

An Axiomatic Theory of Engineering Design Information

F. A. Salustri*

R. D. Venter

Department of Mechanical Engineering
University of Toronto

Published in:

Engineering with Computers, 8(4):197-211, 1992.

Abstract

Recent research in design theory has sought to formalize the engineering design process without particular concern for the paradigm used to model design information. The authors propose that no correct formalization of the design process can be achieved without first formalizing the semantics of the information used in the process. To this end, the authors present a new formal theory of design information. The theory, called the Hybrid Model, is an extended form of axiomatic set theory, and relies on it for consistency and logical rigor. The theory is stated as a collection of axioms, using a standard logic notation. Design entities are modeled by formal units called *objects*. Generalized functions and relations are used to formalize important ordering schemes and abstraction mechanisms relevant to design, including classification by structure and by function, aggregation, specialization and generalization. The hybrid model is meant not only to aid in the study of the design process itself, but also to improve communications between designers, assist standardization of design specifications, and develop new, powerful software tools to aid the designer in his work.

1 INTRODUCTION

Design Theory and Methodology (hereunder abbreviated to DTM) is the branch of engineering research devoted to the formalization of engineering design and the methods used by designers. Efforts in this field have rightly depended on the use of computers for their capacity to store information accurately and to calculate and maintain the complex relationships that exist between data. As the world economy continues to move from a product and service base to an information base, information management issues in design will become more important.

DTM research in the recent past has examined the use of databases, application-specific languages and expert systems, as well as more theoretical studies in such areas as constraint satisfaction and symbolic computation. These tools have been applied with varying degrees of success to component assembly [1], design exploration [2, 3], solid modeling [4], finite element analysis [5], etc. The unique nature of design suggests that generalized information management approaches will not necessarily support all of its aspects [6, 7].

Currently, however, there is growing concern regarding the *semantics* of engineering design. Many recent research efforts have met with limited success because not enough is understood about the *meaning* of the tools and information we use. The understanding we do have tends to be empirical and intuitive [8, 9] and its organization is neither particularly structured nor logical. In response to this, researchers have begun to backtrack, seeking a return to sound, logical first principles in design. Two notable examples of this trend are [10, 11]. The notion of the existence of formal first principles for design has guided the authors' work presented herein as well.

*Corresponding author: University of Toronto, Department of Mechanical Engineering, 5 King's College Road, Toronto, Ontario, Canada M5S 1A4

The importance of the *organization* of design information cannot be overemphasized. This issue is strongly tied to the search for semantic formalization. To organize information means to order it. The imposition of order on information is identical to the extraction of meaning from it, and makes *explicit* such information as would otherwise be *implicit* only. Increasing the amount of explicit information present in a collection of data decreases the amount of *interpretation* that must be performed to extract its semantics. Therefore, the study of organizational schemes for design information is synonymous with the study of its semantics. The theme of organization pervades the authors' work.

Naturally, no discussion about design information can be carried out without some reference to the design process, but the authors will demonstrate that it is both reasonable and advantageous to separate information about a design artifact from the actions carried out on or with this information. We assert that it is essential to understand what forms of information are available before any meaningful discussion regarding design *processes* can occur. Thus, the authors' current work deals specifically and only with design information. The design process will be discussed only insofar as to define the design information management problem. Issues such as concurrent design are not addressed because they are aspects specific to the design process; that is, they affect how information is manipulated, but not the information itself. The reasoning that led us to this observation is discussed in section 2.

The search for a formalization for engineering design must necessarily be conducted in a logical, scientific and internally consistent manner [11]. For this purpose, the authors have chosen axiomatic set theory, and discuss it briefly in section 3.

The rest of this document is devoted to an examination of design information with the aim of formalizing its structure. This is done by proposing a series of axioms, theorems and definitions that rely on axiomatic set theory for internal consistency. The resulting model is termed by the authors the "hybrid model" (abbreviated to HM) of design information.

Finally, we present some of our conclusions based on the work done to date and a brief overview of future directions.

2 MODELING THE DESIGN PROCESS

It has already been suggested [12] that design information can be considered separate from the engineering design process. The authors agree: the nature and complexity of design problems – and of the methodologies for their solution – are such that this division is both reasonable and advantageous. In addition, we believe that a good understanding of the design information must *precede* any real understanding of the design process itself. In this section, we present our model of the design process with the intention of supporting the hypothesis regarding the separation of design information and processes.

At its current stage of development, the authors' work does not require a particularly detailed model of the design process; yet, some idea of how design proceeds in general must exist to provide a reference with respect to which HM can be developed. It also brings to light several important aspects of the design process that affected the development of HM.

The model regards the design process from a functional point of view, and we refer to it simply as the "functional model"; it is quite similar to an earlier model, suggested by the authors in [13], called the state transition model; however, we have found the functional model to be more elegant than its predecessor.

We begin by making the relatively trivial statement of a generic mathematical function, namely:

$$y = f(x)$$

Here, f is some function that maps an input value represented by the variable x to some output value represented by the variable y ; indeed, y and $f(x)$ are identical. Now, from the point of view of design, we might rewrite this equation as:

$$S = d(P)$$

P represents a design problem, S its solution and d the design process. We may state this in words as: "There is a design process that operates on a particular design problem and results in a corresponding design as a solution".

This is not an unreasonable statement to make, and though it may still appear trivial, it does carry some important implications.

Clearly, the solution depends on the problem (the output is the dependent variable). Also, as stated above, S and $d(P)$ are identical.

From a purely mathematical point of view, one may be inclined to stop here. But there is more than one way to design a certain entity. That is, given a particular design problem, there may be more than one design process d that can provide equally acceptable solutions. Selection of a design process depends at least in part on the kinds of information about the problem that are available to the designer. (It would also depend on optimization considerations as well as the more intangible preferences and judgments of the designer.)

If the input is badly or incorrectly defined or specified, then selection of an appropriate solution function may be difficult or impossible. That is, if the design problem is badly stated, then the selection of a design process is prone to error (since, as is indicated above, the design process can be considered to be dependent on the problem), and thus affect the confidence of our solution. Therefore, the problem must be *clearly understood* and *precisely defined before* a solution process can be selected.

There is another issue that is an essential component of almost every non-trivial design task: iteration. That is, a (possibly dynamically changing) design process will be applied iteratively to a design problem in order to reach a final solution. We can represent this in our mathematical notation by:

$$S_{i+1} = d(S_i + P)$$

For each iteration $i + 1$, the design process d is applied to the problem *plus* the solution, such as it exists, at iteration i . Put another way, our solution at iteration $i + 1$ is based on both the problem *and* the i th solution. Without including the solution at iteration i in the argument to the design function, convergence would never occur. So, at each iteration is a design cycle, the existing – though possibly incomplete and/or incorrect – solution is used to drive the next iteration of the design cycle. The essential observation here is the “superposition” of the problem with the i th solution: the problem and the solution must be representable in a compatible way or the iteration process cannot proceed.

Therefore, the functional model provides us with two important insights into the requirements that must be met by a formal system for design information:

- a formal understanding of design state information is necessary *before* the design process can be successfully formalized to any significant degree;
- the organization of information is relevant both for the problem definition and the solution and *any theory of this information must be unified over both problem and solution domains*.

Other researchers have used a functional perspective to model the design process. Suh [10] differentiates from the outset between the “functional requirements” that define the design problem and the “design parameters” (which are *not* functional, but physical in nature) that define the solution. In [14], Yoshikawa defines two separate spaces – a function space and an attribute space – and each of the design problem and solution are defined in terms of one of these spaces only. These approaches, among others, are similar in that both consider a design artifact (as represented by Suh’s design parameters or Yoshikawa’s attribute space) as a separate entity from the problem that caused it to be designed (the functional requirements or space).

Such a separation of functional and physical domains is beneficial from a conceptual point of view, permitting modularization of the task into smaller segments that can be studied individually. However, it can also lead to a divergence at the theoretic level that will prevent final integration of these domains into a single, global theory.

The authors’ recognize that in an iterative process such as design, the *cumulative* information generated from the iteration is an essential component of finding a correct solution. In order to merge the accumulated information with the design problem for the iteration to continue, a *unified* representation of both problem and solution must exist. Our model of design maintains the integrity of problem and solution specification while dividing the problem along a different and, in our opinion, more important boundary, that between static, passive information and dynamic, active functions that *transform* the information. As will be explained in the sections to follow, we believe that this approach can lead to a clearer, more rigorous and complete theory of design.

Another contribution that should be mentioned is that of Fauvel [15], wherein the relation

$$(\text{Activity } (n), \text{ Embodiment } (n)) \rightarrow \text{Activity } (n+1)$$

is seen as representative of the design process. An *Activity* (*i*) is some component process of the overall design process and an *Embodiment* (*i*) is the physical manifestation of the result of the completion of an activity. It is interesting to note the shift in point of view between our model and that of Fauvel. His is based on the notion that given some initial design activity, the results of that activity drive the selection and execution of other activities. Ours is based on the notion that an initial embodiment (in Fauvel's terms) drives the selection and execution of activities that lead to other embodiments.

The authors maintain our model as presented above because of the observation that reliable, accurate and usable information must exist *prior* to the selection of any processes meant to act on this information; that is, we believe the emphasis should be placed on information as the driving force behind a design enterprise.

Fauvel reasons in detail on the role of various kinds of activities that are relevant to design without dwelling on the nature of the embodiments. His results are quite clean and elegant; this encourages the authors to believe that the separation of design information from design actions is not only appropriate, but necessary if DTM is ever to meet with success.

3 BASIC STRUCTURES AND CONCEPTS

3.1 Basic Aim of HM

3.1.1 A Prescriptive, Axiomatic Approach

The basic aim of HM is to provide a prescriptive, axiomatic theory of the information present during the course of a design. Naturally, only information relevant to a particular design task is considered, thus restricting the universe of discourse significantly. This restriction plays an important role in the development of HM; this is discussed below.

HM is *prescriptive* in that it prescribes how information should be specified for a design task. We do not use the term to indicate an attempt to reduce the problem to a purely mechanical one requiring little or no human intervention. Given the relative immaturity of DTM as a science, the authors believe that the best results can be achieved by a *symbiotic* relationship between the designer's innate capabilities (including such intangibles as judgment, creativity and intuition) and some more formal, logical system.

We do use the term *prescriptive* to indicate that HM is a system that lies outside the actual thought processes of the designer; i.e. it lies within a theoretical, logical domain. The authors see this as a more reasonable objective than that of the "descriptive" school [16], which seeks to quantify and formalize the actual cognitive functions (i.e. mental content and processes) of the designer.

The theory is "axiomatic" in that it relies on axiomatic set theory as its foundation. Our initial attempts sought a formalization based on existing information management paradigms (object orientation and hypertext in particular), but we found the lack of existing formalization in these fields unsatisfying. For example, in [7], object orientation is referred to more as a philosophy or point of view than an actual formal paradigm; the status of hypertext is even more tenuous [17]. It became necessary to return to more basic first principles, and it was during our study of symbolic logic that axiomatic set theory presented itself as a very viable system upon which to base HM.

Axiomatic set theory has taken on various forms [18, 19, 20] but every form is based on the classical theory developed by Zermelo and Fraenkel [18] and which is generally referred to as ZF set theory, or just ZF. We will, for brevity's sake, adopt this convention also.

ZF deals with groups of completely general entities; a group of entities is called a *set*. The theory formalizes the nature of sets to such a degree as to permit the derivation of almost all the classical branches of mathematics and logic [19]. The most interesting implication of set theory as far as the authors are concerned regards consistency of theories that are supersets of classical axiomatic set theory. In [18], it is proved that any axiom system that can be rewritten in terms

of ZF without introducing any new atomic statements, quantifiers or connectives is consistent. As will be seen, this criterion is satisfied by HM. This means that we know at once that HM is a consistent axiom system.

3.1.2 Universe of Discourse and Design Entities

The term *universe of discourse* denotes the overall domain within which all interesting arguments are made. The entities contained within the universe of discourse comprise a complete vocabulary. All correct, consistent statements that can be made within the universe of discourse, then, use only symbols that are available within that universe.

In our work, the universe of discourse is that of *design specification*; that is, the specification of – or statement of facts about – a design problem and the various components and aspects of its solution, without consideration to any processes required to generate that solution. Within this universe of discourse, the entities of relevance are whatever design entities are available to the designer in order that he/she may fulfill the task at hand. This greatly restricts the space of possible entities (as compared to, for example, ZF, where any item at all may be considered to fall within the universe of discourse). It is exactly because of the specific nature of the entities involved that much more can be said about them than is normally possible (as, again for example, in ZF). *The notion of a restricted universe of discourse is essential to being able to derive the hybrid model at all.*

The authors informally define a *design entity* in HM as some real-world structure that is meaningful from a design point of view. This unit need not be physically realizable *per se*: it can be a process, such as a run of a finite element program or a manufacturing process plan. It can also be a *feature*, in that threads, holes, fillets, etc. are also design entities. HM deals, then, with the formalization of design entities. In the sections following, the exact formalization is stated and discussed.

3.2 Basic Structures and Concepts

The basic logical structure in HM is an **Object**. An **Object** captures a unit of information that is meaningful to a designer, and thus, an **Object** is the formal representation in HM of the informal notion of a design entity.

A bolt, a truss, an airplane, a hole and a run of a finite element program would all be represented by **Objects**. **Objects** may “contain” other **Objects** (this is discussed below). The entity represented by an **Object** need not be physically realizable (for example, a hole or a fillet), thus including *features* [21, 22]. **Objects** may represent machining and other processes.

The use of **Objects** is important because it permits *encapsulation* of information, i.e., the discretization of a quantity of information into meaningful structures that can be treated as single units. For example, the alphabet is a structure containing the ordered sequence of written expressions of the phonemes that compose the English language. Similarly, a *screw* is an ordered collection of information that models a device used in the real world as a kind of fastener.

Let the set of all **Objects** be denoted by O , and let X, Y, Z be members of this set (i.e. be **Objects**).

Axiom 1 (Uniformity of Structure) *All design entities are represented by **Objects**.*

Axiom 2 (Uniqueness of Object Identifiers) *A unique **Object** has a unique identifier.*

Although the relevance of axiom 2 may seem at first glance to be restricted to more practical issues (i.e. those regarding definitions of databases and other computer software systems), there is also a more basic, philosophical concern. We must be able to identify any design entity if we are to use it. The process of identification is essential to be able to distinguish between entities in the universe of discourse. The manifestation of the process of identification is the attachment of an identifier to an entity. Since **Objects** model design entities directly, we *must* also be able to identify **Objects**.

At a more practical level, an **Object** is a conceptual tool that permits us to abstract, infer, and deduce information about real world entities, and to classify them by their conceptual definitions.

One of the principal concerns in any system of information is that of ordering or organizing the information. That is, the definition of relationships between entities is of primary importance. It is a means of making *explicit* the information

that would be otherwise only *implicit* within a collection of data. In HM, this is done with *relations* and *functions* as defined in ZF.

A relation in set theory is a statement that defines a relationship between entities. Application of the relation to the members of a collection of sets (A, B, C, \dots) results in a set of ordered sequences $\langle a, b, c, \dots \rangle$ such that $a \in A, b \in B$ and so on. The ordered sequence is often used as a representative notation for the relation itself. That is, $\langle x, y \rangle$ (where $x \in A$ and $y \in B$) represents all ordered pairs arising from the application of some relation R on two sets A and B .

A function in set theory is defined the same as is a relation, with the added restriction that the relation R can map a single value of $x \in A$ to exactly one member of $y \in B$. Functions are often written $f : X \rightarrow Y$ and are read “ f is a function that maps the members of set X to the members of set Y ” [18].

It is noted that functions and relations as defined within set theory provide the formal grounds not only for mathematical functions and relations as they are understood outside the field, but also for relations in relational databases, methods in object oriented systems, procedures and routines in conventional programming languages and links in hypertext. They are also essential to the development of specification languages such as “Z” [23].

Functions and relations are used to order members of sets, and their formalization is a key part of the authors’ work. HM currently supports four ordering mechanisms for design information based on functions and relations. They are discussed in section 5.

4 THE STRUCTURE OF OBJECTS

4.1 Introduction

A design entity is defined by its observable or otherwise known *attributes*. These attributes define the structure of, and function provided by, the entity. For example, a tree is defined by its shape, size, strength of the wood, etc. In fact, the concept “tree” is really nothing but a label attached to a set of observed attributes that are shared by all trees [24]. Attributes are important in design because they model identically the designer’s perceptions of the real world.

Let the set of all attributes be denoted by A , and let a, b, c denote members of that set.

Definition 1 (Definition of an Object by its Structure) *An Object is a set of unique, identifiable, measurable attributes.*

$$\forall(X) [(X \in \mathbf{O}) \Rightarrow (\text{SET}(X)) \bullet (\forall(a) [(a \in X) \Rightarrow (a \in \mathbf{A})])] \quad (1)$$

Since A is the set of all attributes, we can also write this as:

$$\forall(X) [(X \in \mathbf{O}) \Rightarrow (X \subset \mathbf{A})] \quad (2)$$

A unique design entity is one whose attributes differ in some way from the attributes of all other real world entities. If the attributes are the same, then any operation that can be performed on the entities will yield the same results.

Axiom 3 (Identity of Objects) *If the sets of all attributes of any two Objects have identical members, and if corresponding attributes in each Object have equal values, then the two Objects are identical.*

$$\forall(X) [\forall(Y) [(X = Y) \equiv (\forall(F)(F(X) \equiv F(Y)))]] \quad (3)$$

where F is any unary predicate.

This axiom of identity is identical to the definition of the Axiom of Extensionality in ZF [20], namely

$$(A = B) =_{df} \forall(x)((x \in A) \equiv (x \in B))$$

but is derived from design considerations rather than purely mathematical ones.

The authors’ formalization of Object attributes is designed to account for certain properties that are particularly important to engineering design.

4.2 Views

The first property is termed *relevance*, and it is manifested as *views* of Objects.

A view of an Object partitions its attributes, making only some visible and manipulable. Views do not affect the Object itself, but creates a projection of the Object wherein only certain attributes are accessible. A view partitions the attributes of an Object according to criteria explicit in the view itself. The attributes remaining after the partitioning form a subset of the attributes of the Object being viewed, and are stored in another Object called a *view* Object. Since an attribute may be active in a number of views but need not be active in all views, we define an attribute as containing a context-dependent set of view criteria within which it is relevant.

If we consider a particular Object to be a “complete” model of some design entity, then any proper subset of the members of the Object can be considered a specific view of the Object with respect to the selected members.

We now demonstrate how ZF can be used to formalize these notions. Using the definition of subsets in [20], we can write in the notation of HM:

$$\forall(X)(\forall(Y) [(X \subset Y) =_{df} \forall(x) [(x \in X) \Rightarrow (x \in Y)]])$$

If $(X \subset Y)$ but $(X \neq Y)$, the subset is a *proper subset*.

Clearly, the number of possible views of an object is the number of elements in the power set of the members of the Object, written $\mathcal{P}(X)$.

Many of these views would be trivially unimportant to a designer. But there is no way to define *a-priori* only the views that are important. On the other hand, we can restrict our definition of a view in a manner similar to the way that the definition of a subset is restricted in set theory. This kind of restriction also happily prevents certain kinds of logical paradoxes from arising in the resulting theory.

We begin with what is generally called the Axiom-schema of Separation in ZF.

$$\exists(S) [\forall(x)((x \in S) \equiv ((x \in A) \bullet \varphi(x)))]$$

where there are no free occurrences of S in φ . This says that for *any* set A , and any propositional function (i.e. predicate) φ , there is a set S that is a subset of A that contains only members of A that satisfy φ .

We call it an *axiom-schema* because the symbol φ represents a *group* of predicates. We may write the set of all predicates φ as Φ . The equation above therefore actually represents a group of axioms, each having a different predicate substituted for $\varphi \in \Phi$. In section 3.1.2, it was explained that the restricted nature of the universe of discourse of HM lets us investigate the nature of entities within that universe much more closely than is possible in ZF. Here is one example of the degree of detail that is possible: in ZF, little can be said about the possible predicates that can be substituted for φ in the Axiom-schema of Separation; but, as we shall see below, in HM we can investigate a number of important groups of predicates that apply to design. Views are the first such case.

We interpret the Axiom-schema of Separation for HM as follows: For an Object X and any propositional function φ , there is another Object Y whose members form a subset of the members of X , and all of which satisfy φ .

When we say that there can be no free occurrences of S in φ , we mean only that φ must not contain occurrences of $\exists(S)$ or $\forall(S)$ since this would imply that S is defined in terms of itself and would lead to paradoxes. This is not a real problem in HM itself, because it would be meaningless to define a view with respect to itself, so a designer would likely never attempt such. However, it is enforced in HM for completeness and consistency.

In HM the axiom of views is in fact an axiom schema subset of that of the Axiom-schema of Separation. It can be written as follows.

Axiom 4 (Axiom-Schema of Views)

$$\exists(Y) [\forall(a)((a \in Y) \equiv [(a \in X) \bullet \gamma(a)])] \quad (4)$$

where γ contains no free occurrences of Y .

γ is a new symbol, and is used to represent a possible φ predicate that defines a view; different γ predicates will produce different views. γ , then, is the criterion by which a specific view is defined. These criteria are attribute-specific. For example, if γ were such that only attributes that modeled spatial dimensions satisfied it, the resulting view of an Object would be its 3D geometric representation.

Furthermore, we define Γ as the set of all views; i.e. $\Gamma \subset \Phi$ and $\gamma \in \Gamma$.

Note that if theorem 4 is stated simply ($a \in X \equiv a \in Y$), then $X = Y$ by the axiom of Object identity. But by increasing the constraint on Y by adding $\gamma(a)$, Y must be a *subset* of X . This notion of a view being a subset of an Object is captured by the following definition.

Definition 2 (Views) VIEW() is a binary function whose input is an Object and a view criterion specification (γ), and whose output is another Object called a view Object whose attributes are a subset of the attributes of the input Object, selected according to the given criterion.

$$\forall(X) [\forall(\gamma)(\exists(Y) [(Y = \text{VIEW}(X, \gamma)) \bullet (Y \subset X)])] \quad (5)$$

Objects that are the same in every regard are identical. Clearly, this statement applies to view Objects as well. We can also say something about the Objects that generated the views. If there exists a view Object that can be derived from two non-identical Objects through the use of a single view criterion γ , then we define the two non-identical Objects as being *similar*.

The inclusion of similar Objects in HM is motivated by the observation that for many design tasks, only a certain view of an Object is sufficient to allow its completion. Also, different views of the same Object can be used by different designers to perform different tasks; separation of components of an Object into views can help maintain design information consistency and integrity. It is therefore important to be able to make statements about the Objects that give rise to the view Objects.

Theorem 1 (Similarity of Objects) Any two Objects are similar if the same view of the two Objects produce identical view Objects.

$$\forall(X) [\forall(\gamma) [\forall(Y) ((X \sim Y) \equiv (\text{VIEW}(X, \gamma) = \text{VIEW}(Y, \gamma)))]] \quad (6)$$

The symbol \sim is used to denote similarity of Objects.

The authors' motivation to have views of design information is four-fold. First, completeness requires that HM extend to cover the entire universe of discourse; and in a design environment, the universe of discourse includes views. Second, from the standpoint of conciseness, views permit a structure to exist in the simplest form that maintains its semantics. Third, from an organizational standpoint, views permit information to be ordered by its relevance to a task. Lastly, views implement information hiding, which is desirable practically for a number of reasons. The designer will have a simpler task if only relevant information is visible. Information selected by view can be presented to the user in a more understandable form. Superfluous information can be excluded to increase efficiency and robustness. Views can provide uniform interfaces to parts of a database even though the internal structure of the Objects in the database may change.

View functions are an especially powerful tool when defined as acting on attributes. Given two attributes, the sets of view functions that include the attributes can be intersected to discover functional or other coupling. The view function sets of two Objects can likewise be operated on and the intersection set size can be used to measure of the degree of coupling between the Objects. Similarly, small sets of view functions can be used to study how different types of attributes affect the coupling of two Objects.

4.3 Domains And Ranges Of Attributes

4.3.1 Set Theoretic Considerations

Although we have defined Objects in terms of attributes, we have said little about the attributes themselves. But in order to do this, we need to review some set theoretic concepts that deal with domains and ranges of sets. These

concepts will form the basis of how attributes are dealt with in HM. First, we consider the formal set theoretic definition of a *relation on sets*.

$$\forall(x) [(x \in R) \Rightarrow (\exists(u) [\exists(v)(x = \langle u, v \rangle))]]$$

where x is an ordered pair, $u \in U$ and $v \in V$ (U and V are sets) and R is a relation. It is also the definition of the *Cartesian Product* $U \times V = R$. The domain and range of R are given by:

$$\begin{aligned} \text{dom}(R) &=_{df} \{x : \exists(y) (\langle x, y \rangle \in R)\} \\ \text{ran}(R) &=_{df} \{y : \exists(x) (\langle x, y \rangle \in R)\} \end{aligned}$$

Let D be the set of all possible attribute domains and R be the set of all possible attribute ranges. A is the set of all attributes, so we may now write:

$$A = D \times R$$

and $\text{dom}(A) = D$ and $\text{ran}(A) = R$. Also, because of definition 1, we can write for an Object X :

$$\begin{aligned} \text{dom}(X) &\subset D \\ \text{ran}(X) &\subset R \end{aligned}$$

4.3.2 Domains and Ranges in HM

An attribute representing a property or behavior of a design entity is specified by two pieces of information. First, the concept that typifies the attribute is needed: its *domain*. In the most general sense, domains of attributes are completely arbitrary. In most computer systems, attribute domains can include integers, real numbers, text, arrays, etc. The domains of attributes relevant to engineering design are discussed below. Second, a specification of how the property is exhibited by a particular entity is required. The set of possible values that an attribute can have is called the *range* of the attribute.

Axiom 5 (Domain of Attributes) *The domain of an attribute is the abstracted, observable, quantifiable property of a real world entity that the attribute represents. The domain of an attribute includes an associated dimensional unit.*

Axiom 6 (Range of Attributes) *The range of an attribute is the set of all values that are meaningful within the domain of the attribute, and any one of which may be the actual value within an arbitrary object containing that attribute. The set of values can be discrete or continuous, single-valued or multi-valued.*

The dimensional units mentioned above will be explained in the next section.

Let D be restricted to the set of all attribute domains in HM only, and let R be the set of all attribute ranges in HM only.

Theorem 2 (Attributes) *Attributes are ordered pairs $\langle d, r \rangle$ where $d \in D$ and $r \in R$, and the set of all attributes is the Cartesian Product $D \times R$.*

Finally, we can define the property of identity for attributes in light of what set theory provides.

Theorem 3 (Identity of Attributes) *Two attributes are identical if their domains are identical and their ranges are equal.*

4.4 Dimensions Of Measurement

In order for many kinds of attributes to be meaningful, they must not only be observable; they must also be measurable. If an attribute is not measurable, its value cannot be compared to other values or used in computation, and would hence be comparatively meaningless. In order to be measurable, an attribute must be of known dimensionality. Therefore, the domains of attributes must include a specification of *dimensions of measurement* against which the attribute can be compared.

In order to satisfy the condition of measurability of attributes given above, the authors have defined the members of the set of attribute domains D based on the primitive dimensional properties of physical entities. The members of D in HM are: length, mass, time, cost, quantity (or enumeration), NDU (non-dimensional units, for ratios, etc.) or any combination of these (e.g. velocity, energy, and so on). Although only length, mass and time are commonly considered, the authors have elected to add other dimensions because of their relative importance in design. This approach is far more powerful than schemes that only represent numeric quantities because it is a natural form of expression that is physically meaningful, and because it captures all the necessary semantics of dimensional standards at the axiomatic level. For example, correct dimensional analysis becomes an inherent property of HM.

The members of R , the set of ranges, in HM are: integers (\mathcal{I}), real numbers (\mathcal{R}) and text (\mathcal{T}). As of this writing, the authors are undecided as to whether complex numbers should also be included as possible range values: though they are of use in many design methodologies (particularly in the area of analysis), they can also be considered as composites made up of two real numbers.

4.5 Attributes and Constraints

Constraints are the principal driving force of the engineering design process. They are manifested at least in part as constraints imposed on information specific to design states and entities. Insofar as constraints are vital to design information specification, they must be represented by HM.

As was done with views, above, the authors define constraints to operate at the attribute level within the hybrid model. Both attribute domains and ranges may be constrained. Attribute domains may be constrained to be of specific types (e.g. modular assembly components might have a constrained kind of shape or material), and attribute values may be constrained to be constant-valued, single-valued or multi-valued, continuous or discontinuous, and so on.

HM supports constraint specification through the use of relations and functions. Relations define ordered pairs that can explicitly be constrained based on either the domain or range of the pair; functions provide criteria to partitioning sets, hence providing the means to test whether particular members of sets satisfy a constraint.

The issue of constraints in design is far more complex than may be implied here; the authors recognize that the NP-complete nature of constraint satisfaction makes this issue problematic [25]. However, the *process* of constraint satisfaction is a component of the design process itself, and therefore falls outside the bounds of the immediate concern of the authors in this document. While this simplifies our task here, it is also recognized that more work is needed to make the constraint support mechanism in HM explicit. However, it should be clear that the ability to *specify* constraints, in the form of functions and relations, is inherent in HM.

It is noted that by the set theoretic definition of a relation, the set $A = D \times R$ requires that a relation on the members of D and R must exist and that it that must be satisfied for any valid members of A . If any combination of members of D and R is allowed, then some of the resulting attributes would be meaningless in a design environment. For example, an attribute with the domain of *quantity* cannot have a range within the set of real numbers. Clearly, the acceptable members of A must be *constrained* to remain within a realm meaningful to design. Such a constraint is provided by the relation defining $D \times R$.

5 ORDERING SCHEMES FOR DESIGN INFORMATION

Having established the theoretical foundations of the internal structure of *Objects*, we now turn to the issue of organizing *Objects* into meaningful collections. This section covers four ordering schemes (or *abstraction mechanisms*) that can be imposed on *Objects* to make semantic content explicit. These mechanisms are derived from

considerations of the types of information available to a designer and are defined within a set theoretic framework (i.e. HM).

5.1 Types Of Objects

One of the most natural and useful abstraction mechanisms for ordering (or classifying) entities is by structural similarities of the entities. Schemes of this kind provide *partial* orderings of information because many such schemes may be imposed on the same collection of entities, each yielding a differently ordered collection. The human mind thinks about entities so grouped by thinking about an abstracted (or *generalized*) entity that captures only that which is constant about all the members of the collection. It leaves indeterminate (or at least variable) those aspects of the members of the collection that are used as keys in the partial ordering operation. In HM, the generalized, conceptual entity meant to represent a partially ordered collection of design entities is called a `TYPE`. The partial orders that `Types` impose on collections of entities define relationships shared by the members of the collections, thus making information regarding the members explicit. Since `Objects` in HM model real design entities, and since `Objects` are defined in terms of attributes, `Types` in HM must model relationships between `Objects` by modeling relationships between attributes of `Objects`.

Because of their abstract nature, `Types` do not exist at the same “logical level” as do `Objects`. The notion of logical levels descends from Russell’s Theory of Logical Types [19] and is basically a generalized mechanism to distinguish between sets based on the degree of abstraction required to create the sets. It becomes essential in ZF to keep clear the differences between sets, sets of sets, sets of sets of sets, and so on, or various kinds of paradoxes can occur that will render the axiom system inconsistent. The Axiom-Schema of Separation, as it is used in [20] and by the authors, implies this same distinction between logical levels. Thus, ZF supports the distinction of logical levels according to Russell’s theory. The distinction is also important to HM.

For a given `Object`, we might expect predicates such as “*This entity has a threaded shaft*” or “*This entity is made of cold rolled steel*”. For a `Type`, we might expect predicates such as “*This entity is a bolt*” or “*This entity occupies space and is made of metal*”. Here we see a basic property of entities that are of different logical levels: the kinds of attributes that can be predicated on entities depend on the logical level of the entities.

There is more to the notion of logical levels than the authors suggest here, but what has been stated is sufficient for our purposes.

It is noted that in considering `Types`, we disregard any predicates that depend on the values of attributes of `Objects`; i.e. we know at once that a predicate such as “*This entity is 3.5 centimeters long plus or minus 1 millimeter*” applies to an `Object`, while a predicate such as “*This entity has a dimension that we call its length*” applies clearly to `Types`. The distinction is that the former is a predicate on a design entity, and the latter is a predicate on an abstract entity that generalizes some aspect of the former. Notice too that the former mentions an attribute domain (length) and a range value (3.5 centimeters . . .) while the latter mentions only a domain. The process of generalization of attributes, then, involves neglecting the values of the attributes and dealing only with their domains.

A `Type` is a collection of attributes, as is an `Object`, but the attributes in a `Type` regard only our generalized understanding of `Objects` and not any particular `Object`. A `Type`’s attributes may describe `Objects` by describing the domains that the members of a collection of `Objects` would all have.

In a more practical sense, we may state this as follows.

In HM, `Objects` are *typed* (or classified) by their structure: similarities in structure are expressed by similarities in the *domains* of the attributes of the `Objects`. Attributes of `Objects` are quantitative measures and represent in the hybrid model only those quantitative aspects of design entities.

The criteria used to define a `Type` of `Object` is based on attribute domains. An `Object` is included as an instance of a `Type` if the domains of all its attributes map identically to all the domains in the criterion.

Axiom 7 (Abstraction of Structure) *Abstraction of Object structure is based on generalization of Object attributes and results in Types, which are Objects of one higher logical level than are the Objects being abstracted.*

Definition 3 (Object Types) *A Type is an Object that defines a collection of other Objects, the domains of the*

attributes of all the members of which are defined by the `Type`.

$$\forall(X) [\forall(Y) [\forall(T) [(IS_A(X, T) \bullet IS_A(Y, T)) \equiv (\text{dom}(X) = \text{dom}(Y))]]]] \quad (7)$$

The authors denote the set of all `Types` by `T`, and a member of this set by `T`. The unary predicate `IS_A` is defined as:

Definition 4 (The Typing Predicate) `IS_A()` is a unary predicate that is satisfied by an `Object` `X` and a `Type` `T` iff `X` is of `Type` `T`.

$$IS_A : O \rightarrow T \quad (8)$$

Since `Types` are just special `Objects`, they obey all the axioms regarding `Objects`. In particular, `Types` obey the axiom of `Object` identity (axiom 3); this means that all `Types` are unique. If all `Types` are unique, then there can be only one `Type` that describes a particular `Object`. Therefore, every `Object` is of exactly one `Type`, and the ordering mechanism of `Types` provides an exact one-to-one mapping between `Objects` and their `Types`.

As we have already seen in section 4.2, the key to providing a *consistent* theory of design information lies with the Axiom-Schema of Separation. Again, our approach is to interpret the Axiom-Schema in terms of design considerations and eventually draw forth a subset of the φ predicates that have meaning from a design standpoint.

We begin with the Axiom-Schema of Separation. It is written:

$$\exists(S) [\forall(x)((x \in S) \equiv ((x \in A) \bullet \varphi(x)))]$$

With regards to `Types`, the authors interpret the Axiom as follows: there exists a subset `X` of the set of all `Objects` `O` all the members of which satisfy a predicate θ . In this case, θ is the predicate that differentiates `Objects` by type – in other words, `IS_A`. We can then reduce the Axiom-Schema of Separation to the following axiom in `HM` for `Types`.

Axiom 8 (Axiom of Types)

$$\forall(T) \quad \exists(X) [\forall(Y)((Y \in X) \equiv [(Y \in O) \bullet IS_A(Y, T)])] \quad (9)$$

The axiom of `Types` is not an axiom-schema because it uses precisely one predicate: `IS_A`.

Implicit in this discussion and in all the theorems and axioms in this section is the fact that `Types` and `Objects` are not directly comparable because they are of different logical levels. Therefore, we cannot make apparently intuitive statements such as $O \cap T = \emptyset$ because the entities contained by the sets `O` and `T` are not compatible. Were we to make statements such as that given, the resulting system would become inconsistent and there would be no way of assuring that all statement formulable in `HM` are provably correct. This is a good example of the power of set theory: it forces clarity of thought, giving a system wherein logical errors are more easily detectable and avoidable, without restricting the freedom to make consistent, relevant statements about design information.

5.2 Aggregation Of Objects

We have seen that `Types` permit partial ordering of `Objects`, and function by generalizing the attributes of those `Objects`.

An *aggregate* is also an ordering mechanism for `Objects`, but it works through a different abstraction: recursive containment. Put simply, an aggregate is an `Object` that contains other `Objects`. There is nothing in `HM` (or in `ZF` for that matter) that would prevent the value of an attribute from being another `Object`. In fact, it is an important part of set theory that the value of any element of a set may actually be another set.

Aggregates permit the hierarchical ordering of `Objects`. Design is strongly hierarchical. Parts may be assemblies of other parts. Even processes (for example, a finite element analysis, or a machining process) are composed of subprocesses. The larger the design problem, the more important hierarchical ordering becomes.

Once again, we rely on the Axiom-Schema of Separation to guide us in formalizing our notion of aggregate `Objects`. In this case, we write it as the following axiom schema.

Axiom 9 (Axiom-Schema of Aggregates) *There is a subset O_A of the set \mathbf{O} all the members of which satisfy a particular predicate δ .*

$$\exists(O_A) [\forall(X)((X \in O_A) \equiv [(X \in \mathbf{O}) \bullet \delta(X)])] \quad (10)$$

O_A is an aggregate Object, and δ is one of a set of predicates Δ used to define membership in aggregate Objects.

The predicate set Δ is an important one: it provides the means of defining all the necessary relationships in an aggregate Object. The consequences of this statement are made quite clear by considering a simple physical assembly of components. Objects would model each component. An aggregate Object would model the assembly. It would be the role of the δ predicate for that assembly to provide the exact relationship that exists between the components in the assembly. In other words, it is the aggregate predicates Δ that permit the definition of how components in a physical assembly mate, the tolerances of the fit, the manner in which the mating occurs (the assembly process itself) and so forth.

An aggregate Object does not contain *only* other Objects, but has attributes itself. For example, an aggregate representing a physical assembly of parts would have attributes of size and shape. These clearly cannot be derived from the components of the aggregate alone (i.e. information about a part tells us nothing about the size of the assembly in which the part is to be used). Aggregates, then, are not just sets of Objects, but Objects with attributes whose values are other Objects.

Aggregate Objects do differ from non-aggregates in one very important way: they are not all of the same logical type. Although this difference is handled by the predicates Δ , the issue is important enough to deserve special mention. Attributes that are predicable on parts of an assembly are not predicable on the assembly itself. For example, the volume of a part is calculable from its geometry, but the volume of the assembly is calculable by summing the volumes of its parts. Methodologically, there is a vast difference between the attribute of volume of a part and the attribute of volume of an assembly. In order to maintain the consistency of the theory, it is necessary to ensure that these differences are accounted for. This is possible by separating the logical levels at which aggregates and non-aggregates exist.

At this point, we could begin a detailed study of the properties of the predicates in Δ , but shall not. This document is meant to be both an overview of HM and a statement of the *base* axioms that compose it. In this spirit, then, we defer such discussions.

Finally, the authors note the conceptual similarity between the notions of aggregation and part assembly. Aggregate Objects can recursively contain other aggregates; the recursion ends when a non-aggregate Object is found. Each level of aggregation implies a higher logical type than the last, with non-aggregates being of the lowest logical type. HM can distinguish between all the various levels of aggregation as well as between any aggregate and any non-aggregate simply by the kinds of attributes found in an Object. Similarly, assemblies can contain sub-assemblies recursively; the recursion ends when a part is found. Each level of assembly, sub-assembly, and so on, is functionally and structurally distinct, with parts being the basic building blocks of all assemblies. In the creation, analysis and manufacture of assemblies, it is important to be able to differentiate between the different levels of assembly, and between parts and assemblies.

We see, then, that the abstraction mechanism of aggregation is very well suited for the modeling of design entities that are hierarchically defined.

5.3 Classification Of Objects

In addition to classification by structure, Objects can be also classified by the function they are meant to provide. The importance of capturing the semantic content of function is best exemplified by conceptual design.

Conceptual design is one of the first steps in a design process, and impacts the greatest on downstream decisions [26, 27, 22]. In general, conceptual design is considered to be the mapping between the function provided by some entity and the physical manifestation of the entity. Very little is known about conceptual design and we do not presume a simple solution to the problem here. However, the authors have devised a mechanism to ease the development of a system of classification by function.

The mapping between structure and function is not necessarily one-to-one: a particular structural component may

provide more than one function, or vice versa. We cannot use the `Type` mechanism discussed in section 5.1 because such classification occurs as a one-to-one mapping that is clearly inappropriate here. Hence, the structural properties of a design entity do not entirely capture the semantics of its function. A classification mechanism different from that of `Types` is needed.

The authors define a `Class of Objects` as an aggregate `Object` whose members all represent real world entities that exhibit a given function. The hybrid model considers inclusion of an `Object` in a `Class` sufficient to establish that the `Object` exhibits a given function. An `Object` that provides more than one function would be a member of more than one `Class`.

It is unclear to the authors at this time whether a `Class` should include information for modeling of the function itself, or whether this information should be contained by the member `Objects`, or by the δ function of the aggregate that relates them. The relationship modeled by the function would permit access to the members of the `Class`, much as a `IS-A` provides for `Types`, but without the constraints that `IS-A` imposes on the attributes of member `Objects`. In particular, though an `Object` is constrained to be of exactly *one* `Type`, it may be a member of *many* `Classes`.

5.4 Specialization And Generalization Of Types

5.4.1 Specialization Of Types

The abstraction mechanism of *specialization* is implemented in `HM` by *inheritance* of `Type`. Inheritance is a mechanism that acts on `Types` in a manner similar to aggregation in `Objects`, but wherein there is no layering of logical levels, as there is in aggregation. Instead, inheritance is controlled by the operation of set *union*.

The difference between inheritance and aggregation is very important from a semantic point of view. For example, to say that an automobile inherits the attributes of its engine (e.g. power) is meaningless; the reason why humans can make sense of such statements is because we can *interpret* it correctly and extract necessary *implicit* information from the statement. This highly informal and subjective approach is very undesirable. In order to phrase the above example in terms that can be supported by a formal theory, we would say that the automobile is an aggregate, one component of which is an engine that has a certain power rating.

Because the hybrid model is meant to formalize design information, the distinction between aggregation and inheritance becomes essential.

A `Type`, then, is the union of all the attributes of the `Types` that are inherited by it. Union of sets is very well understood in set theory and provides a simple and rigorous way to formalize specialization through inheritance.

Using the Axiom-Schema of Separation again, we can write the following.

Axiom 10 (Axiom of Specialization) *For a given type T , if there exists a set of Types $\{U\}$ such that T contains all the attributes of all the members of $\{U\}$, then T is said to inherit from the members of $\{U\}$, and T is a Type specialized from the Types in $\{U\}$.*

$$\forall(T) \quad \exists(\{U\}) [\forall(V)((V \in \{U\}) \equiv [(V \in T) \bullet (V \subset T)])] \quad (11)$$

In design, specialization is an important mechanism because it permits the creation of specialized `Types` from a collection of more general, abstract `Types`. Thus, specialization is a *top-down* procedure.

Design also tends to be a top-down process [28, 29], moving from the general (conceptual design) to the specific (detailed design). This impacts on how we treat design information. Because design begins from the general and moves to the specific, we can expect that at an arbitrary point along the development of a design artifact, information regarding the artifact will be incomplete in detail.

Specialization, then, being a top-down process, is used in `HM` to permit incomplete information about design entities to be captured in a consistent manner, and to permit the generation of (application) specific `Types` from general `Types`.

5.4.2 Generalization Of Types

Generalization is the inverse of specialization, and from the purely theoretic point of view of `HM`, the relationship

between the two is bidirectional (i.e. specialization and generalization are opposites but equivalents). Thus:

Axiom 11 (Axiom of Generalization) *If all the members of a set of Types $\{U\}$ have in common some attributes, then we can define a general Type T containing those common attributes and state that all the Types in $\{U\}$ inherit them from T .*

$$\forall(\{U\}) \quad \exists(T) [\forall(a)((a \in T) \equiv [(a \in \mathbf{A}) \bullet (\forall(V) [(V \in \{U\}) \bullet (a \in V)])])] \quad (12)$$

Generalization is important in design for two reasons. The first reason is quite practical: in an environment where a number of Types of design Objects have been generated independent of each other, generalization provides a formal mechanism by which we can *integrate* our models of the Objects. The Types can be examined for similarities, and general Types created that can then be inherited by each of the other Types. Generalization can help to unify our models of design entities. Unification is important because it can (a) bring to light model inconsistencies thus reducing the chances of generating faulty models, and (b) increase the explicit information content of a given model.

The second reason that generalization is important is more theoretic and deals with taxonomic concerns. One obviously desirable goal in DTM is the generation of usable, globally applicable taxonomies of design entities. Taxonomies themselves are classification mechanisms that can help standardize our models of design entities and control the information required for the models. Generalization in HM gives us a very specific, formal methodology for generating design entity taxonomies. Taxonomies resulting from the application of generalization to Types in HM would result in inheritance networks of Types that would permit the classification (at least in theory) of an arbitrary kind of design entity. The issues involved in generating such taxonomies are interesting and many, but are beyond the scope of this document.

6 CONCLUSIONS

6.1 Summary And Discussion

The hybrid model is still under development, but the core of the model as presented here is accurate and consistent, and should not change as the model develops. HM is a variant of classical ZF set theory, extended and interpreted to suit engineering design information. It is not intended to automate the design process, but rather to provide a structured notation that makes information about design entities clearer and thus permits the designer to apply whatever thought processes he/she prefers. The authors perceive the designer as existing in a symbiotic relationship with design tools such as HM, rather than being replaced by them.

HM is based on a functional model of the engineering design process that permits us to view design information and the various processes that act on this information during the course of design development as separate areas. The key issue that permits the model to be stated at all is that the universe of discourse of ZF is restricted to only those entities pertinent to design. In this way, we can make more specific statements (i.e. axioms) than the more general ZF permits.

HM does not alter any of the axioms of ZF, upon which it is based, but rather expands on them. Because of this, and of the arguments in section 3.1.1, the authors assert that HM is “correct” from the point of view of logic; i.e. HM is a formal, consistent axiom system.

Objects are sets of attributes, and capture meaningful information about design entities. This provides a natural form of expression for design information because Objects are conceptually equivalent to the entities they model. Attributes are defined in terms of domains and ranges; domains of attributes in HM include generalized dimensions of measurement. Views permit Objects to be partitioned according to criteria specific to particular design tasks. Relations and functions are used to define general classes of operators on Objects, and provide a flexible, extensible mechanism for the logical representation of various kinds of relationships and constraints. The specific relationships of structural similarity, functional similarity, aggregation, specialization and generalization are all modeled in a straightforward way by HM.

Since the hybrid model applies to different design tasks, such as solid modeling, analysis, and so on, it presents an integrated approach to the specification of design information that is extensible: new entities and relationships can be specified using the model without altering the model itself.

HM provides a basis for building taxonomies of design entities, a generalized approach for making statements about design entities independent of how the entities were generated (i.e. independent of the design process used to create them) and a formal syntactic notation for the standardization of design entity specification.

6.2 Future Work

Research on the hybrid model is going on in a number of different areas. The role of constraints in the hybrid model must be examined, and suitable theory generated. Relatively little work has been done in this area. This may be because the design entities that are constrained have been vaguely and/or imprecisely defined in the past. A formal understanding of these entities may also help us understand the constraint satisfaction problem better.

Classification of design entities by function is another area where current understanding is weak; emphasis will be placed on this in the future work of the authors.

Due to the hierarchical nature of design entities (particularly mechanical design entities), aggregation is a key abstraction mechanism. Further study of the Δ predicates that HM supports for the definition of aggregates is warranted.

Other ordering mechanisms not discussed in this document (such as parameterization) can be very useful in a design environment. The authors intend to expand the number of mechanisms supported by HM in the future.

The next stage in the authors' research will involve testing the hybrid model by coding an experimental software system that will satisfy it. The software system will be used to check the applicability of HM to real design situations. Examples in the areas of solid modeling, finite element mesh specification, assembly of components and kinematic analysis will be used.

At the theoretical level, a formal theory of design information will be a useful tool for the study of the design process itself by providing a uniform lexicon and grammar for information specification. The authors' note that it is possible to blend ZF and first order predicate calculus [18]. The predicate calculus is the formal basis of such tools as expert and knowledge processing systems. It is interesting to speculate about the nature of the results of a combination of the predicate calculus with HM, and its possible applications to DTM.

At a more practical level, application of the theory to design systems has the potential to improve communications between designers by providing a common vocabulary, assist in the standardization of design specifications, and lead to new and more powerful software tools to aid the designer.

References

- [1] Thomas G. Dietterich and David G. Ullman. FORLOG: A Logic-based Architecture for Design. In John S. Gero, editor, *Expert Systems in Computer-Aided Design*, Proceedings from IFIP WG 5.2 conference on Expert Systems in Computer-Aided Design, pages 1–17, Amsterdam, 1987. North-Holland.
- [2] K. Ishii and P. Barkan. Rule-based Sensitivity Analysis. In John S. Gero, editor, *Expert Systems in Computer-Aided Design*, Proceedings from IFIP WG 5.2 conference on Expert Systems in Computer-Aided Design, pages 179–198, Amsterdam, 1987. North-Holland.
- [3] Peter Struss. Multiple Representation of Structure and Function. In John S. Gero, editor, *Expert Systems in Computer-Aided Design*, Proceedings from IFIP WG 5.2 conference on Expert Systems in Computer-Aided Design, pages 57–84, Amsterdam, 1987. North-Holland.
- [4] Douglas S. Green and David C. Brown. Qualitative Reasoning During Design about Shape and Fit: a Preliminary Report. In John S. Gero, editor, *Expert Systems in Computer-Aided Design*, Proceedings from IFIP WG 5.2 conference on Expert Systems in Computer-Aided Design, pages 93–112, Amsterdam, 1987. North-Holland.
- [5] Gregory L. Fenves. Object-Oriented Programming for Engineering Software Development. *Engineering with Computers*, 6(1):1–15, 1990.
- [6] Suad Alagic. *Object-Oriented Database Programming*. Springer-Verlag, New York, 1989.
- [7] Stanley B. Zdonik and David Maier. *Fundamentals of Object-Oriented Databases*, pages 1–36. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 1990.
- [8] J. R. Dixon and M. R. Duffey. The Neglect of Engineering Design. *California Management Review*, 32(2):1–19, 1990.
- [9] D. L. Hawla and H. Neishlos. Knowledge Acquisition for Effective and Efficient Use of Engineering Software. *Engineering with Computers*, 6(2):67–80, 1990.
- [10] Nam P. Suh. *The Principles of Design*. Oxford University Press, New York, 1990.
- [11] John R. Dixon. The State of Education. *Mechanical Engineering*, pages 64–67, Feb 1991.
- [12] J. Wade and J. S. Colton. A Framework for Feature-Based Representation of the Design Process. *Engineering with Computers*, 6(3):185–192, 1990.
- [13] F. A. Salustri and R. D. Venter. Towards a Logical Theory of Engineering Design Information. In Gopal Gupta and Terry E. Shoup, editors, *Proceedings of the 1991 ASME Computers in Engineering Conference*, pages 161–167. ASME, American Society of Mechanical Engineers, 1991.
- [14] Tetsuo Tomiyama and Hiroyuki Yoshikawa. Extended General Design Theory. In H. Yoshikawa and E. A. Warman, editors, *Design Theory for CAD*, Proceedings from IFIP WG 5.2 Working Conference on Design Theory for CAD, pages 95–130, Amsterdam, 1987. North-Holland.
- [15] Owen R. Fauvel. Expanded Use of Function Language in Mechanical Design. In N. Popplewell and A. H. Shah, editors, *Proceedings of the Thirteenth Canadian Congress of Applied Mechanics*, pages 692–693, Winnipeg, Man., 1991. Print Management, Ltd.
- [16] John R. Dixon. New Goals for Engineering Education. *Mechanical Engineering*, pages 56–62, Mar 1991.
- [17] Mark H. Chignell. A Taxonomy of User Interface Terminology. *ACM SIGCHI Bulletin*, 21(4):27–34, 1990.
- [18] Irving M. Copi. *Symbolic Logic*. Macmillan, 1979.
- [19] Abraham A. Fraenkel, Yehoshua Bar-Hillel, and Azriel Levy. *Foundations of Set Theory*. North-Holland, 1973.
- [20] Patrick Suppes. *Axiomatic Set Theory*. Dover Publications, Inc., 1972.

- [21] P. H. Gu, H. A. ElMaraghy, and L. Hamid. FDDL: A Feature Based Design Description Language. In W. H. ElMaraghy, W. P. Seering, and D. G. Ullman, editors, *Design Theory and Methodology*, ASME Design Automation Conferences, pages 53–64, New York, 1989. ASME.
- [22] J. J. Cunningham and J. R. Dixon. Designing With Features: The Origin of Features. MDA Technical Report 3-88, Mechanical Design and Automation Laboratory, University of Massachusetts at Amherst, Amhurst, MA, 1988.
- [23] J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*. Cambridge University Press, Cambridge, England, 1988.
- [24] Julian Jaynes. *The Origin of Consciousness in the Breakdown of the Bicameral Mind*. University of Toronto Press, 1976.
- [25] James Bowen and Peter O’Grady. A Technology for Building Life-Cycle Design Advisers. In G. L. Kinzel and S. M. Rohde, editors, *Proceedings of the 1990 ASME Computers in Engineering Conference*, pages 1–7. ASME, American Society of Mechanical Engineers, 1990.
- [26] Micheal J. Wozny. Beyond Computer Graphics and CAD/CAM. In Tosiyasu L. Kunii, editor, *Proceedings of Computer Graphics International ’87*, pages 3–9, Tokyo, 1987. Springer-Verlag.
- [27] Zhentao Zhang and Stephen L. Rice. Conceptual Design: Perceiving the Pattern. *Mechanical Engineering*, pages 58–60, Jul 1989.
- [28] Kincho H. Law, Thierry Barsalou, and Gio Wiederhold. Management of Complex Structural Engineering Objects in a Relational Framework. *Engineering with Computers*, 6(2):81–92, 1990.
- [29] M. K. Zamanian, S. J. Fenves, C. R. Thewalt, and S. Finger. A Feature-Based Approach to Structural Design. *Engineering with Computers*, 7(1):1–9, 1991.