Original paper

Res Eng Design 14 (2003) 185–199
DOI 10.1007/s00163-003-0035-3

Topological structures for modeling engineering design processes

Dan Braha, Yoram Reich

Abstract In this paper, a mathematical framework for describing a variety of complex and practical design processes is developed. We demonstrate that our model has the desirable quality of representing several, seemingly distinct, approaches as instances of the same framework. In addition, General Design Theory is shown to be a special case of the proposed framework. Using simple examples throughout the paper, we also hint at the potential for the framework to serve as a basis for a descriptive study of design. Various design phenomena such as design failure, identification of design knowledge bottlenecks, and benefits of collaborative design could be understood using the proposed model.

Keywords Design process, Design analysis and synthesis, Design methodologies, Formal Design Theory (FDT), General topology

1 Introduction

Mathematical formalisms of design have been of great interest to many researchers over the years. The ability to formalize a complex problem such as design, and solve it using algorithms whose results could be proven and performance guaranteed, have been a major attracting force. This interest has been equally received with skepticism since these ideas have made limited-scope impact on real design. For example, General Design Theory (GDT) (Yoshikawa 1981; Tomiyama and Yoshikawa 1987; Reich 1995) models design as an immediate result obtained once all specifications are provided. The extended version of GDT models design as an evolution of models (Tomiyama and Yoshikawa 1987). However, the theory does not detail the nature or process of this evolution.

Received: 11 April 2001 / Revised: 7 January 2002
Accepted: 13 November 2002 / Published online: 6 November 2003

D. Braha (✉)
Department of Industrial Engineering, Ben-Gurion University,
PO Box 653, 84105 Beer-Sheva, Israel
E-mail: brahad@bgumail.bgu.ac.il

Y. Reich
Department of Solid Mechanics, Materials and Systems,
Tel Aviv University, 69978 Ramat Aviv, Israel

Most researchers and practitioners recognize that design processes cannot be fully formalized mathematically. Nevertheless, casting design in mathematical terms serves several goals. First, by developing increasingly better mathematical models of design we improve our understanding about the limit of formalizing design and the limit of automating it. Second, studying mathematical models of design could produce practical guidelines or ideas for implementing design support procedures or systems.

Consider the following design scenario. A designer obtains general specifications from a customer. She has in mind several general ideas about addressing the specifications. She studies the specifications in relation to the general ideas and refines them. She selects two of the ideas and details them. The analysis that follows uncovers constraints not previously anticipated. These are added to the specifications. The analysis results are contrasted with design codes and requirements. New laws for recycling take effect and need to be considered as part of the evolving specification list. Therefore, an extended design process commences and terminates when an acceptable solution is found.

Our goal is to develop a mathematical framework that is broad enough to describe a variety of complex and practical design processes. For example, the framework has to account for the following properties of design processes:

1. Design starts from some abstract specifications and terminates with a description of a product.
2. In design, the product specifications are gradually refined. Better understanding of the specifications is a by-product of design.
3. Design is an iterative, exploratory, and sometimes chaotic process.
4. Intermediate states of the design process might include conflicting specifications and product description.
5. Design progresses by context-dependent activities: thinking, alternatives, and decisions emerge from the situation as it unfolds by bringing diverse knowledge to bear on the present context.
6. Designers make use of diverse knowledge whether they work alone or collaboratively.
7. Design is about finding solutions, not (globally) optimal solutions. Designers generally do not receive the knowledge or resources needed to achieve optimality.

Our mathematical framework should also be detailed enough (1) to allow theoretical statements about design

procedures to be made, and (2) to provide ideas for improving practical processes and insight regarding their outcome. In line with the descriptive focus of the framework, we do not suggest that we can automate design processes, nor is it our goal. Moreover, some of the constructs in the framework are computationally expensive to create or manipulate; therefore, our framework suggests that it is less than likely for design automation to be realized.

We present our framework in two steps in order to improve its clarity. Initially, we present a basic model that introduces several key concepts, even though it is unrealistic. Subsequently, we introduce an extended model that addresses the aforementioned properties of design processes. Due to space limitation, we illustrate only several concerns and leave the rest to subsequent papers.

The remainder of the paper is organized as follows: Section 2 provides an overview of the models; Section 3 describes the framework in more detail; and Section 4 illustrates the benefits of the framework. First, we show that GDT is a special case of our framework. Second, we show how a simple rule-based design system can be cast as an instance of the framework. Third, we briefly demonstrate how the learning system ECOBWEB (Reich and Fenves 1992) can be interpreted in our framework. Section 5 concludes the paper.

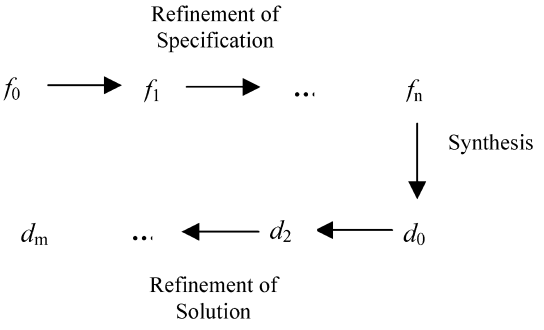


Fig. 1.

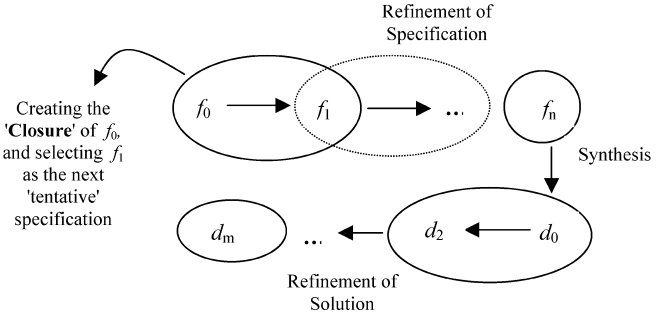


Fig. 2.

2
Overview of the models

This section presents an overview of the various "proximity" models for engineering design processes. The proposed models clarify and extend previous work (Braha and Maimon 1998), motivated by empirical observations on how engineers design (e.g., Akin 1979; Blessing 1995; Ullman et al 1986; Hales 1987) and the nature of complex product design activities (Blessing 1995; Hales 1987; Reich et al 1999). In the following, a basic model for design synthesis is described, and several key concepts such as *proximity* and *process* are introduced. Subsequently, an extended model for design synthesis is presented that reflects scenarios that are more realistic.

2.1
Basic model

The basic model formulates design as a process that starts from abstract specifications such as customer needs or functional requirements in the *function space*. These specifications are iteratively refined by moving to a better "proximal" specification list. At some point, the designer is able to match partial structural information, in the *structure space*, with the current refined specifications. Then, the process continues in the structure space by refining the partial structural information until a design solution is obtained. This process is depicted in Fig. 1.

Roughly speaking, this process is viewed as a local exploration procedure that finds a design solution (*terminal state*) with respect to the proximal structure. That is, if f_i is the "current" ("tentative" at stage i) specification list, then f_{i+1} is a proximal refined specification list if it does not differ substantially from f_i. Given a tentative specification list f_i, the local exploration looks for a proximal refinement of f_i. When no further refinement is possible[1], the "refinement phase" stops, the synthesis mapping begins, and a similar refinement process is carried out in the structural domain. We call this process the basic model.

To support local exploration of a specific design problem we need a method for finding the initial description at each phase of refinement (i.e., f_0 or d_0), and a proximal structure of any feasible description. We also need to know how to select among several proximal items at each refinement step so as to arrive at better solutions.

In this paper, we propose a formalism that captures the above considerations. This is done by introducing a proximal structure called *closure space* (or proximity space, see Cech 1966). For each functional description f (or structural description d) in the function space **F** (or in the structure space **D**), a *closure* $U_{\mathbf{F}}(f)$ (or $U_{\mathbf{D}}(d)$) is identified. Given two functional descriptions f and g, if $g \in U_{\mathbf{F}}(f)$ we say that g generates f, or f is generated by g. We also define the closure of any set of functions. In a logical framework, the closure operation may be associated with *abduction* (see Takeda et al. 1990 for discussion on abductive inference in design). That is, for any two expressions a and b, if $b \in U(a)$ then $b \rightarrow a$ (see Sect. 4.2). Obviously, the closure operation can be realized in many other ways (see Sect. 4) depending on the refinement strategies used to create the proximal structure. In view of the local exploration presented above, the model in Fig. 1 is elaborated in Fig. 2.

[1] For instance, when the designer reaches a specification f_n that cannot be refined due to limited knowledge, or when sufficient information is gathered that enables the "synthesis" mapping to take place.

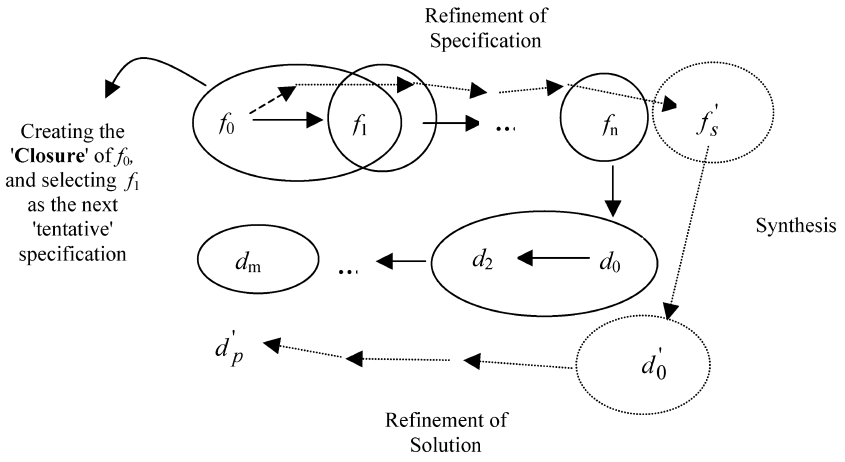


Fig. 3.

Each circle in Fig. 2 represents the closure of the current description (functional or structural). In general, the closure of a description contains more than a single element. A unique design process is obtained by selecting at each refinement stage a single generating element out of many possible ones.

The creation of the closure and the selection of a generating element are knowledge driven. Therefore, the availability, richness, and coherence of knowledge strongly influence the ability to obtain quality design solutions. Missing, partial, or otherwise poor quality knowledge will lead to familiarity with only part of the closure, potentially exploring only inferior parts of the closure, leaving out the more promising solutions.

We denote the synthesis operation as a mapping Υ from a functional description to a set of structural descriptions, where each selected structural description may be described in terms of partial information only. The selected structural descriptions correspond to the *output* of the particular synthesis method the designer uses. Our formalism provides the means for representing several possible design processes. For instance, the set of all design solutions (structural descriptions) that can be obtained from the initial specification list f_0 by applying two steps of functional refinements, synthesis, and two steps of structural refinements can be formally described as follows:

$$U_{\mathbf{D}}(U_{\mathbf{D}}(\Upsilon(U_{\mathbf{F}}(U_{\mathbf{F}}(f_0))))) \quad (1)$$

Equation 1 encapsulates the set of *all trajectories* that are obtained starting from f_0 and ending with a design solution with the same given number of steps. More generally, different design solutions or trajectories might require a different number of design steps as shown in Fig. 3. A single trajectory, which defines only one possible candidate development, is a list of "visited" functional and structural descriptions. A single trajectory can be described by selecting, at each stage, one description from the closure of the current description. In reality there are sometimes few parallel candidate developments as shown in Fig. 3.

2.2
Limitations of the basic model

The basic process model has several limitations. First, it is too linear. The designer carries out an extensive phase of specification refinement, followed by a "synthesis jump" and an extensive phase of solution refinement (see Fig. 1). This model is "basic" in the sense that some aspects of "real design" are not captured by the model, including (1) repeated interplay and multifold contextual mappings between the function and structure spaces, (2) the association-intensive search and feedback loops involved, and (3) the continuing learning that refines the requirements even during synthesis. To overcome these and other simplifications, we propose an extended topological model called *coupled design process*.

2.3
Real design model

The extended model captures the interplay between *design descriptions* (or process states), each of which is represented by a pair[2] of functional and structural descriptions $<f,d>$. This extension, which simultaneously handles the function and structure spaces, could be further expanded to explicitly incorporate and deal with more complex models that have functional, structural, behavioral (e.g., $<f,d,b>$), or other design aspects such as the environment, organization, and design processes employed by designers (Blessing 1995). The nature of the interplay is not built into the framework (similarly to not insisting on a particular form of proximal structure as discussed above). Therefore, the framework could model different views of design such as design as exploration (Smithers 2000), or empirical findings about design such as the diffused distinction between different design stages as seen in real projects (Dasgupta 1994; Hales 1987).

Formally, design descriptions are elements of the Cartesian product of the function and structure spaces $\mathbf{F}\times\mathbf{D}$, which is called the *design space*. A "coupled design process" is performed as follows. The designer starts with the initial design description $<f_0,d_0>$. The f_0 are the initial specifications and d_0 is the initial context description, e.g., a product during redesign or improvement or an abstract idea. In general, during the transition from the current

[2] This natural type of representation has been utilized, explicitly or implicitly, as a means for describing the design process by other researchers (e.g., Pahl and Beitz 1984; Gero 1990; Takeda et al. 1990; Dasgupta 1994; Simon 1996; Suh 2001). Here, we use it as a basis for modeling the underlying topological spaces.

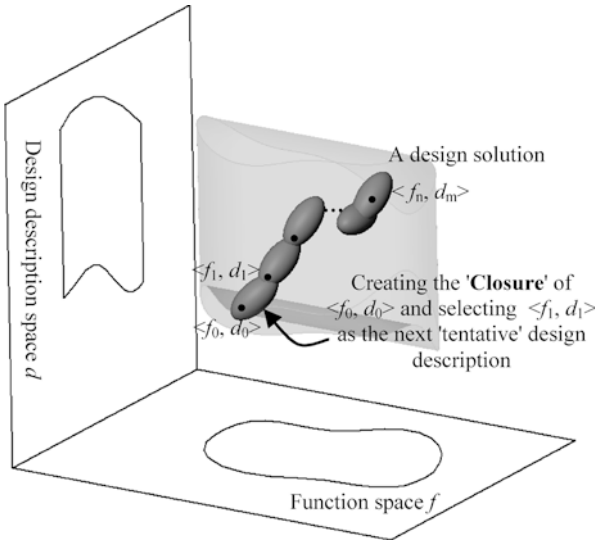


Fig. 4.

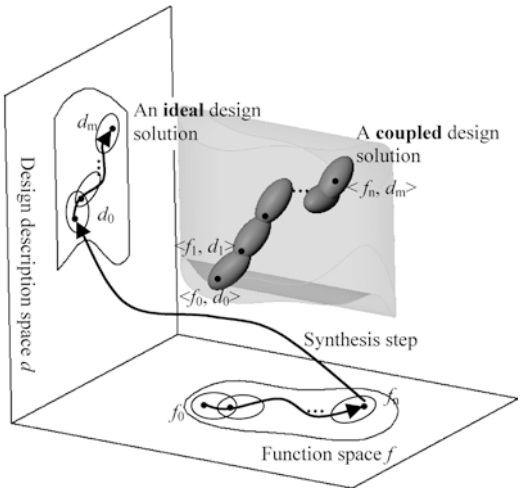


Fig. 5.

design description $<f_i,d_i>$ to the new design description $<f_{i+1},d_{i+1}>$, both structural and functional descriptions can be refined. This transition can explain complex processes involving simultaneous refinement of the structural and functional descriptions. It can also capture pure synthesis, which maps part of the functional description f_i to a structural description that augments the current structural description d_i. In this case, the new functional description f_{i+1} incorporates specifications that have already been satisfied as well as those that remain to be satisfied.

The refinement and synthesis operations are *context dependent*; that is, both f_i and d_i are needed in order to carry out the operations. In addition, $<f_i,d_i>$ may be refined in case some design constraints are violated (see example in Sect. 4.2).

To capture the transition operation using our formalism, we simply introduce a proximity structure for the space **F**×**D**. That is, for each design description $<f_i,d_i>$, a closure operation $U_{\mathbf{F}\times\mathbf{D}}(<f_i,d_i>)$ is identified. Every design description in $U_{\mathbf{F}\times\mathbf{D}}(<f_i,d_i>)$ generates the design description $<f_i, d_i>$. The coupled design process is illustrated in Fig. 4. Each circle in Fig. 4 represents the design descriptions obtained by applying the closure operation. In general, the closure of the current description includes more than a single element. In real design, it could contain an infinite number of elements. Thus, a *unique* design process is obtained by selecting at each stage a single element out of the many possible ones without necessarily creating the complete closure. A design process ends with a design description of the form $<f_n,d_n>$, where f_n are the detailed specifications manifesting a deeper understanding of the problem, and d_n is a design description that satisfies these specifications and can be realized. Given $<f_n,d_n>$, d_n may satisfy f_n though it may not be realized, in which case $<f_n,d_n>$ is not a final state. This intermediate favorable situation can happen particularly during conceptual or detailed design stages. In these situations, designers continue to refine the specifications knowing they are still incomplete. We might be able to say that in such a situation conceptual design terminates and another stage begins in a new space.

2.4
Relating the basic model to the real design model

If the closure operation $U_{\mathbf{F}\times\mathbf{D}}$ can be decoupled into two independent closure operations $U_{\mathbf{F}}$ and $U_{\mathbf{D}}$, such that $U_{\mathbf{F}\times\mathbf{D}}(<f,\mathrm{d}>)=U_{\mathbf{F}}(f)\times U_{\mathbf{D}}(d)$, then $<f_i,d_i>$ can be refined to a new design description $<f_{i+1},d_{i+1}>$ by first refining f_i using $U_{\mathbf{F}}$ and then refining d_i using $U_{\mathbf{D}}$. (When $d_0=\phi$, we need the synthesis operation to obtain d_1.) The effect of the closure operation $U_{\mathbf{F}}$ (similarly $U_{\mathbf{D}}$) can also be viewed as applying $U_{\mathbf{F}}$ to the *projection* of the pair $<f_i,d_i>$ onto its f-coordinate (similarly d-coordinate).

The basic design process presented in Fig. 2 is a special case of the real design model presented above. Indeed, the basic design process can be described as a sequence of design descriptions $<f_0,\phi>$, ... $<f_n,\phi>$ followed by a synthesis mapping that leads to $<f_{n+1},d_0>$, and a sequence of design descriptions $<f_{n+1},d_0>$, $<f_{n+1},d_1>$, ... $<f_{n+1},d_m>$. For example, $<f_{n+1},\phi>$is generated from $<f_i,\phi>$, where f_{n+1} is selected from the closure $U_{\mathbf{F}}(f_i)$. This description is illustrated in Fig. 5, where it is shown that a basic design process (solid line) follows a specific pattern, i.e., a sequence of operations in the function space, and is followed by synthesis and a sequence of operations in the structure space. In general, however, we cannot perform such decoupling because design decisions are context dependent, i.e., both the current specifications and the product description affect refinement decisions. For expositional purposes (and without loss of generality), we introduce the model assuming the basic process presented in Fig. 3.

3
Topological structures for design

In this section, we present a model that attempts to cast design more formally in the framework of general topology, in particular closure spaces (Cech 1966). In this paper, we have considered general topology as a mathematical base for our discussion. By placing richer structures on a topological space, additional spaces may be obtained (e.g., especially metric spaces). Here we follow a

"least committed" research approach, where some of these extended spaces will warrant continued attention over time, and will be introduced as they arise naturally in applications. For brevity and clarity, some of the formal definitions and theorems are excluded (they are presented in Braha and Maimon 1998), and concepts are presented intuitively.

3.1 Structure and function spaces

The structure and function spaces are defined as follows:[3]

Definition 1

A structural description d is defined by its observable or otherwise measurable attributes. The attributes may be of several types, e.g., physical or geometrical. A structural description has respective values for its attributes. For designers, the structure space is the set of *all* structural descriptions they can generate with their present knowledge; this is denoted by **D**.

Definition 2

A functional property is the behavior that an artifact displays when it is subjected to a situation. The collection of all functional properties observed in different situations is the functional description of the artifact. The function space is the set of *all* functional descriptions, and is denoted by **F**. A functional description $f \in \mathbf{F}$ is also referred to as a *design specification* list since it designates the sought-for functionality of the artifact.

In a similar way, we could introduce into the formulation additional aspects beyond **D** and **F**, such as behavior, as desired by a particular design approach. The next definition formalizes the intuitive notion of proximity in the structure and function spaces. The definitions are provided for the related function space. Similar definitions can be provided for a structure space or coupled space (discussed in Sect. 2.3).

3.2 Closure spaces

Definition 3 (Closure operation and function space)

If $U_{\mathbf{F}}$ is a function that maps a set of functional descriptions to a new set of functional descriptions (i.e., $U_{\mathbf{F}}$ is a set function that maps elements of $2^{\mathbf{F}}$, denoting the power set of F, into $2^{\mathbf{F}}$), then we say that $U_{\mathbf{F}}$ is a *closure operation* (or closure) for **F**, provided that the following conditions are satisfied:

1. $U_{\mathbf{F}}(\emptyset)=(\emptyset)$,
2. $F \subseteq U_{\mathbf{F}}(F)$ for each $F \subseteq \mathbf{F}$, and
3. $U_{\mathbf{F}}(F \cup G)=U_{\mathbf{F}}(F) \cup U_{\mathbf{F}}(G)$, for each $F \subseteq \mathbf{F}$ and $G \subseteq \mathbf{F}$.

The structure $<\mathbf{F},U_{\mathbf{F}}>$ is called a *closure function space*. A closure operation $U_{\mathbf{D}}$ for the structure space **D** is defined in a similar manner. The structure $<\mathbf{D},U_{\mathbf{D}}>$ is called a *closure structure space*; in real design, the closure would originate from the designers' knowledge or other sources.

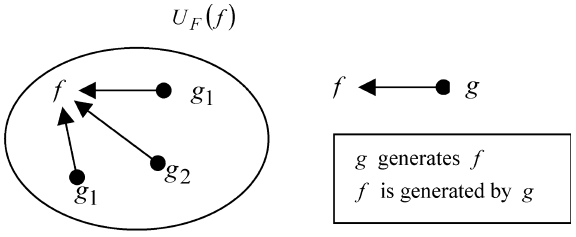
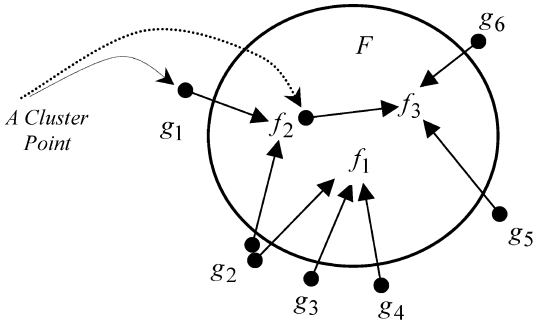


Fig. 6.



$F = \{f_1, f_2, f_3\}$

Closure of F, $U_{\mathbf{F}}(F)$:
$\{f_1, f_2, f_3, g_1, g_2, g_3, g_4, g_5\}$

Derivative of F (set of **all** cluster points):
$\{f_2, g_1, g_2, g_3, g_4, g_5\}$

Fig. 7.

Discussion

If $F=\{f\}$, then every functional description g in the closure of f (i.e., such that $g \in U_{\mathbf{F}}(f)$) is termed as a generator of f (see Figure 6). Alternatively, we say that g generates f, or that f is generated by g. In general, for any subset $F \subseteq \mathbf{F}$, any functional description f in the closure of F (i.e., such that $f \in U_{\mathbf{F}}(F)$) is termed as a generator of F.[4] Intuitively, f is a generator of F if f generates *some* functional description in F as illustrated in Fig. 7.

Generally speaking, the relation "generated by" is associated with the relation "refined by". That is, as related to the process presented in Fig. 2, the progression from a specification list f_i to a refined specification list f_{i+1} implies that the specification list f_i is generated by the refined specification list f_{i+1}, or that f_{i+1} generates f_i. In formal terms, the refined specification list f_{i+1} is included in the closure of f_i.

As discussed in Sect. 2.1, in reality there are sometimes several parallel candidate developments starting from the initial specification list f_0 (see Fig. 3). Applying Definition 3, this situation may be described as follows. The designer

[3] Some definitions may resemble GDT terminology (Yoshikawa 1981); nevertheless, as already discussed, GDT is only a special case of our framework.

[4] Note that we use the term "f is a generator of F" although f may only generate part of F.

starts by creating the closure of f_0, $U_{\mathbf{F}}(f_0)=F$. Creating the entire closure may be an expensive operation. Therefore, the designer may create part of the closure, and make a decision based on the partial knowledge regarding the complete closure. Each specification list in F generates f_0. While creating the closure, the designer may select and proceed with any refined specification list in the part of F thus far created. The next refinement step involves finding the closure of the selected refined specification list. This step is repeated until no further refinements are possible (more detail regarding this later). Thus, F denotes the set of *all* possible refined specification lists of f_0. Similarly, the closure of F, $U_F(F) = \bigcup_{f \in F} U_F(f)$, denotes the set of *all* possible refined specification lists in the second and subsequent stages. A *trajectory* of a refinement process is created by selecting a single specification list from each closure in each refinement stage (see Fig. 3). In real design, one or several trajectories are explored, and the complete **F** or $U_{\mathbf{F}}(F)$ are not exhaustively created due to limited resources such as time, money, or knowledge.

Having examined the meaning of the closure concept, let us explore its structure. First, it is clear that F is included in its closure. The closure of F may be composed of the following three categories:

1. Specification lists that are included in F such that each one does not generate any other specification lists in F (except for themselves).
2. Specification lists that are included in F such that each one generates specification lists in F (other than themselves).
3. Specification lists that are *not* included in F, and such that each one generates specification lists in F.

Each specification list that falls within the second and third categories above is called a *cluster* point, and the set of all cluster points is called the *derivative* of F as illustrated in Fig. 7. The specification lists that are included in the derivative of F, and that are not included in F (class 3), represent the *new explicit information* that is gained by applying diverse knowledge during the refinement process. Thus, the closure of a set F is the union of F with its set of cluster points. The formal definition of a cluster point and derivative of a set is presented in Appendix 1.

Often, solving a design problem is a collaborative effort carried out by several designers, each possessing different knowledge. The notion of a closure may be useful in interpreting some aspects of collaborative design. The following example illustrates a simple collaborative scenario. Assume that there are two designers, Alice and Bob, working collaboratively on a certain problem. Bob and Alice each carry out (independently) few parallel candidate developments based on his/her private knowledge. At some point in the refinement process, Bob and Alice share their intermediate results F_n^{Bob} and F_m^{Alice}, respectively. Each set represents a set of possible refined specification lists (starting from f_0). If F_n^{Bob} and F_m^{Alice} intersect, they can continue together to refine the intersection; otherwise, they will have to collaborate together to find refinements within a closure created by their mutual knowledge. In some cases, the collaboration means creating shared terminology, effectively realizing that their closure structures overlap. In other cases, there are true gaps in their knowledge that mandate collaborative learning. The intersection of sets or the closure created by mutual knowledge is more focused than each of the sets or closures separately. The usefulness of collaborative design may be seen through the above topological illustration.

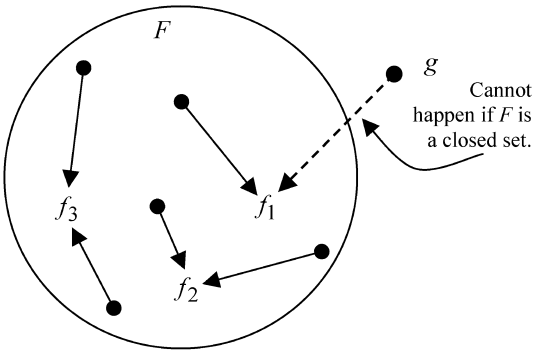


Fig. 8.

Definition 4 (Closed sets)
A subset F of a closure function space $<\mathbf{F},U_{\mathbf{F}}>$ is called *closed* if $U_{\mathbf{F}}(F)=F$. Closed sets in a closure structure space $<\mathbf{D},U_{\mathbf{D}}>$are defined in a similar manner.

Discussion
Intuitively, a set F is closed if every element in F is generated only by elements in F as shown in Fig. 8. As explained above, the set of all possible refined specification lists (starting from f_0) in a particular stage of refinement can be defined recursively as follows:[5]

1. $F_0=\{f_0\}$
2. $F_1=U_{\mathbf{F}}(F_0)$
3. $F_{i+1}=U_{\mathbf{F}}(F_i)$

If, at some stage, the derived set of refined specification lists F_n is a *closed* set[6], then we are assured that every specification list in F_n is generated only by specification lists in F_n. Thus, no further refinement stage that yields *new refined specification lists* (or new information) is possible. Formally, it means that $U_{\mathbf{F}}(F_n)=F_n$. In a real situation, this means that there is not enough information to further continue design exploration in the function space. Therefore, the refinement process stops and *synthesis* must begin (see Fig. 1). Prior to performing the synthesis operation, the designer searches within F_n for a suitable refined specification list that can be mapped to structural descriptions in the structure space. The refinement process in the structure space then begins (see Fig. 1). It is important to note that not every refinement process will end with a closed set in a finite number of steps. However, if at some point a closed set is found, then the refinement process stops.

[5] It could be prohibitively expensive to generate this set in real design.
[6] It may be impossible to verify this property in real design; however, engineering intuition, experience, or lack of knowledge might imply it.

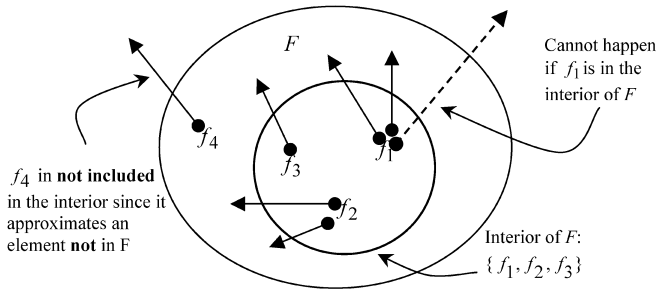


Fig. 9.

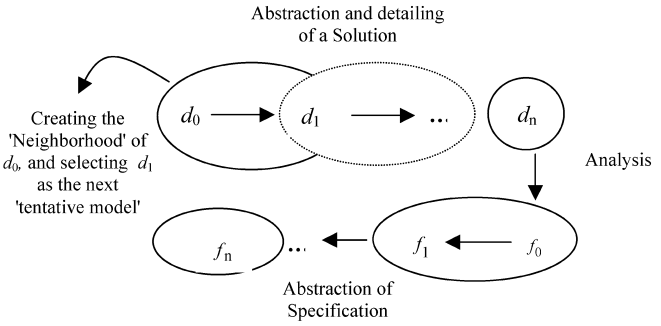


Fig. 10.

As stated above, the set of all possible refined specification lists (starting from f_0), at any particular stage of refinement, can be defined recursively as $F_{i+1}=U_{\mathbf{F}}(F_i)$. It can be shown that, after a long time (even an infinite number of steps), repeated application of the closure operation $U_{\mathbf{F}}$ will result in a closed set. That is, $\lim_{i\to\infty} F_i$ is a closed set. Therefore, every refinement process *must* terminate at some point (though not necessarily at a usable or good specification).

3.3
Interior, neighborhood, and the analysis process

For any set F, the *interior* of F is defined as the set of descriptions in F that generate descriptions only in F. For example, the set $\{f_1,f_2,f_3\}$ in Fig. 9 is the interior of F. The description f_4 is not included in the interior of F since it generates some description that is not in F. If we verify that the functional description f is included in the interior of a set G, then G is a *neighborhood* $\mathrm{N}(f)$ of the functional description f. In general, if a set F is included in the interior of a set G, then G is a neighborhood of F. The formal definitions are provided in Appendix 1.

Discussion

The definitions of interior and neighborhood are useful for interpreting certain analysis activities as follows. The designer starts with a candidate design solution d_0 that needs to be analyzed. Sometimes, d_0 cannot be analyzed directly (e.g., by performing finite-element analysis), since its structural description is not provided in a form suitable for analysis. To overcome this problem, the designer creates a series of successive design descriptions, such that each design description in this "implication"[7] chain is implied by the design description that precedes it. After a suitable structural description d_m is obtained, the designer is able to analyze the object. This *analysis* operation, Γ, is expected to result in the behavior of the product f_0. A similar implication process is followed for the function space until a comprehendible behavior for the product is obtained. For example, in finite element analysis, the initial design description is broken down into an analysis model d_m. This includes a series of abstraction steps as well as detailing steps for preprocessing the product information, which is required by the underlying analysis procedure. The design description d_m is analyzed in order to extract the detailed design behavior f_0, which undergoes postprocessing to result in a compressed, user-friendly result f_n.

How can the above analysis process be described using our topological concepts? This is done by equating the relation "d_{i+1} is implied by d_i" with "d_{i+1} is generated by d_i" or "d_i generates d_{i+1}".[8] Therefore, if $\mathrm{N}(d)$ is a neighborhood of structural description d, then by the neighborhood definition every structural description that is implied by d (or "is generated by d") must be in $\mathrm{N}(d)$. By associating a neighborhood with each structural description d (similarly functional description f), the analysis process can be described as shown in Fig. 10. Each circle in Fig. 10 represents a neighborhood of the current description (functional or structural). In general, the neighborhood of a description can contain more than one element. In this case, the designer explores the neighborhood for descriptions that the current description generates. Since every description that is generated by f is guaranteed to be in its neighborhood, the designer can focus her exploration efforts on the neighborhood of the current description (whose cardinality is much smaller than all possible descriptions). A *unique* analysis process is obtained by selecting, at each stage, a single description out of the many possible neighborhood descriptions.

3.4
The relationship between neighborhoods and closures

Two main topological concepts have been introduced thus far. The closure operation serves as the underlying concept for design synthesis, while the neighborhood concept is used for modeling design analysis. We anticipate that they have a relationship given the similarity between Figs. 10 and 2. However, design analysis is supported by employing neighborhood processes (Fig. 10), whereas design synthesis is driven by closure procedures (Fig. 2). The following simple but important theorem shows that closures are completely determined by neighborhoods (see also Fig. 11).

Theorem 1

A functional property $f\in\mathbf{F}$ belongs to the closure of a subset F of a space $<\mathbf{F},U_{\mathbf{F}}>$ if and only if each neighborhood of f in $<\mathbf{F},U_{\mathbf{F}}>$ intersects F.

[7] Note that our use of the term "implication" is not necessarily identical to "logical implication". In a logical framework, the implication relation is associated with deduction. Consequently, by *modus ponens,* every description in the implication chain is implied by the initial candidate solution d_0.

[8] <fn>Recall that "d_{i+1} is generated by d_i" or "d_i generates d_{i+1}" if $d_i\in U_{\mathbf{F}}(d_{i+1})$.

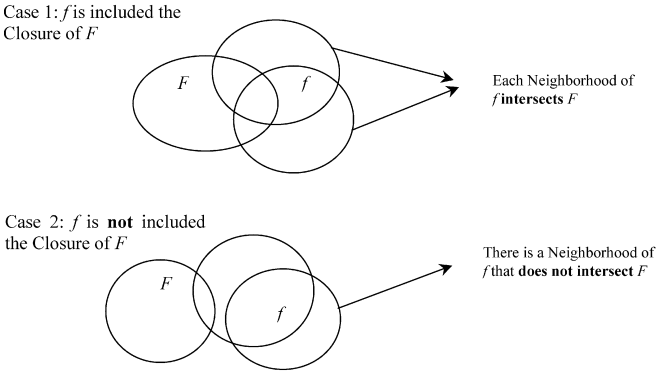


Fig. 11.

Discussion

As shown above, a closure space can be constructed by directly providing the closure operation (see Definition 3). This requires that for each description, the designer has knowledge regarding the set of descriptions that generate it. Theorem 1 suggests an alternative way of constructing a closure space by using neighborhoods. In reality, creating the entire neighborhood system for each description may be an expensive operation, in which case the following approximate procedure may be employed:

Algorithm **Approximate** *Closure*
Input: functional property f, collection of neighborhoods
Output: The approximate closure of f
begin
for several $g \in F$ do
begin
if most neighborhoods of g contain f *then*
add g to the closure of f
end
return approximate closure of f
end

An approximate closure accounts for real, imperfect, partial, or incorrect knowledge; thus, it may help to explain design failures. In contrast, limited resources can explain suboptimal designs. Studying the nature of approximate closures is a subject for future work.

It is also possible to construct the neighborhood of any design description by utilizing the closure operation. Indeed, neighborhoods have been defined using the interior concept (see Appendix 1). The interior of a set, in turn, has been defined using the closure operation (see Appendix 1). This quality, together with Theorem 1, renders the neighborhood and closure concepts "dual".

3.5 Open sets

Definition 5 (Open sets)

A subset F of a closure function space $<\mathbf{F},U_{\mathbf{F}}>$ is called *open* if its complement (relative to **F**) is closed, i.e., if $U_{\mathbf{F}}(\mathbf{F}-F)=\mathbf{F}-F$. Open sets in a closure structure space $<\mathbf{D},U_{\mathbf{D}}>$ are defined in a similar manner.

Discussion

A set is called open if every description in the set generates only descriptions that exist within the set. It was shown that closed sets play an important role in synthesis processes. Specifically, if at some point a closed set is found, no further stage of refinement that yields new refined specification lists (or new information) is possible, and the refinement process stops. This situation can trigger knowledge acquisition in an attempt to continue the refinement or synthesis processes. The open set concept has a similar role in the context of analysis processes. That is, if at some stage of "implication", the set of all structural descriptions D_n that can be generated from d_0 is an open set, then we are assured that every description in this set generates descriptions that already exist within the set. In this case, the implication process stops and the analysis operation may be applied as shown in Fig. 10. This is the ultimate analysis that uncovers the underlying behavior of the product. Prior to performing the analysis operation, the designer explores within D_n for a suitable structural description that can be mapped into functional descriptions in the function space. This is followed by a derivation process in the function space. It is important to note that not every derivation process will end with an open set. However, such an occurrence will guarantee the termination of the derivation process.

4 Putting the framework to use

In order to illustrate the usefulness of the framework and explain the various concepts, three examples are presented. The first example shows that General Design Theory (GDT, see Yoshikawa 1981; Tomiyama and Yoshikawa 1987; Reich, 1995) is a special case of our framework. The second example clarifies the various concepts through a simple knowledge-based design system. The third example demonstrates how the machine learning system ECOBWEB (Reich and Fenves 1992) can be elucidated in our framework.

These three examples follow the same structure:

1. Given a particular context, interpret the closure and neighborhood concepts in a way suitable to the context.
2. Collect the knowledge to realize the interpretation.
3. Execute design situations.

The three examples are very different, demonstrating the diversity of situations that can be understood with the framework. These examples certainly are not indicative of the scope of the framework.

4.1 General Design Theory (GDT) as a special case

4.1.1. Topological closure spaces

GDT is defined in terms of the set of all real artifacts that did exist, do exist, and will exist **S**; and a collection of subsets (called *abstract concepts*) of the space **S** that satisfy the axioms of point-set topology (Croom 1989). For example, in the domain of chairs (Reich 1995) **S** includes

eight chairs as depicted in Fig. 12. One abstract concept could be chairs that *constrain back* defined as the set {*B*,*C*,*F*,*G*}. Another concept could be a *movable* chair consisting of {*A*,*E*,*F*,*G*,*H*}.

Let us call <**S**,$\mathcal{J}$> the Yoshikawa–Tomiyama space (**Y–T** space). According to GDT, a design process is a mapping between a **Y–T** function space and a **Y–T** structure space (see Reich 1995[9]). For example, the specification of a chair that will be movable and constrain back leads to two potential designs, F and G. These designs can be generated in two ways. The first way starts with {*A*,*E*,*F*,*G*,*H*} as the movable designs and refines them (by set intersection) with the constrain back property. The second way starts with {*B*,*C*,*F*,*G*} as the constrain back designs and refines them with the movable property. Note that the refinement process was made easier by the use of the eight representative chairs as mediators between the specification and the design description. In the absence of these chairs, the process might have been more difficult.

In the following, we show that the GDT's main assumption regarding design knowledge being a "point-set topology" is a special case of our main concept of closure spaces. To this end, we show that for every **Y–T** space there corresponds a unique closure space, and that the **Y–T** space can be derived from this unique closure space. We also demonstrate how to construct the unique closure space. This can be done without knowing all the entities in advance.

First, it can be shown that the open sets (Definition 5) of any closure space <**S**,U_s> satisfy three conditions, which are the axioms defining a point-set topology for **S** (Braha and Maimon 1998). Next, we ask the following question: given a **Y–T** space, is there a *unique* closure space <**S**,U_s>, such that the open sets of <**S**,U_s> coincide with the sets in the **Y–T** space? In general, the answer to this question is negative. That is, for any **Y–T** space, there may correspond many closure spaces where the open sets of each closure space coincide with the **Y–T** space. However, if we focus on a special class of closure spaces (called *topological closure spaces*, see below), then it can be shown that for any **Y–T** space there corresponds a unique topological closure space (Braha and Maimon 1998). In other words, if the open sets of two topological closure spaces coincide with the sets of some **Y–T** space, then the two topological closure spaces are *identical* (in the sense that both have the same closure operation). Therefore, **Y–T** spaces correspond to a *subset* of closure spaces. This quality makes closure spaces more general than **Y–T** spaces. How, then, is a topological closure space defined?

A topological closure space is a closure space <**F**,$U_{\mathbf{F}}$> where each $U_{\mathbf{F}}(F)$ is a closed set (see Definition 4). That is, if we apply a refinement process starting from the initial specification list f_0 (see Fig. 2), then any refinement process ends in a *single* step! The reasoning is as follows: starting from f_0, we create the closure $U_{\mathbf{F}}(f_0)$ of f_0. Next, we find the closure $U_{\mathbf{F}}(U_{\mathbf{F}}(f_0))$ of $U_{\mathbf{F}}(f_0)$. However, since <**F**,$U_{\mathbf{F}}$> is a topological closure space, $U_{\mathbf{F}}(f_0)$ is a closed set, and thus $U_{\mathbf{F}}(U_{\mathbf{F}}(f_0))=U_{\mathbf{F}}(f_0)$. Consequently, no further refinement is possible and the refinement process terminates. This type of design process is rather unrealistic; nevertheless, it is also a consequence of GDT axioms (Reich 1995). Next, we show how to construct the topological closure space out of a **Y–T** space.

4.1.2
Constructing a topological closure space from a Y–T space

First, we need to define two concepts related to **Y–T** spaces (not to closure spaces): (1) *limit point* (also called cluster point) of a subset *S*; and (2) *closure* of a subset *S* (not to be confused with the closure operation of a closure space).

Definition 6

Let <**S**, $\mathcal{J}$>be a **Y–T** space. An entity s^* (real artifact) in **S** is a limit point of a set $S \subseteq \mathbf{S}$ if every abstract concept in containing s^* contains a point of *S* distinct from s^* (see Fig. 13). The set of limit points of *S*, *S*′, is called the derived set of *S*. The *closure* $\bar{S}$ of *S* is the union of *S* with its set of limit points, i.e., $\bar{S} = S \cup S'$.

Discussion

An entity s^* is a limit point of a set $S=\{s\}$ if every property of s^* is also a property of *s* (recall that in GDT properties are abstract concepts). In general, an entity s^* is a limit point of a set *S* if every property of s^* is also a property of some entity in *S*. Having defined the closure of a set in a **Y–T** space, the closure operation $U_{\mathbf{F}}$ in the corresponding topological closure space is defined simply as $U_{\mathbf{F}}(S)=\bar{S}$. The structure space is defined similarly. It can be shown that the above construction satisfies three conditions, which are the axioms defining a closure operation (Braha and Maimon 1998). The closure operation can be used in design execution. For instance, starting with the specification "a chair that will be *movable* and *constrain back*," the designer starts with a Scandinavian chair (chair E in Fig. 12) that is known to be movable (but not to constrain back), and explores

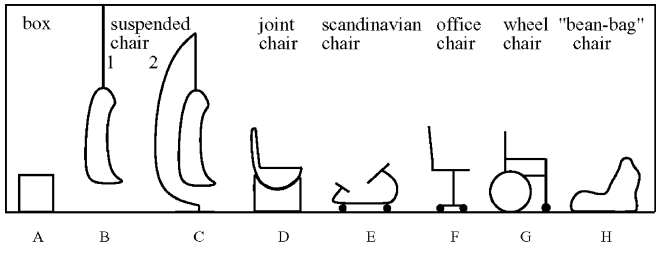


Fig. 12.

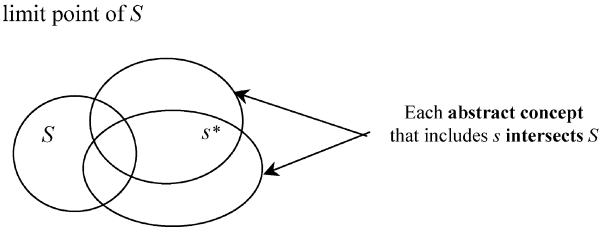


Fig. 13.

[9] We use this source rather than the original papers since it summarizes GDT with simple intuitive examples.

similar chairs in the closure of E, U_F({Scandinavian Chair}). This step will obtain all the chairs that share similar functional properties with E. In particular, if there is a chair that is both movable and constrain back, it will be included in the closure of E (e.g., office chair and wheel chair). The designer searches within the closure U_F ({Scandinavian Chair}) for a suitable design solution (e.g., wheel chair) and then performs the synthesis operation that uncovers the structural description of the wheel chair in the structure space.

To summarize: (1) the closure of a subset in a point-set topology is the particular interpretation of the closure operation in a related topological closure space; (2) entities are the knowledge elements; and (3) design processes are executed as described above as a consequence of the closure interpretation.

4.2
Rule-based design

This section presents a simple rule-based design example in order to illustrate the topological concepts presented in this paper. Rule-based systems are useful for this illustrative purpose since their two methods of control—forward chaining and backward chaining—can be construed as forms of analysis and synthesis.

Using forward chaining, we start with the known facts in the knowledge base and generate new conclusions that in turn can allow more inferences to be made (Russell and Norvig 1995). This kind of deductive inference is useful in design analysis, where the task is to uncover the behavior displayed by the artifact when it is subjected to various situations.

Alternatively, the strategy of backward chaining is to begin with the functional requirements we want to achieve, and then attempt to accomplish their conditions (new functional requirements). In backward chaining, the search process is repeated recursively when there is a goal (functional requirement) to be achieved. This is done by selecting and applying transformation rules to a candidate unsatisfied goal. The search process terminates whenever a set of functional requirements is identified, which can be matched to known structural attributes. If incompatibilities (e.g., violated constraints) happen in later design phases, then backtracking is applied and the inference engine generates another set of unsatisfied goals. Design synthesis, which is the task of finding a structure given a functional requirement, is likely to utilize backward chaining rather than forward chaining (e.g., Takeda et al 1990). In order to automate a backward chaining inference engine, it is important to generate "effective" subgoals in a controlled manner as well as employing efficient backtracking methods in order to deal with the enormous search space of possible goals (Chandrasekaran 1990).

4.2.1
Rule-based design and closure spaces

Let $<f_i,d_i>$ be all the information (at any stage of the solution process) about the problem being solved, where f_i and d_i are the current functional and structural descriptions, respectively. By utilizing a backward chaining inference mechanism (also called abduction), a new design description $<f_{i+1},d_{i+1}>$is obtained. The descriptions $<f_i,d_i>$ and $<f_{i+1},d_{i+1}>$ are related via the logical formula $<f_{i+1},d_{i+1}> \rightarrow <f_i,d_i>$, where $<f_{i+1},d_{i+1}>$ is referred to as the *antecedent* and $<f_i,d_i>$ as the *consequent*[10]. The above logical implication is performed with respect to the underlying designer's knowledge.

Backward chaining can be interpreted using our topological concepts as follows. For any two expressions a and b, if $b \in U(a)$ then $b \rightarrow a$. That is, the closure of any expression a includes expressions that logically derive the expression a. By applying the above construction, rule-based design can be cast into our design process framework. The designer starts by creating part of the closure of the initial design description $<f_0,d_0>$. This is performed by applying one or more production rules in the knowledge base (see example below). The designer selects any description in $U_{F\times D}(<f_0,d_0>)$, finds part of the closure of the selected design description, etc.

The interpretation of abduction in terms of closures may be extended to other topological notions. For example, a set A is closed if every logical expression in A is *logically derived*[11] only by expressions in A. The interior of a set A includes expressions in A that logically derive only expressions in A. A set A is a neighborhood of an expression a if the expressions that are logically derived by a are included in A. Thus, if a is a known fact (e.g., "the stretched area is 5×7 cm"), then any expression that is deduced by a (e.g., "the tensile stress is high") is included in a neighborhood of a. This kind of deductive inference is useful in analysis. A set A is open if every expression in A logically derives only expressions in A. Thus, if an open set A includes known facts, no new fact can be derived from the known facts in A.

In the following, the problem of designing an automobile using a rule-based system is used to explain the concepts of the topological model.

4.2.2
Automobile design example

Let the structural and functional descriptions be specified in terms of a list of properties. The structural properties that specify the configuration of actual cars as well as the functional properties that are manifested by actual cars are presented in Table 1a and Table 1b, respectively. A small sample of the domain-specific knowledge relevant to the car design domain is expressed in terms of the production rules presented in Appendix 2.

Assume that the designer is faced with the problem of designing a car that is able to achieve the following functional attributes as requirements:

1. The car creates minimal pollution (f^3).
2. The car is capable of high driving speed (f^{10}).
3. The car has low fuel consumption (f^{12}).
4. The car is safe (f^{18}).

[10] Here, in any formula of the form $A \rightarrow B$, A is referred to as the *antecedent* and B as the *consequent* (Russell and Norvig 1995).
[11] By applying a specific knowledge base.

Table 1a. Structural attributes for the automobile design example

Structural attributes	
(d_1) 4-wheel drive	(d_{23}) High transmission ratio
(d_2) 4-wheel steering	(d_{24}) Horn
(d_3) 6–8 cylinders	(d_{25}) Hydraulic disk brakes
(d_4) Absorbent front end	(d_{26}) Large pistons & cylinders
(d_5) Air bag	(d_{27}) Light weight
(d_6) Air-cooled engine	(d_{28}) Liquid cooling system
(d_7) Air deflector	(d_{29}) Low & small structure
(d_8) An engine that deflects down	(d_{30}) Muffler
(d_9) Antilock braking system (ABS)	(d_{31}) Power brakes
(d_{10}) Automatic belts	(d_{32}) Powerful starter
(d_{11}) Catalytic converter	(d_{33}) Radial tire
(d_{12}) Deep thread patterns	(d_{34}) Richer mixture fuel
(d_{13}) Disconnecting fan system	(d_{35}) Rigid passenger compartment
(d_{14}) Drum brakes	(d_{36}) Stabilizers in the front
(d_{15}) Electric powered	(d_{37}) Suspension system
(d_{16}) Electronic ignition	(d_{38}) Tubeless tire
(d_{17}) Extra differential	(d_{39}) Windshield defroster
(d_{18}) Extra strong door	(d_{40}) Windshield washer & wiper
(d_{19}) Extra-strong roof	(d_{41}) Tire with 205 width symbol
(d_{20}) Fog lights	(d_{42}) Tire with 155 width symbol
(d_{21}) Fuel injection	(d_{43}) Tire with 60 aspect ratio symbol
(d_{22}) High ground clearance	(d_{44}) Tire with U speed symbol

Table 1b. Functional attributes for the automobile design example

Functional attributes	
(f_1) Aerodynamic design	(f_{16}) Passable in difficult terrain
(f_2) Diesel engine	(f_{17}) Reliable tire
(f_3) Creates minimal pollution	(f_{18}) Safe car
(f_4) Easy parking	(f_{19}) Safe in accidents
(f_5) Efficient engine	(f_{20}) Safe in bad weather
(f_6) Economical	(f_{21}) Safe in flipping over
(f_7) Reliable brakes	(f_{22}) Safe in head-on collisions
(f_8) Heavy car	(f_{23}) Safe at high driving speed
(f_9) High power output	(f_{24}) Safe in off-highway road
(f_{10}) High driving speed	(f_{25}) Safe in poor external conditions
(f_{11}) High-volume combustion chamber	(f_{26}) Safe in poor visibility
(f_{12}) Low fuel consumption	(f_{27}) Safe in side collisions
(f_{13}) Low maintenance costs	(f_{28}) Small car
(f_{14}) Mechanically dependable and durable	(f_{29}) Small engine
(f_{15}) Off-highway tire	(f_{30}) High-powered engine

The following "facts" are assumed: (1) the car is used for family driving, (2) the external conditions are good, (3) the maximum allowed speed is 160 km/h, and (4) the car is used for urban driving.

The following constraints[12] are further considered: (1) the structural attributes "tire with 205 width symbol (d^{41})" and "tire with 155 width symbol (d^{42})" cannot be included together in a design description, and (2) the structural attribute "the car is electric-powered (d^{15})" cannot be included in a design description if the functional attribute "the car has DIESEL ENGINE (f^2)" is satisfied.

The rule-based design system needs to search through the problem space for a path from the initial requirements to some state of the car's structural description such that the requirements f^{18}, f^{10}, f^{12}, f^3 are achieved, and while adhering to the compatibility constraints.

In attempting to achieve the initial requirements, a backward chaining inference with depth-first control strategy is used by the rule-based system. In addition, if any of the above constraints are violated during the search, *dependency-directed* backtracking[13] (Brown and Chandrasekaran 1989) is utilized and an alternative rule is chosen from the list of available finite choices.

In the example, the design description is a conjunction of structural and functional attributes (if a functional attribute appears in a process state it means that it remains to be satisfied). Table 2 shows the trace of "process descriptions" $<f_i,d_i>$ generated in the course of searching for a solution to the automobile design problem. As mentioned above, $<f_{i+1},d_{i+1}>$ is a description in the closure of $<f_i,d_i>$. At each step of the search the consequent parts of the production rules (listed in Appendix 2) are matched against the current unsatisfied functional attributes and known facts. The current unsatisfied functional attributes in the design description $<f_i, d_i>$ are given by f_i as shown in the second column of Table 2. Since many production rules may match the current description, the preferred rule is the one that matches the first leftmost unsatisfied functional attribute (a depth-first control strategy). Topologically speaking, the designer selects a refined description in the closure of $<f_i,d_i>$ according to the above matching heuristic procedure.

To summarize: (1) the concept of logical abduction is used as an interpretation of the closure operation; (2) a rule base constitutes the knowledge; and (3) design and analysis processes are executed as backward and forward inferences, respectively.

4.3 ECOBWEB

ECOBWEB (Reich and Fenves 1992) is a machine learning system that creates a classification hierarchy from design examples that are represented by lists of property-value pairs including functional and structural properties. Given a partial design description (which could include only a part of the functional specifications) and a previously learned classification hierarchy, ECOBWEB can synthesize a number of candidate designs from the hierarchy. The candidates are complete descriptions of designs consisting of all functional and structural descriptions. ECOBWEB has several design strategies. It has been previously shown that ECOBWEB approximates the topological structure of GDT (Reich 1990). Now we illustrate how ECOBWEB's

[12] Constraints in *discrete* domains can be expressed as compatibility relations between attributes, stating that certain combinations are allowed or not.

[13] Dependency-directed backtracking provides a way of taking into account the information about which pieces of knowledge contribute to the failure. This information is used in the decision of how far to backtrack.

Table 2. A "coupled" design process for the automobile design problem (complete table may be found in Braha and Maimon 1998)

Process step	Design description < *f*,d>		Candidate rules	Selected rule
	Structural description and "open" functional attributes	Satisfied functional attributes		
1	f_{18} **and** f_{10} **and** f_{12} **and** f_3	$\varnothing$	Rule 4	Rule 4 is fired to decompose f_{18}: "the car is SAFE"
2	f_{19} **and** f_{10} **and** f_{12} **and** f_3	f_{18}	Rule 6	Rule 6 is fired to decompose f_{19}: "the car is SAFE IN ACCIDENTS"
3	f_{22} **and** f_{27} **and** f_{21} **and** d_{10} **and** f_{10} **and** f_{12} **and** f_3	$f_{19}$and f_{18}	Rule 11	Rule 11 is fired to decompose f_{22}: "the car is SAFE IN HEAD-ON COLLISIONS"
4	d_4 **and** d_5 **and** d_8 **and** f_{27} **and** f_{21} **and** d_{10} **and** f_{10} **and** f_{12} **and** f_3	f_{22} **and** f_{19} **and** f_{18}	Rule 12	Rule 12 is fired to decompose f_{27}: "the car is SAFE IN SIDE COLLISIONS"
11	d_4 **and** d_5 **and** d_8 **and** d_{18} **and** d_{19} **and** d_{35} **and** d_{10} **and** d_{29} **and** d_7 **and** d_{26} **and** d_3 **and** d_{32} **and** d_{28} **and** d_{34} **and** d_{30} **and** d_{23} **and** d_{41} **and** d_{43} **and** d_{44} **and** f_{12} **and** f_3	f_2 **and** f_{11} **and** f_{30} **and** f_1 **and** f_{10} **and** f_{21} **and** f_{27} **and** f_{22} **and** f_{19} **and** f_{18}	Rule 21, Rule 38	Rule 38 is used to decompose f_{12}: "the car has LOW FUEL CONSUMPTION"
12	d_4 **and** d_5 **and** d_8 **and** d_{18} **and** d_{19} **and** d_{35} **and** d_{10} **and** d_{29} **and** d_7 **and** d_{26} **and** d_3 **and** d_{32} **and** d_{28} **and** d_{34} **and** d_{30} **and** d_{23} **and** d_{41} **and** d_{43} **and** d_{44} **and** f_5 **and** d_{13} **and** d_{42} **and** f_3	f_{12} **and** f_2 **and** f_{11} **and** f_{30} **and** f_1 **and** f_{10} **and** f_{21} **and** f_{27} **and** f_{22} **and** f_{19} **and** f_{18}		Backtracking The structural attributes d_{41} ("tire with 205 width symbol") **and** d_{42} ("tire with 155 width symbol") cannot be included together in a design description
13	d_4 **and** d_5 **and** d_8 **and** d_{18} **and** d_{19} **and** d_{35} **and** d_{10} **and** d_{29} **and** d_7 **and** d_{26} **and** d_3 **and** d_{32} **and** d_{28} **and** d_{34} **and** d_{30} **and** d_{23} **and** d_{41} **and** d_{43} **and** d_{44} **and** f_{12} **and** f_3	f_2 **and** f_{11} **and** f_{30} **and** f_1 **and** f_{10} **and** f_{21} **and** f_{27} **and** f_{22} **and** f_{19} **and** f_{18}	Rule 21, Rule 38	Rule 21 is used to decompose f_{12}: "the car has LOW FUEL CONSUMPTION"
18	d_4 **and** d_5 **and** d_8 **and** d_{18} **and** d_{19} **and** d_{35} **and** d_{10} **and** d_{29} **and** d_7 **and** d_{26} **and** d_3 **and** d_{32} **and** d_{28} **and** d_{34} **and** d_{30} **and** d_{23} **and** d_{41} **and** d_{43} **and** d_{44} **and** d_{16} **and** d_{21} **and** d_{13} **and** d_{27} **and** d_{11}	f_3 **and** f_5 **and** f_{12} **and** f_2 **and** f_{11} **and** f_{30} **and** f_1 **and** f_{10} **and** f_{21} **and** f_{27} **and** f_{22} **and** f_{19} **and** f_{18}		**STOP** Consistent solution is obtained

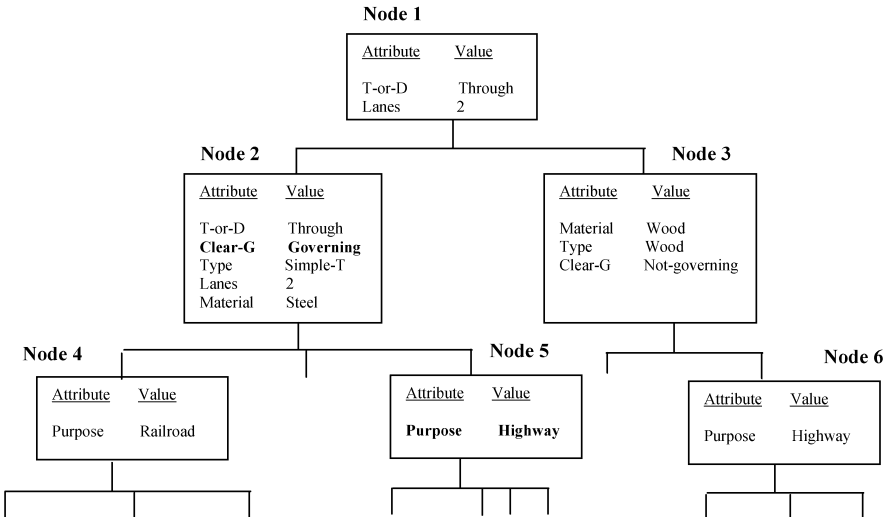


Fig. 14.

prototype-based synthesis strategy works in our framework.

Given an initial $<f_0,d_0>$ that could contain some functional specifications and a partial design description, such as design a highway bridge where vertical clearance (Clear-G) governs (see Fig. 14), ECOBWEB gradually refines it until it becomes a description containing all property-value pairs $<f_n,d_n>$. The refinement is done in stages. First, $<f_0,d_0>$ is pushed down the hierarchy until a node is found that has a characteristic value for functional or design description properties (characteristics T-or-D and Lanes in Node 1). These characteristics (if not part of the present description $<f_0,d_0>$) are assigned to the current design description. If it is a characteristic function, $<f_0,d_0>$ is transformed into $<f_1,d_1>$ where $d_{1=}d_0$. If the characteristic is a structural property, $<f_0,d_0>$ is transformed into $<f_1,d_1>$ where $f_{1=}f_0$. If the node included a mix of functional and structural characteristics, the description $<f_0,d_0>$ is transformed into $<f_{1,}d_1>$ (that is, <{Purpose Highway, Clear-G Governing, Lanes 2}, {T-or-D Through}>).

Subsequently, the design is pushed down to Nodes 2 and 5 and its design description is augmented with Type Simple-T and Material Steel while its initial specifications are matched by the functions in these nodes. The design could end here with an abstract description or, given diverse knowledge expressed in a deep hierarchy, the push down operation would terminate when all the missing design description properties have been matched with and assigned characteristic values. Limited knowledge, exemplified by a shallow hierarchy, will lead to pushing the description to a leaf node and completing the missing values from that leaf.

Implicitly, the hierarchy created by learning from examples can be interpreted as a sequence of closures. The closure operation could be defined as: if $<f_i,d_i>$ is the present description then its closure $<f_{i+1},d_{i+1}>$ includes the properties in $<f_i,d_i>$with several additional properties. ECOBWEB only approximates this closure (see Sect. 3.4); therefore, its synthesis is not guaranteed to be successful at all times. In particular, when the hierarchy is shallow, the closure approximation is coarse and failures are unavoidable. In contrast, when the hierarchy is deep, the chances of getting successful results increase (Reich 1995), even though the hierarchy is still an approximation of a closure. ECOBWEB push down operation implements a greedy search algorithm for reducing computational complexity; therefore, no optimal designs are expected.

To summarize: (1) the hierarchical structure created by learning is an approximation of closure; (2) the examples are the source of knowledge; and (3) design processes are executed according to the aforementioned procedure.

5 Discussion

We have described a mathematical framework of design processes based on closure spaces. This framework is shown to be more general than GDT. In addition, the framework has been utilized to describe the operation of rule-based systems, and a particular learning system. These results demonstrate that our model has the desirable quality of representing several, seemingly distinct, approaches as instances of the same framework. It is important to note that our framework is not suggested as a means for supporting or proving theorems for automated design.

Using simple examples throughout the paper, we also hinted at the potential for the framework to serve as a basis for a descriptive study of design. Various design phenomena such as design failure, identification of design knowledge bottlenecks, and benefits of collaborative design could be described and understood using the framework. This potential needs to be explored further.

Another consequence of the proposed model is related to design sensitivity and design robustness. More specifically, the topological concept of *continuity* can be developed in order to address the important issues of product and process robustness (Braha and Maimon 1998; Braha and Reich 2001). By "process robustness" we mean the desirable property of a design process by which a small change in the input (i.e., specifications expressed in terms of the product's functionality) results in a slight modification in the design process output (i.e., artifact's

Table 3. A sample of rules for the car example

RULE 1	If	the car has OFF-HIGHWAY TIRES **and** 4-wheel drive **and** extra differential **and** high ground clearance **and** is light weight
	Then	the car is PASSABLE IN DIFFICULT TERRAIN
RULE 2	If	the car is SAFE IN POOR EXTERNAL CONDITIONS
	Then	the car is SAFE **and** the external conditions are **not** good
RULE 3	If	the car is SAFE AT HIGH DRIVING SPEED
	Then	the car is SAFE **and** the maximum allowed speed is 200 km/h
RULE 4	If	the car is SAFE IN ACCIDENTS
	Then	the car is SAFE **and** the car is used for family driving **and** the external conditions are good **and** the maximum allowed speed is 160 km/h
RULE 5	If	the car is SAFE IN POOR VISIBILITY **and** SAFE IN BAD WEATHER **and** SAFE IN OFF-HIGHWAY ROADS
	Then	the car is SAFE IN POOR EXTERNAL CONDITIONS
RULE 6	If	the car is SAFE IN HEAD-ON COLLISIONS **and** SAFE IN SIDE COLLISIONS **and** SAFE IN FLIPPING OVER **and** has automatic belts
	Then	the car is SAFE IN ACCIDENTS
RULE 11	If	the car has an absorbent front end **and** air bags **and** an engine that deflects down
	Then	the car is SAFE IN HEAD-ON COLLISIONS
RULE 12	If	the car has extra strong doors
	Then	the car is SAFE IN SIDE COLLISIONS
RULE 21	If	the car has AERODYNAMIC DESIGN **and** an EFFICIENT ENGINE **and** a DIESEL ENGINE **and** a disconnecting fan system **and** is light weight
	Then	the car has LOW FUEL CONSUMPTION
RULE 22	If	the car has tubeless tires **and** radial tires
	Then	the car has RELIABLE TIRES
RULE 38	If	the car has AERODYNAMIC DESIGN **and** an EFFICIENT ENGINE **and** a DIESEL ENGINE **and** a disconnecting fan system **and** tires with 155 width symbol **and** tires with 60 aspect ratio symbol
	Then	the car has LOW FUEL CONSUMPTION

structural description). Process robustness supports design synthesis since a new problem, whose specifications differ slightly from the specifications of a known product, may be solved by slightly modifying the structure of the known solution (similar to case-based reasoning). This may not yield a creative design but, nevertheless, a better design. "Product robustness" refers to the positive aspect of the product where only a small change in the product's behavior occurs when the product's structure (e.g., geometry) is slightly perturbed. The robustness property of a product also guarantees that a sequence of incremental refinements to its structure (e.g., due to temporal drift, deterioration, or failure) will cause only small incremental changes to its functionality.

In the future, we intend to address several additional issues. (1) Demonstrating the utility of the proposed framework in classifying and analyzing different design methodologies, such as the conceptual design with a "function structure" (Pahl and Beitz 1984), or the systematic concept generation for technical products (Hubka and Eder 1988); (2) showing the utility of our framework in describing collaborative design, and elaborating on the consequences of design robustness as discussed above; and (3) exploiting the computational aspects (considering realistic approximation procedures) and considering the underlying topological concepts (e.g., closure and neighborhood).

Appendix 1

Cluster point, interior, and neighborhood

Definition A.1 (Cluster point of a set)
A *cluster point* of a set F in a closure function space is a point f belonging to the closure of $F-\{f\}$ The set of all cluster points of a set F is denoted by F' and called the *derivative* of F in the closure function space. Clearly, the closure of a set F is the union of the *set* F with its set of cluster points.

Definition A.2 (Interior of a set)
With any closure $U_{\mathbf{F}}$ for a set $\mathbf{F}$ there is associated the interior operation $int_{U\mathbf{F}}$, denoted briefly by *int*. The operation *int* is a function that maps elements of $2^{\mathbf{F}}$ into $2^{\mathbf{F}}$, such that for each $F \subset \mathbf{F}$, $int(F)=\mathbf{F}-U_{\mathbf{F}}(\mathbf{F}-F)$. The set $int(F)$ is called the *interior of* F in $<\mathbf{F},U_{\mathbf{F}}>$.

Definition A.3 (Neighborhood)
A *neighborhood* of a subset F of a closure function space $N(F)$ is any subset G of $\mathbf{F}$ containing F in its interior. By a neighborhood, $N(f)$, of a functional description f of $\mathbf{F}$, we mean a neighborhood of the one-point set $\{f\}$. The *neighborhood system of a set* $F \subseteq \mathbf{F}$ (a point $f \in \mathbf{F}$) in the space $<\mathbf{F},U_{\mathbf{F}}>$ is the collection of all neighborhoods of the set F (the point f).

Appendix 2

production rules

A small sample of the domain-specific knowledge relevant to the car design domain is expressed in terms of the production rules presented in Table 3.

References

Akin O (1979) An exploration of the design process. Design Methods and Theory 13(3–4):115–119

Blessing LTM (1995) A process-based approach to computer-supported engineering design. In: Proceedings of the workshop on knowledge sharing environment for creative design of higher quality and knowledge inventiveness, RACE-Institute, University of Tokyo

Braha D, Maimon O (1998) A mathematical theory of design: foundations, algorithms, and applications. Kluwer, Boston

Braha D, Reich Y (2001) Topological structures for modeling engineering design processes. International conference on engineering design (ICED 01), Glasgow

Brown, D, Chandrasekaran, B (1989) Design problem solving: knowledge structures and control strategies. Morgan Kaufmann, San Francisco

Cech, E (1966) Topological spaces. Wiley, London

Chandrasekaran, B (1990) Design problem solving: a task analysis. AI Mag 11:59–71

Croom, F (1989) Principles of topology. Sounders College, Chicago

Dasgupta S (1994) Testing the hypothesis law of design: the case of the Britannia bridge. Res Eng Des 6:38–57

Gero JS (1990) Design prototypes: a knowledge representation schema for design. AI Mag 11:26–36

Hales C (1987) Analysis of the engineering design process in an industrial context. Dissertation, University of Cambridge, Gants Hill, Hampshire

Hubka V, Eder WE (1988) Theory of technical systems: a total concept theory for engineering design. Springer, Berlin Heidelberg New York

Pahl G, Beitz W (1984) Engineering design. The Design Council, London

Reich Y, Fenves SJ (1992) Inductive learning of synthesis knowledge. Int J Expert Syst 5:275–297

Reich Y, Subrahmanian E, Cunningham D, Dutoit A, Konda S, Patrick R, Westerberg A, the *n*-dim group (1999) Building agility for developing agile design information systems. Res Eng Des 11:67–83

Reich Y (1990) Converging to "ideal" design knowledge by learning. In: Fitzhorn PA (ed) Proceedings of the first international workshop on formal methods in engineering design, Colorado Springs, pp 330–349

Reich Y (1995) Measuring the value of knowledge. Int J Hum Comput Stud 42:3–30

Reich Y (1995) A critical review of General Design Theory. Res Eng Des 7:1–18

Russell S, Norvig P (1995) Artificial intelligence: a modern approach. Prentice Hall, New Jersey

Simon HA (1996) The sciences of the artificial, 3rd edn. MIT Press, Cambridge

Smithers T (2000) Synthesis in designing as exploration. In: Proceedings of the 2000 international symposium on modeling of synthesis, Tokyo, pp 89–110

Suh NP (2001) Axiomatic design: advances and applications. Oxford University Press, New York

Takeda H, Veerkamp P, Tomiyama T, Yoshikawa H, (1990) Modeling design processes. AI Mag 11:37–48

Tomiyama T, Yoshikawa H (1987) Extended General Design Theory. In: Design theory for CAD, Proceedings from IFIP WG 5.2, Amsterdam

Ullman D, Stauffer L, Dietterich T (1986) Preliminary results of experimental study of the mechanical design process. Technical Report 86–30–9, Oregon State University

Yoshikawa H (1981) General design theory and a CAD system. In: Sata E, Warman E (eds) Man–machine communication in CAD/CAM, North–Holland, Amsterdam, pp 35–58