different system is specified.

Compound systems may be specified through the juxtaposition of multiple compound components or multiple unifonn systems within a set of prescribed geometric constraint. For example, a *braced frame* may be specified as a horizontally adjacent set of *plane frames* and vertical *plane trusses,* all in the same plane. In this way a hierarchy of higher-level relevant graph templates may be defined using a small number of

simple graphs which **may** be repeated an indefinite number of times during their instantiation in an adjacency structure. A few uniform and compound systems are informally specified below.

Plane Truss. A plane truss is represented as **a** graph repeating the *truss panel* component, i.e., each of whose nodes is **a** member of **a** minimal cycle of length six composed of three pairs of alternating zero- and one-dimensional nodes. Each panel is merged with at least one other panel, i.e. each cycle has at least one one-dimensional node and its two adjacent zero-dimensional nodes as members of one other cycle. The zero-dimensional elements (and therefore the one-dimensional elements also) are arranged in a single plane parallel to the orientation of the forces the truss resists, and the forces are applied only to the zero-dimensional elements.

Plane Frame. A plane frame is represented as a graph repeating the *bent* component, i.e., each of whose nodes is a member of a minimal cycle of length eight composed of four pairs of alternating zero- and nonzero-dimensional nodes. The composition of the plane frame is achieved by horizontally merging *bent* primitives and vertically abutting *bent* primitives. Additionally, the zero-dimensional elements are arranged in a single plane parallel to the orientation of the forces the frame resists.

Braced Frame. A braced frame is a compound system composed of one or more *plane frames* and one or more vertical *plane trusses* adjacent to each other and in the same plane. Each adjacency of the two system types is a merging along a vertical line of shared alternating zero- and one-dimensional nodes.

Shearwall. A shcarwall is represented as a vertical non-cyclic series of two-dimensional nodes. Each pair of two-dimensional nodes is mediated by a horizontal one-dimensional interface node.

Shearwall-Plane Frame. A shearwall-plane frame combination is represented as an embedding of a shear-wall within a plane frame. All one-dimensional nodes within the intersection of the shearwall and the plane frame are removed from the plane frame adjacency structure before arcs are inserted to combine the two systems.

3.2.3 Three-Dimensional System Components

The systems described above are planar compositions of smaller subgraphs. Higher level compositions of these planar systems can be defined to specify three-dimensional compound systems.[3] A few of the large number of three-dimensional systems are described in this section to explain their expression in the adjacency structure representation. First, gravity load resisting systems are discussed, separately from the lateral load resisting systems which they must eventually be unified with.

A bay of a flooring system is represented as a graph containing a two-dimensional node whose geometry is a single horizontal plane. Loads applied to the two-dimensional element are oriented normal to its plane. Various flooring types have different adjacency characteristics, and three examples are described below.

[3] **A frequent remark from experienced structural engineers, when discussing historical developments in structural engineering, has focused on developing the ability to think (compute) & visualize about structures in three dimensions. The specification of three-dimensional adjacency structures provides the ability to reason in three dimensions.**

Flat Plate. A bay of a flat plate flooring system is represented as a graph composed of a two-dimensional node and four pairs of one-dimensional nodes. A zero-dimensional node (denoting the interface) is adjacent to **the** two-dimensional element and each of the paired one-dimensional nodes as shown in Figure 2. Each one-dimensional node is oriented normal to the two-dimensional element.

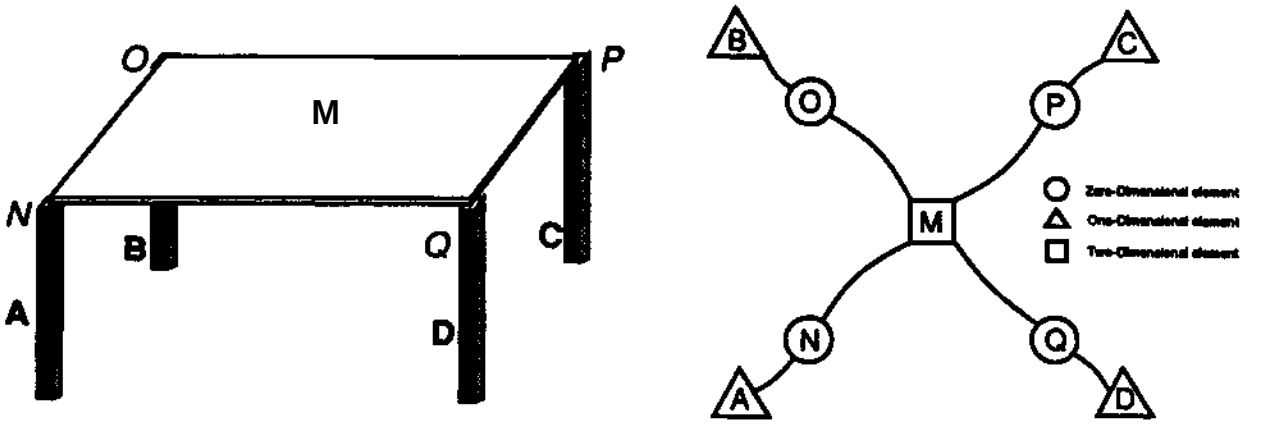
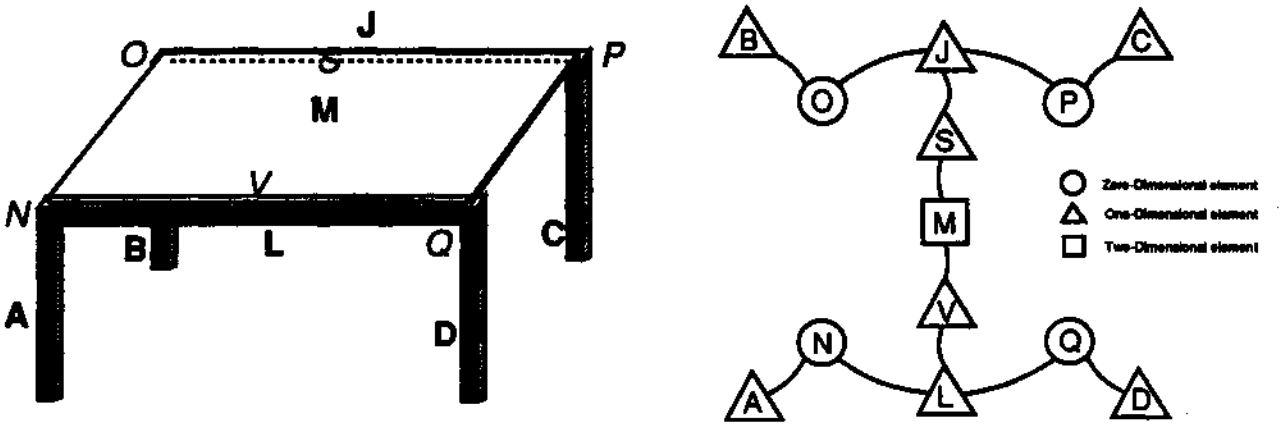


Figure 2: Pictorial and graph representation of flat plate flooring system.

One-way Slab. A one-way flat slab system is represented as a graph composed of a horizontal two-dimensional element abutted to *bent* components on alternating edges of the two-dimensional element. As shown in Rgure 3, the two-dimensional element and each of the *bent* components is mediated by an adjacent horizontal one-dimensional interface element introduced during the abut operation.



Rgure 3: Pictorial and graph representation of 1-way flat slab flooring system.

Two-way Slab, A two-way flat slab system is represented as a graph composed of a horizontal two-dimensional node abutted to horizontally merged *bent* components on each edge of the two-dimensional node. As shown in Rgure 4, the two-dimensional node and each of the *bent* components is mediated

by an adjacent horizontal one-dimensional interface node introduced during the abut operation. The **number of merged *bent* components equals the number of edges** of **the** horizontal two-dimensional **node. In addition to the** topological and orientation requirements listed above, a bound on the aspect ratio of the two-dimensional element of the two-way flat slab must be included in the graph template.

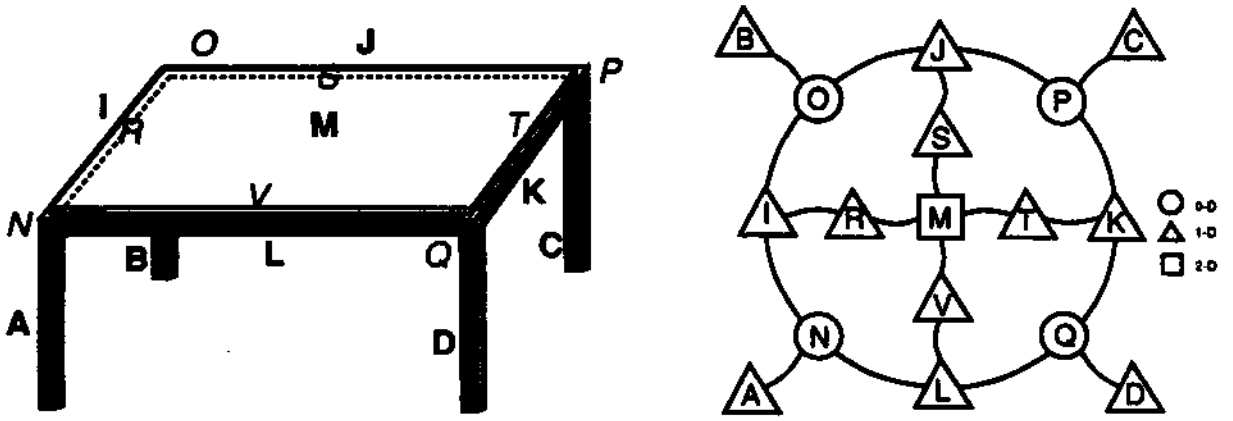


Figure 4: Pictorial and graph representation of 2-way flat slab flooring system.

At a higher level of the representation hierarchy, orthogonal plane frames remain a predominant three-dimensional framing system in current buildingpractice. This three-dimensional lateral-load resisting system is composed of two sets of plane frames. Each set of frames is oriented orthogonally to the other, and the two sets of frames share columns where they intersect. Non-orthogonal intersecting systems of plane frames may also be used in buildings which are based on other than orthogonal architectural grids. In contrast, the tube structural system can be viewed as a wrapping of a planar system about the three-dimensional envelope of the building volume.

A more complex composition in three dimensions is the hat truss, combining a number of planar trusses into a system which must also be defined in terms of its adjacency with other subsystems in the building and in term of its location within the overall system. A hat truss is a three-dimensional arrangement of plane trusses placed at the top **a** building. The function of the hat truss is to reduce the building's lateral deflection. It accomplishes this function through the behavior of tying together the building's core and perimeter frames, **and** thereby altering the shape of the building's deflection curve. The specialized location, topology and desired behavior of a hat truss lead to different types of constraints which must be incorporated in the semantic template of a hat truss, and which can sufficiently describe a hat truss without overburdening the process of matching on the overall adjacency structure. A belt truss fulfills a similar function through a similar behavior, but is located in a geometrically different relation to the overall structural system. The identification and distinction of these two systems is a severe measure of the type of representation and process which we are introducing.

Orthogonal Plane Frames. An orthogonal plane frame system is a non-planar graph composed of intersecting *plane frame* uniform systems, *braced frame* systems or*frame and shearwall* systems. Each frame

intersection is along a merged vertical series of paired zero- and vertically oriented one-dimensional nodes. Each shearwall intersection occurs along an abutted vertical edge of two shearwall panels, and each shearwall-frame intersection occurs by embedding a bent and shearwall primitive, removing a vertical one-dimensional node and reattaching its two adjacent zero-dimensional nodes to the vertical two-dimensional node representing the shearwall panel.

Framed Tube. A framed tube is a graph combining three or more *plane frame* uniform systems. Each successive *plane frame* is attached to the previous *plane frame* by merging a vertical series of paired zero- and one-dimensional nodes. Each of the one-dimensional nodes in the series is oriented vertically, and the last *plane frame* system merges to the first *plane frame* to complete a cycle. At a higher level of granularity the framed tube is a cycle of $2n$ nodes. Every other node represents a *plane frame* and its two adjacent nodes represent their shared, vertical interface.

33 Constraints Within Semantic Templates

The semantic templates specify a graph structure relevant to the domain. The topology of the graph clearly specifies how the nodes are configured, but the template nodes representing physical objects have only their dimensionality fixed by the template; the template does not assign the geometry, magnitude etc. of the constituent nodes. Previous sections have mentioned the constraints associated with components and systems. This section discusses the various types of constraints which may be associated with the semantic templates. Constraints are represented as part of the transformations which decompose (or compose) the templates during parsing (or generation.) The five type of constraints are geometric constraints, constraints on the location of the template, constraints on the association of node types within the template, functional constraints which are expressed in terms loads and behavioral constraints which relate applied loads to the resulting displacements.

- Geometric constraints:
  - Planarity: All the nodes of the graph, or of a specific subgraph, must reside within a single plane. The orientation of this plane may be specified as horizontal, vertical etc.
  - Symmetry: The geometry of a subgraph (each node and its arcs) must be equivalent to that of another node in the graph under a reflective transformation.
  - Aspect ratio:
    * Single node: The geometric proportions of an individual two- or three-dimensional node must be within a certain upper or lower bound, or within a particular range.
    * Complete graph: The geometric proportions of the convex hull of a semantic template must be within a certain range.
- Location within a larger system: A subgraph must be placed in a particular relation to the overall model in world coordinates.
- Associativity: The nodes of one subgraph must be joined to the nodes of a specific dimensionality, location or orientation in another subgraph.

- Functional constraints:
  - Dimensionality: the applied loads or displacements must be of a specific dimensionality; e.g., zero-dimensional or point loads applied to a truss template.
  - Location: the applied loads or displacements must be located at specific positions of the graph to which they are attached; e.g., point loads attached to zero-dimensional nodes of a truss template.
  - Orientation: the applied loads or displacements must be oriented in a specific direction relative to the graph to which they are attached; e.g., distributed loads oriented perpendicular to nonzero-dimensional nodes.
- Behavioral constraints:
  - Stiffness: the lateral deflection of a specific location of the template is directly proportional to the applied load and inversely proportional to the flexural stiffness of constituent members or compound components. This type of constraint can also implicitly describe the stiffness of joints in the composition.
  - Strength: the internal forces of constituent members or compound components in the template are related to the applied loads, and the material properties and cross section area of the members or components.

3.4 Translation and Discretization of the Geometric Model

Adjacency structures are a graphical representation of individual, physical objects and their adjacencies. Geometric modeling systems represent these objects in various ways. This section presents an algorithm for translating an arbitrary geometric modeling representation into a canonical overall adjacency structure. It is the responsibility of a translation function to be able to recognize, for a particular geometric modeling system, the characteristics which determine what data structures represent a physical object being modeled in that particular geometric modeling system. For example, when using a boundary representation geometric model, each node of the overall adjacency structure will represent one solid, or the discretization of one solid, in the geometric model. In a non-manifold geometric modeling system, in contrast, dangling faces and edges may also represent physical objects in the domain. Once the individual object's representations have been distinguished, these individual objects may need to be discretized into smaller objects before becoming part of the overall adjacency graph. One motivation for discretizing the model is to arrive at a scheme for producing a canonical representation of any model. For example, when representing a 40-story building, is a geometrically continuous column to be a 40-story tall object? We could use the maximal line method from shape grammars to produce a canonical representation. However, looking back at the truss example of Figure 1, with the maximal line method the top and bottom chords of a truss would each be modeled as one node rather than as separate nodes for each panel. Instead, we work under a minimal extent scheme, discretizing every modeled object at any location of intersection or adjacency with another modeled object.

The discretization of the geometric model is the translation of the geometric model into the overall adjacency structure. The formation of the overall adjacency structure begins by translating each object in the geometric model into one node in the overall adjacency structure. For each two objects, *u* and v,

which are adjacent in the geometric model, an arc is placed in the overall adjacency structure connecting the corresponding nodes u' and v\ Next, a nonregularized intersection is performed on each pair of adjacent nodes. Four results of the intersection are possible, each of which leads to a distinct operation in forming the overall adjacency structure. The four conditions and their results are listed below and shown pictorially in Figure 5.

1. The nonregularized intersection of it' and v' describes a point set which divides neither of the nodes into two distinct point sets. One interface node *W* with the dimensionality and geometry of the nonregularized intersection is introduced into the overall adjacency structure. The arc previously connecting *u'* and v' is replaced by an arc connecting *u'* and *w'* and an arc connecting v' and w\

2. The nonregularized intersection of *u'* and v' describes a point set which divides only one of the nodes, say u\ into two distinct point sets outside of the intersection. One interface node w' with the dimensionality and geometry of the nonregularized intersection is introduced into the overall adjacency structure, and *u'* is split into two nodes *s'* and *f.* The arc previously connecting *u'* and v' is replaced by an arc connecting v' and w', an arc connecting *s'* and *W,* and an arc connecting *t* and w'.

3. The nonregularized intersection of *vC* and v' describes a point set which divides both *u'* and v' into two distinct point sets apart from the intersection. One interface node w' with the dimensionality and geometry of the nonregularized intersection is introduced into the overall adjacency structure, and both *u'* and v' are split into two nodes *s'* and i\ and x^y and *y*\ respectively. The arc previously connecting u^f and v' is replaced by four arcs connecting *W* to each of the nodes *s*\$f_f$x? and *y*\

4. The nonregularized intersection of *u'* and v' describes a point set which divides one node, say u\ into two distinct point sets and intersects the other node, v\ within its boundary. This case requires a two-stage process: one interface node w" with the dimensionality and geometry of the nonregularized intersection is introduced into the overall adjacency structure, and *u'* is split into two nodes s^f and *t'* which are also inserted into the overall adjacency structure. However, v' temporarily remains one node. When all objects adjacent to *V* are processed v' may be divided into multiple nodes as follows. All interface nodes within or adjacent to v' are collected into a set and ordered according to their geometric location. A temporary graph is formed from these nodes with each node connected to its nearest neighbors.[4] Finally v' is divided along the geometric lines connecting each node in the temporary graph to its neighboring nodes. The subdivisions of *V* arc inserted into the overall adjacency structure separated by appropriate interface nodes. The arc previously connecting *u'* and v' is replaced by two arcs connecting connecting w" to the nodes *s'* and *t*\ and an arc connecting w" to each adjacent node resulting from the subdivision of v\

   When translating the geometric model of a flooring system for example, each *u'* will be a one-dimensional node oriented perpendicular to the two-dimensional node v' (columns perpendicular to the slab). Each w" will be a zero-dimensional node located in the plane of v\ If the one-dimensional

[4]Thc assistance of the user is required to determine the precise meaning of "nearest neighbors" for many interesting problems. Therefore, we do not complete the specification of this part of the case.

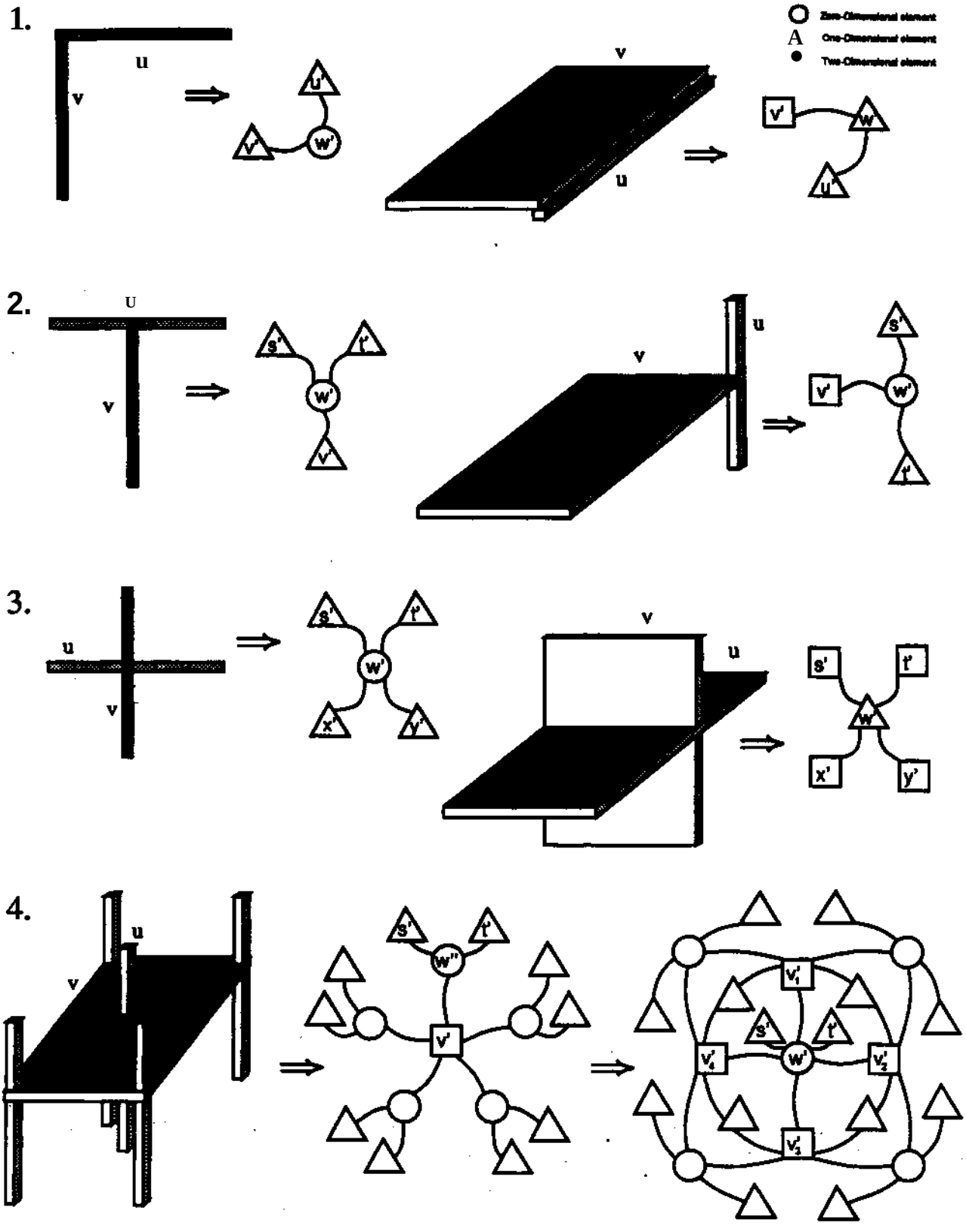


Figure 5: Results of nonrcgularized intersection of adjacent geometric objects.

nodes arc arranged according to an orthogonal grid the interface nodes on the corners of v' will be connected to two other interface nodes* whereas other nodes on the boundary of v' will be connected to three other interface nodes, and nodes on the interior of v' will be connected to four other interface nodes in the temporary graph. Each subdivision of v' is a two-dimensional node adjacent to four zero-dimensional interface nodes and adjacent to multiple one-dimensional interface nodes separating it from the other two-dimensional subdivision nodes.

4 Semantic Interpretation of Syntactic Structure

The formation of the overall adjacency structure from a geometric model is a translation of one model's syntax into the syntax of another modeling system. The expense of this translation is only worthwhile if the new modeling system is more useful for some particular purpose. In this section we present how the adjacency structure representation of a geometric model may be used to discover both intended and emergent semantics within the syntactic representation. This discovery process is the parsing mode mentioned at the beginning of this paper.

As intimated above, the parsing of an existing representation is basically a subgraph matching problem. General subgraph matching is regarded to be an NP-complete problem [Aho 74]. That is, no algorithm, bounded by polynomial time, is known for deciding whether a graph *G*\ contains a subgraph G2. The inclusion of geometric information within the adjacency graph, in addition to providing the ability to reason about the system's behavior and functionality, alleviates much of the matching complexity attendant to the general subgraph matching problem [Schnitzler 82]. By incorporating node dimensionality and geometric information into the representation the matching is made more specific because more information is being matched upon.

The parsing of an overall adjacency structure in terms of semantic templates is an inductive procedure, translating a specific syntactic structure into a set of semantic templates general to the domain. To illustrate this procedure let us assume that we have translated the geometric model representing the structural system of a building into the overall adjacency structure as described in Section 3.4. Beginning from the highest level of the adjacency structure taxonomy, an attempt is made to unify semantic templates with the overall adjacency structure. This is a hierarchical process because the high-level templates are defined in terms of lower level templates. In structural engineering there is a set of three-dimensional systems which incorporate both a lateral load resisting system and, with the addition of floor slabs, a gravity load resisting system. This set includes the varieties of tube structures and orthogonal framing systems. Therefore, the first templates which may be unified with the overall adjacency structure are these three-dimensional system templates. No more than one semantic template should match on a given overall adjacency structure during parsing if the templates are defined in exclusive terms. For example, a framed tube structure should not be described as an orthogonal rigid frame structure even though the framed tube may consist of four rigid frames oriented orthogonally.

After the highest level adjacency structure has been unified with the overall adjacency structure, matches are sought for any remaining subgraphs. These ancillary adjacency structures must be compatible with the previously unified adjacency structures. Adjacency structure compatibility entails being able to combine

multiple subgraphs through the available composition operation described in Section 3.2.1 while satisfying the applicable constraints on each adjacency structure.[5] The need for ancillary adjacency structures will be common for building with non-rectangular plans and massings, for example.

The purely syntactic matching process can discover emergent systems because it inspects the graph structure to find what is contained in the model* not what is said to be in the model. For example, if the structural system is generated as a set of plane frames in the *x-z* plane, and then each frame is connected by beams (at the joints of each frame) in the *y-z* plane the *plane frame* template will match on frames in both the *x-z* plane and in the y-z plane, thereby discovering that there is an orthogonal plane frame system, i.e., plane frames exist in both directions.

Additionally, this representation and parsing process may be used to confirm intended syntactic compositions and to evaluate the condition of existing syntactic compositions. That is, the matching process may also be utilized to find the presence of extraneous elements or the omission of necessary elements. After the structural system has been parsed, the resulting definition of the system in terms of semantic templates may be compared to the structural system graph translated from the geometric model. If a boolean difference between the overall adjacency structure and the union of semantic templates parsed from the overall adjacency structure leaves any structural elements remaining, these remaining elements may be said not to participate in the structural system described by the semantic templates. These elements which are not included in the semantic templates are extraneous to the system defined by the templates and possibly may be removed from the design.

Alternately, the initial architectural definition of the building contains a set of components which may form an intentionally incomplete structural system. For example, the floor slab and column placements may be specified by the architectural design. However, this design is not meant to rule out the use of beams in the structural design or to preclude the use of a one-way or two-way fiat slab gravity system. Parsing the overall adjacency structure finds that there are no lateral load resisting system templates that will match on the overall adjacency structure, but that many templates may be unified with it if the design process is shifted to the generation of a lateral load resisting system.

These three results of parsing can be used to summarize the syntactic condition of the design. The overall adjacency structure may be completely parsed into atomic elements thereby describing the overall adjacency structure in terms of the templates at successive levels of decomposition. This signifies a syntactically complete design. Alternately, the overall adjacency structure may be decomposed, but the decomposition leaves elements that are not parsed out of the overall adjacency structure. This signifies an intentionally or unintentionally redundant design. Finally, the overall adjacency structure may be incompletely decomposed, halting at the system level unable to match on the existing adjacency structure. This signifies a syntactically incomplete design that requires additional elements for completion.

[5]The recurrent combining of adjacency structures not contained as single compound components in the representation hierarchy during the parsing process may afford one method of learning or "chunking" new or appropriate system components.

5 Adjacency Structures in A Design Process

The various design processes in which an architect and a structural engineer interact may be viewed along the two axes of phase and domain, shown in Figure 6. The predominant architect-engineer design process begins with an architectural parsing of the client's design brief, then proceeds with an architectural generative phase. Next, structural design begins with a parsing of the architectural design to find pertinent geometric and functional information, then proceeds through a structural generative phase. In this design process, the architectural phase sets the geometry of the building envelope and constrains the location of any internal elements which the structural engineer may introduce. The role of the structural engineer during preliminary design becomes one of proposing a small number of "good" structural systems which can support, and be accommodated within, the architecturally subdivided envelope. This section presents the use of adjacency structures in the transformation of functional requirements into a structural description.

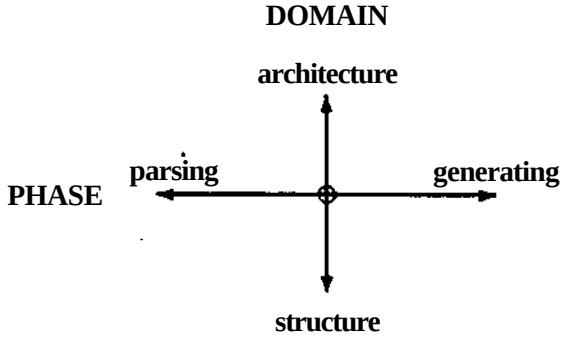


Figure 6: Design as processes along Domain - Phase axes.

The mapping from the functional requirements of a design to instances of design structure which satisfy those requirements can be performed as a successive refinement of elements of a functional hierarchy. A rcprcsentation for such a process must include attributes which can explicitly rcprcsent at least function, if not behavior. The inclusion in the semantic templates of nodes representing loads and displacements can represent both given and propagated functional requirements such as applied loads or imposed displacements. Behavior, in the form of the flexural stiffness and axial forces of the system, is represented as constraints on the system components of the rcprcsentation hierarchy. Thus, a uniform rcprcsentation is used to model function and structure, providing a transparent method of propagating functional requirements within a partial design solution.

To illustrate the use of adjacency structures in a design process let us assume that we have a geometric model representing a preliminary architectural design of a building. The architectural description includes a definition of the building's envelope as the external surface of the geometric model, plus any interior geometric entities representing the location of partitions such as the service core, floors and permanent internal walls. The geometric model of the building is translated into the overall adjacency structure according to the minimal extent principle described in Section 3.4. Through design standards or experience, lateral and gravity loads arc specified for the building form as a function of height and occupancy type. These lateral and gravity loads arc added to the overall adjacency structure as *load* nodes. Also through design standards or experience, allowable deflections are specified as a function of the applied loads. Together, the

architectural envelope and internal elements, and the applied loads and allowable deflections form the specific functional requirements for the structural design. The structural design phase consists of instantiating sets of subgraphs which together satisfy the specific functional requirements for an individual building, along with the general requirements for all buildings such as constraints on member forces.

Design generation, like the parsing process, begins by seeking matches for the semantic templates of the representation hierarchy giving primacy to three-dimensional systems. The control mechanism for the generation process differs from that of the control of the parsing process in that multiple instantiations are sought which can satisfy the functional requirements; the control of the generation process should branch the single set of design requirements into multiple potential solutions. The complete instantiation of semantic templates to form an adjacency structure which satisfies the functional requirements may be divided into two phases: topology and parameterization. In the first phase the building envelope is populated with specific types and numbers of semantic templates. In the second phase the (primarily geometric) unassigned data fields of the nodes composing the templates are assigned values. An attempt is made to topologically instantiate the highest level semantic templates possible through matching on the object and load nodes in the overall adjacency structure while satisfying the templates' constraints. The load nodes are propagated subject to the constraints on the allowable dimensionality, location and orientation for load nodes on the semantic templates being instantiated. Originally the lateral and gravity loads of the functional requirements are distributed loads inserted into the overall adjacency structure as two-dimensional nodes. The lateral loads must be propagated as one-dimensional nodes when instantiating such templates as frames or framed tubes, as zero-dimensional nodes when instantiating truss templates or remaining as two-dimensional nodes when instantiating shearwall templates.

Another fundamental constraint on each high-level system during topological instantiation is that the system must be composed of an integer repetition of its constituent templates. For example, a *plane frame* template must be composed of an integer number of horizontally merged *bent* templates and an integer number of vertically abutted *bent* templates. Thus, topological instantiation involves a determination of the dimension of the target region[6] for the potential instantiation of a template and a comparison of the template's application limits to arrive at possible dimensions for the region's division. The order of these two steps is dependent of the design process in which it is used. In the design process described at the beginning of this section, it is appropriate to first determine the acceptable range of the subdivision dimensions. For example, if an office building is being designed with a 45 foot core-to-perimeter dimension which has no permanent interior walls besides the core, is it acceptable to place a column between the core and the perimeter? If it is not architecturally acceptable to do so, then this constraint must be considered during the instantiation of any internal frame or flooring system. This suggests the need for an interactive ability during the constraint satisfaction necessary in the topological instantiation of semantic templates; their own limits of application are underconstrained. When the number of component templates composing the specific system is determined the topological instantiation itself can be accomplished by generating the proper number of subgraphs which realize a compound template, assigning values, to the appropriate data fields in each node of the template, and embedding it in the overall adjacency structure. After the first template system is

[6] A region is a general geometric space in R^1, Z^2 or R^3 i.e. the division of a region may be the division of a beam, floor plan or architectural volume.

instantiated and added to the overall adjacency model, subsequent instantiations are also constrained to accommodate the existing adjacency structures in the model through the composition operations presented earlier. For example, if the lateral load system is satisfied first by an orthogonal rigid frame, the gravity load system is constrained to using the existing beams and columns for instantiating the gravity system templates.

The parametric instantiation is primarily concerned with the defining of member cross-sections. In order to accomplish this parameterization some level of analysis is needed. One advantage of the node and system representation we have described is its ease of translation into the matrix methods of analysis. Each member node, if it had its cross-section geometry defined, would contain enough information to form the member stiffness matrix. The overall adjacency structure would then contain enough information to compose the element stiffness matrices into the global stiffness matrix. Then, the product of the inverse global stiffness matrix (the flexibility matrix) and the force vector results in the deflection vector. However, the topological instantiation does not provide the information needed to complete the element stiffness matrices; it only provides enough information to compose the global matrix from defined element stiffness matrices. Thus, one possible design process is as follows:

1. The topological instantiation defines the length of one-dimensional elements and the breadth and depth of two-dimensional elements when the building envelope is subdivided into an integer number of templates.

2. The relative stiffness of all member nodes is defined. This relative stiffness is used to define element stiffness matrices for each member node in the overall adjacency structure as a function of its moment of inertia / and the modulus of elasticity £ of a material.

3. The relative stiffness matrix of each element along with the defined topology is used to construct the global stiffness matrix.

4. The deflections of the building structure are determined as a function of the relative stiffness and the applied loads.

5. The stiffness is assigned to limit the deflections to an allowable amount. The assigned stiffness allows the back calculation of the member stiffnesses and, therefore, the defining of their cross-sections.

The instantiation process continues until the overall adjacency structure becomes a complete connected graph, that is, when the applied propagated loads have been connected to system templates for resisting these loads, and when the system templates have been completely instantiated down to their atomic elements. In this way we perform the generative mapping from functional requirements, stated in terms of the architectural form and the applied loads, through the behavior of load propagation and flexural stiffness to derive the structure of a design solution represented as a network of adjacent members.

A design process must also include provisions for changing the model as new information is introduced or existing information is altered. We have discussed the introduction of adjacency structures into the design model, but must also consider the requirements of editing existing adjacency structures. As in their introduction, the editing of adjacency structures may be divided into topological alterations and parametric alterations. A graph grammar may be used for topological edits such as embedding a shearwall or braced

frame template into an existing plane frame system within the overall adjacency structure. On the other hand, the object oriented programming technique of *methods* which operate on a restricted set of abstract data types appears more appropriate for the parametric editing of a specific system component. Instantiation and editing, thus, form the basic operations of the design process with adjacency structures.

6 Conclusion

This research is motivated by the belief that a hierarchical organization of design components reflects the way human designers think about building structures, and that basing a graph representation on the adjacencies of physical components reflects both the way building structures are built and the way they behave as systems. The representation presented in this paper provides a more convenient, higher-level means of operating on a system of geometric objects than such representations as the split-edge data structure used by boundary representation geometric modeling. At the same time, the graph representation of adjacency structures provides a more explicitly system-oriented representation than objects, frames or prototypes for a domain in which a design is composed of a large number of highly interconnected, primarily geometric objects. For these reasons, we feel that adjacency structures provide an intuitive and convenient representation for design generation and evaluation, a representation which captures the essential geometric and systematic nature of discrete static systems.

The typical result of a design process is a description of the physical shape and material composition of an artifact delivered in response to a set of functional requirements. The drawings and specifications delivered by an architectural/engineering firm as the design of a building are a purely syntactic description of the building. However, the design mandate for the artifact was given in terms of the semantics of the design; the functions the proposed artifact must satisfy. The parallel between the form—function dichotomy and the syntax—semantics dichotomy is apparent in a domain where a design may be specified purely in terms of a description of the artifact's shape and material. The syntax of the design describes the form of the proposed artifact whereas the semantics of the design describes the functional requirements and the behavioral expectations of the proposed artifact.

We have presented a representation which incorporates syntactic aspects of a design in terms of adjacent physical objects with semantic aspects of the domain in terms of systems, loads and displacements. We have also presented methods for utilizing this representation in the design process as bidirectional mappings between the functional, behavioral and structural views of a design description. These two mapping directions support design through mapping from function to structure (synthesizing potential solutions), through mapping from structure to behavior (qualitatively analyzing potential solutions), and through mapping from behavior to function (evaluating a potential solution.) The adjacency structure representation also provides an important link between design visualization, provided by geometric modeling, and design analysis provided by the matrix methods. Through these mappings, this representation and design process supports a function-to-structure design method for the domain of discrete static systems.

References

[Aho74] A, V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts, 1974.

[Bunkc79] H. Bunke. "Programmed graph grammars/' In *Graph-Grammars and Their Application to Computer Science and Biology,* volume 73 of *Lecture Notes in Computer Science and Biology,* pages 155-166, Berlin, Springer- Verlag, 1979.

[Cortes 89] L. A. P. Cortes. "Graflog: A theory of semantics for graphics with applications to human-computer interaction and CAD systems/' PhD thesis, University of Edinburgh, 1989.

[Culik73] K. I. Culik. "A model for the formal definition of computer languages.'[9] *International Journal of Computer Mathematics,* A(3):315-345,1973.

[Deransart 88] P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars: Definitions, Systems and Bibliography,* volume 323 *of Lecture Notes in Computer Science.* Springer-Veiiag, Berlin, 1988.

[Eastman 91] C. M. Eastman, A. H. Bond, and S. C. Chase. "A formal approach for product model information." *Research in Engineering Design,* 2:65-80,1991.

[Ehrig 86] H. Ehrig. 'Tutorial introduction to the algebraic theory of graph grammars." In *Graph-Grammars and Their Application to Computer Science,* Lecture Notes in Computer Science, Berlin, Springer-Verlag, 1986.

[Finger 89] S. Finger and J. Rinderle. "A transformational approach to mechanical design using a bond graph grammar." In *Proceedings, Design Theory and Methodology Conference,* ASME, Montreal, September 1989.

[Hemming 86a] U. Flemming. "On the representation and generation of loosely packed arrangement of rectangles." *Environment and Planning B: Planning and Design,* 13:189-205,1986.

[Hemming 86b] U. Flemming, M. Rychener, R. Coyne, and T. Glavin. "A generative expert system for the design of building layouts, version 1." Progress report, Center for Art and Technology, Carnegie Mellon University, June 1986.

[Gero 87] J. Gero. "Prototypes: A new schema for knowledge-based design." Working paper, Architectural Computing Unit, University of Sydney, 1987.

[Gero 88] J. Gero, M. Maher, and W. Zhang. "Chunking structural design knowledge as prototypes." Technical Report 12-25-88, Engineering Design Research Center, Carnegie-Mellon University, 1988.

[Gero 90] J. Gero. "Design prototypes: A* knowledge representation schema for design." *AI Magazine,* ll(4):26-36, Winter 1990.

[Gcro91] J. Gere, H. Lee, and K. Tham. "Behaviour A link between function and structure in design." In *IntCAD[f]9h WTP* Working Group 5.2,Columbus, OH, October 1991.

[Kamopp68] D. Kamopp and R. C. Rosenbeig. *Analysis and simulation of multiport systems; the bond graph approach to physical system dynamics.* M.I.T. Press, 1968.

[Knuth68] D. E. Knuth. "Semantics of context-free languages." *Mathematical Systems Theory,* 2(2): 127-145,1968. Corrections in MST Vol. 5 No. 1 pp. 95-96.

[Krishnamurti 78] R. Krishnamurti and R H. O. Roe. "Algorithmic aspects of plan generation and enumeration." *Environment and Planning B: Planning and Design,* 5:157-177,1978.

[Mitchell 76] W. Mitchell, J. Steadman, and R. Liggett. "Synthesis and optimization of small rectangular floor plans." *Environment and Planning* B_9 3:37 - 70,1976.

[Nagl79] M. Nagl. "A tutorial and bibliographic survey on graph grammars." In *Graph-Grammars and Their Application to Computer Science and Biology*\ volume 73 of *Lecture Notes in Computer Science,* pages 70-126, Berlin, Springer-Verlag, 1979.

[Nagl 86] M. Nagl. "Set theoretic approaches to graph grammars." In *Graph-Grammars and Their Application to Computer Science,* Lecture Notes in Computer Science, Berlin, Springer- Verlag, 1986.

[Payntcr61] H. M. Paynter. *Analysis and Design of Engineering Systems.* MIT Press, 1961.

[Pinilla89] J. Pinilla, S. Finger, and F. Prinz. "Shape feature and recognition using an augmented topology graph grammar." In *Proceedings of the 1989 NSF Engineering Design Research Conference,* Amherst, MA, 1989.

[Rinderic91] J. R. Rinderle. "Grammatical approaches to engineering design 2: Melding configuration and parametric design using attribute grammars." Technical Report 24-53-91, Engineering Design Research Center, Carnegie Mellon University, 1991.

[Rozenberg 86] G. Rozenberg. "An introduction to the NLC way of rewriting graphs." In *Graph-Grammars and Their Application to Computer Science,* Lecture Notes in Computer Science, Berlin, Springer-Verlag, 1986.

[Schneider 75] H. J. Schneider. "Syntax-directed description of incremental compilers." In *Graph-Grammars and Their Application to Computer Science,* volume 26 of *Lecture Notes in Computer Science*^ pages 192-201, Berlin, Springer-Verlag, 1975.

[Schnitzler82] M. Schnitzler. "The isomorphism problem is polynomially solvable for certain graph languages." In H. Ehrig, M. Nagl, and G. Rozenbeig, editors, *Graph-Grammars and Their Application to Computer Science,* Lecture Notes in Computer Science, pages 369-379, Berlin, Springer-Verlag, 1982.

[Stiny 81] G. Stiny.[4<] **A note** on **the** description of designs." *Environment and Planning B: Planning and Design,* 8:257-267,1981.

[Ulrich 87] K, Ulrich and W. Seering. "A computational approach to conceptual design." In *Proceedings* ***of the International Conference on Engineering Design,*** **August 1987.**