Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1996), *10*, 289–312. Printed in the USA.

Function–behavior–structure paths and their role in analogy-based design

LENA QIAN[1] AND JOHN S. GERO[2]

[1]Canon Information Systems Research Australia, 1 Thomas Holt Drive, North Ryde, NSW 2113 Australia
[2]Key Centre of Design Computing, Department of Architectural and Design Science, University of Sydney, NSW 2006 Australia

(RECEIVED April 3, 1995; ACCEPTED January 31, 1996)

Abstract

In many creative design processes, cross-domain knowledge is required to inspire the new design result. Thus, in knowledge-based design, how we represent the cross-domain knowledge becomes a key issue. In this paper, we present a formalism for design knowledge representation. By analyzing function representation in different design domains, from graphic design and industrial design to architectural and engineering device designs, we find that although the focus of each kind of design is different, the function representation can be generalized into a small number of categories. This formalism can be used in an explorative model of design by analogy, where designs from different design domains are sources to help produce a new design.

Keywords: Creative Design; Design by Analogy; Function–Behavior–Structure

1. INTRODUCTION

It has long been recognized that in design it is possible to articulate a description of existing and putative designs using a function–behavior–structure (FBS) framework. This framework has provided a satisfactory basis for the development of design process models of routine design. Here routine design is taken to mean that class of designing where all the design or structure variables and all the performance or behavior variables are known *priori* and what is to be done is to determine values for the structure variables. When all these variables are not known at the outset we move into an area characterized as nonroutine or creative design. A number of processes have been developed to assist designers to introduce new variables in their designs. One such process is analogy, where variables from another existing but appropriate design are introduced into the current design. How to select appropriate designs and appropriate variables from the selected design are important questions, which do not yet have well-understood answers. This paper presents an approach to the solution of these problems using a formalism built around the function–behavior–structure framework. It utilizes the associations between function, behavior, and structure as goal achievement paths, which have been generalized over classes of designs.

It is suggested that although designs may appear different within a single domain and certainly between domains, desired functions can only be achieved in a limited number of ways, called FBS paths or goal achievement paths. These FBS paths are utilized as the basis of a class of design processes.

A design process typically starts with a set of requirements, from which a designer develops a conceptual idea of the design. In an analogy-based design system, this design, referred to as the target concept design, along with its analogous design, referred to as the source design, establishes a platform for the exploration of a new solution. Both target concept design and source design can be represented in the formalism described in this paper during the design process by analogy.

This design model has been implemented as a design support system using C, Lisp, and XView on Sun workstations. Designs from mechanical design, industrial design, and software design have been tested in the system. The system is not an automated design tool, rather, it is an exploration resource in creative design.

In this paper, we focus on the design knowledge representation through FBS paths, as a basis for cross-domain analogy-based design and suggest that this representation

Reprint requests to: Professor John S. Gero, Key Centre of Design Computing, Department of Architectural and Design Science, University of Sydney, NSW 2006 Australia. E-mail: john@arch.su.edu.au.

plays a crucial role in what follows. Then we give a brief description of a model using this formalism. Finally, we provide two examples using cross-domain knowledge based on this approach.

2. KNOWLEDGE REPRESENTATION FOR DESIGN BY ANALOGY

Some designs from different domains might have similarities at different levels of abstraction. Surface similarities such as attributes of a structure (e.g., color and material) and structural relationships are most recognizable as these attributes can be seen, heard and felt, without requiring deeper reasoning. They have been used in many case-based design systems to retrieve relevant information for new design requirements.

However, in analogy-based design, similarity in higher order relations such as function and behavior is the concern. While functions specify what a design does, behaviors describe how a design achieves its functions. A definition of analogous designs is given as:

DEFINITION: Two designs are analogous if they have a similar function or similar behavior; they may or may not have similar structures. ■

The aim of analogy-based design is to obtain new ideas of possible structures and associated operations from an existing design with a similar function and/or similar behavior. Thus, an organization of the design knowledge that can help function and behavior indexing is required. Once an analogous design is found, what is transferred to produce the new design depends upon the structured design knowledge and the model of reasoning. Two representational issues for analogous designs are raised here.

- What should be included in the representation of a design or a class of designs? and
- How should designs from different domains be organized?

The design goal is achieved by conscious ordering and planning of design elements. A design element is either a physical or logical entity and the relationships between elements are physical connections or logical links. Each element can be defined with many attributes. The elements, their attributes, and their relationships form a design description and are considered as shallow, or superficial, knowledge.

Behind the design description are reasons for the design structure and the functionality of that structure. The behavior of the design structure has an associated function and provides a causal explanation for the design. Function and behavior are considered to be deep, or high-level, knowledge.

Design knowledge covers many aspects, for example, quantity, space, time, physics, goals, plans, needs, and communication. However, one design domain might use certain aspects of this knowledge and ignore others. For a bridge design for instance, one needs to calculate the resistance to load that provides the essential function of the bridge. Therefore, this design domain is primarily concerned with quantity, space, and physics. On the other hand, graphic designs do not require exact measurements of quantity, although space and color are taken into consideration. For example, a tree is to be drawn in a picture, it is not necessary to measure exactly where it should be placed. An air traffic control system is concerned with flight departure and arrival times and the spatial arrangement of the airport. Here, time and space need to be taken into account.

2.1. Causal knowledge

A design description contains structural features including elements, their attributes, and their relationships (or configurations). This design description is essential for manufacturers and is the basis for many CAD systems. However, these design descriptions contain only shallow knowledge about the design: its syntactic information is not sufficient for knowledge-based design systems, especially creative designs (Zhao & Maher, 1992). Finding analogous designs using these design descriptions alone is almost impossible, particularly when different design domains are involved.

When a design is analyzed according to its deep knowledge, its behavior can be derived from a given structure (DeKleer & Brown, 1984; Gero, 1990). State changes of the structure, the structure's attributes, the structure's relationship with other structures, and certain external effects interacting with the structure at a particular time, are used in determining the structure's behavior.

Although differences among different kinds (or domains) of designs vary from high-level abstraction (e.g., function, behavior) to low-level abstraction (e.g., structure and words used in expressing knowledge), the causal links in the designs can be generalized into FBS models. Specifically, from a design perspective, an artefact is designed in such a way as to achieve functionality through a FBS path.

This inclusion of the function and behavior in the design description and relating the three individual entities (i.e., function, behavior, and structure) in one module reflects the causal knowledge in the design (Sembugamoorthy & Chandrasekaran, 1986; Gero, 1990). The path from structure to behavior, and then to function for an existing design shows the causal relationship of the design and is a one way derivation:

$$\mathbf{F} \leftarrow \mathbf{B} \leftarrow \mathbf{S}$$

Relations between function, behavior and structure are referred to as a **FBS path** or the **causal knowledge** throughout this paper.

Although behavior can be derived from a given structure, the FBS path can trace individual entities in both direc-

tions. Well-understood behavior derived from the structure can be specified by linking it with its function, thus eliminating the need for derivation every time the same structure is to be evaluated. This behavior can be saved and used efficiently to relate its associated structure and function. Given a function, one can identify the behavior that realizes the function, and the structure that causes such behavior. Given a structure, its expected behavior and function can be found through the FBS path. When design experiences are stored with related causal knowledge or in a FBS path, a retriever can find structure and the reason behind the design.

Compiling causal knowledge of a design into a knowledge base establishes a basis upon which we can reason about a design by analogy. A function does not necessarily associate with only one behavior. Instead, it can be associated in many ways (i.e., one function may correspond to many behaviors and one behavior may be associated with more than one function). Similarly, behavior can be derived from more than one structure. The central idea of design by analogy is to capture this mapping to find different design descriptions. In other words, having specified a design function or behavior, its design structure might be different. The function and behavior are considered to be analogical retrieval cues or search indexes, which retrieve designs of a similar nature.

2.2. Generalized design knowledge

In existing knowledge-based design systems, generalized and specific knowledge are used to carry out the design process. Generalized design knowledge represents the abstraction of many individual design experiences. It covers a range of situations and features under the abstracted variables. Generalized knowledge is stored in memory, and during the design process it is instantiated into a specific design to suit particular design requirements (Gero, 1990).

Case or specific knowledge includes the specific design experience, episode, and/or the process that transformed the design requirements into the design description and all its relevant details. The design case has a design description with specific values for all instantiated variables. With domain-specific knowledge the case-based design process retrieves a similar case, adapts the old design case, and modifies the case to produce the new design (Maher et al., 1995).

Rosenman et al. (1991) have argued that generalized knowledge and specific knowledge are important because the integration of both can compensate for the disadvantages of each. However, analogy-based design is more concerned with generalized knowledge, because the abstraction of a design concept containing general information about the design's function, its behaviour and its structure in terms of variables is sufficient to carry out the design process.

Generalized design knowledge can be characterized by design variables representing structure, behavior, and function as design ontology. It is also required to account for between-domain knowledge, that is, to generalize knowledge from different design domains. To implement this concept, some design systems classify knowledge taxonomically. Information is compiled into abstract trees covering between-domain knowledge. The taxonomies are used for generating individual examples (Sycara & Navinchandra, 1991).

3. STRUCTURE

As a design representation formalism through FBS paths, we need to describe each individual entity, that is, function, behavior and structure, and their links in more detail. This section is about structure.

The structure specifies what elements the design is composed of, what the attributes of the elements are, and how they are related. The existence of each individual element and the relations between them can directly or indirectly contribute to the functionality of the design. Structure representation varies from design to design, and from domain to domain. To initialize analogy-based design, a formalism representing different kinds of structure is essential.

3.1. Structure representation

A structure variable can be an element, an attribute, a relationship, an operation, or a process. A set of potential structure variables with their valid values constructs a design solution space. The design structure can be described with a graph, G_s, which consists of five finite sets for elements, attributes, relationships, operations, and processes (Fig. 1):

$$G_s = \{E, A, R, O, P\} \ . \tag{1}$$

where

A = a finite set of attribute variables (represented by hexagonal shaped nodes linked to the elements in the graph) $\{a_i\}$;

E = a finite set of element variables (represented by circular shaped nodes in the graph) $\{e_i\}$;

O = a finite set of operations (represented by single rectangles) $\{o_i\}$;

P = a finite set of processes composed of operations (represented by double rectangles) $\{p_i\}$; and

R = a finite set of relation variables between two elements (represented by undirected arcs), $(R \subseteq E \times E, \{r_i\})$.

The structure is divided into two parts in Figure 1: the data structure on the left-hand side, and the process structure on the right-hand side. An element is surrounded by a set of attributes and elements are related if they are joined by an arc. The dotted line represents detachable relations. The operations are linked by control flows (thin arrows) that show the sequence of the process. An operation can have more than one branch according to the condition at the time of execution. The thick arrow shows the data flows, which

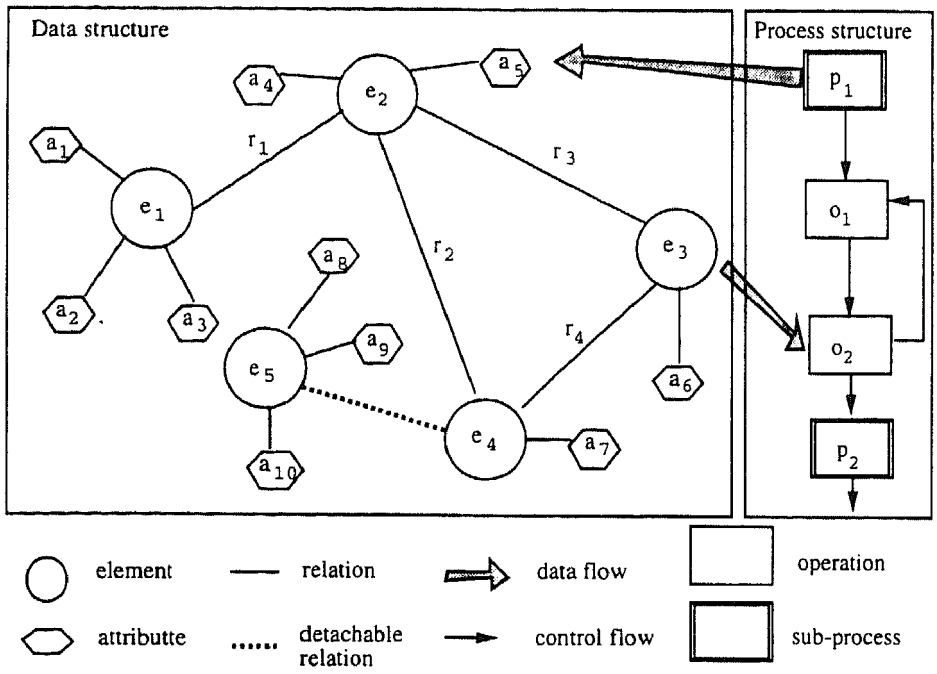


Fig. 1. Structure graph.

result from operations and can come from an attribute or an element.

The description of each term is as follows:

- *Primitive Element and Structure Element:*
  Any design structure is composed of elements. An element in the system that cannot be divided is called a **primitive element**. A primitive element can be either a physical or logical entity. For example, mechanical engineering, civil engineering, and architectural designs involve physical elements, while software engineering designs and graphic designs deal with logical elements. For instance, a chair consists of six physical primitive elements: a seat, four legs, and a back, while in a software design, an array can be treated as a logical element.

  Some elements group together and form a substructure or **structure element**, that has well-defined characteristics. For example, a keyboard is a substructure of a computer. The keyboard has many keys, some LEDs, and a cable. A structure element consists of a combination of primitive elements or other structure elements.

  Both primitive and structure elements are passive. They can be seen, accessed, or manipulated by designers or reasoning processes. These passive elements are referred to as part of the **data structure** and are similar to the concept found in software engineering.

  The most important step in the design process is to choose elements for a design. These elements will affect the function of the design. For instance, a chair is not stable if it has two legs.

- *Attribute:*
  An element has many properties, or attributes, for example, color, shape, material, and so on. The value of an attribute may be continuous (e.g., the width of the door is a continuous numerical value), or discrete (e.g., the material from which the door is made can be wood or glass). The attribute itself or combined with others can play a key role in achieving certain behaviors. For example, the use of color in a picture conveys information that achieves an effect and/or reflects a goal. Not all attributes and their values are necessarily designed to achieve goals. Some are unintentional but can be transformed into an intentional goal if new goals are introduced.

  An attribute is characterized by an **attribute variable** and is encompassed in the data structure.

- *Relationship:*
  In addition to the element attribute, the relations of the elements can also be used to achieve a function. The relationship between the elements is described with **relation variable** and is also included in the data structure.

  If the elements are physical, the relationship between them is a physical interconnection using topological or geometrical data. Examples include the configuration of rooms in a house and the interconnections between parts of a can opener in a mechanical device design. A different topology or geometry of the rooms and the can opener might result in a different use for them.

  If the elements are logical entities, their relationship is abstract and the entities are linked through internal or external actions. For example, in a simple string copy

program, two strings are defined as logical elements. One string is then appended to another by transferring the electronic data. The append action relates the two strings.

A relationship between elements can be fixed or changeable. One kind of changeable relation is that two elements might be tied up together but have loose ends: the relative position of the two elements can be changed, but they are always together. Another kind of changeable relation is detachable, for example, a key can be inserted into a lock or exists by itself. It is only required when one wants to open the lock.

- *Operation and Process:*
  Some designs, like software engineering and chemical control systems, have processes incorporated into the design structure. Operation is an active element that receives information from passive elements and takes appropriate control action to adjust the structure's behavior. A process is formed by a sequence of operations and subprocesses, each of which is referred to as a process unit. As this process is part of the design description, it is referred to as a **process structure**. Examples of such processes are chemical process control systems and programs in software designs.

3.2. Static and dynamic structure

Elements, their attributes, and their relationships are fixed, and no active element or process is included in a static structure. The values of the structure variable are decided at the end of the design process and are used as the design description. They are assumed not to change with time. In architectural and civil engineering design, static structures are dominant. When a building is constructed, the number of floors and the position of the windows and doors are fixed.

Dynamic structure means that elements, attributes, and their relationship to one another can be changed. For example, glass color (an attribute) in phototropic spectacles can darken in the sunlight; a door's structure is dynamic as its position in relation to the door frame changes when it opens or closes. Some substructures are detached from the main body of the design and are thus dynamic structures. For example, a lid can be either on or off on a cooking-pot. Similarly, elements that enter or leave a design are defined as dynamic. For instance, data in a program can be created or deleted at run time. A process structure is itself dynamic as it has a sequence of operations, each executed at a specific time.

The range of values of a dynamic structure variable needs to be determined at the time of designing, but these values can be changed later at "run-time" or once the design is produced. For instance, the angle range of a door in relation to the door frame is between 0 to 90 degrees, which is determined at design time, but the actual position is decided when it is used. Here, the angle is a structure relation variable.

4. BEHAVIOR

Behavior is one of the ways by which the meaning of the structure is inferred by a designer or user. It is characterized by behavior types, behavior variables, and qualitative causal relations in this work.

4.1. Behavior types and behavior variables

Behavior can be generalized into two classes of interest here.

- Spatial: Spatial behavior describes the behavior of ordered objects in 2D or 3D space.
- Temporal: Temporal behavior obeys time constraints. Events can happen sequentially, overlap one another, be simultaneous, or be of specific duration. The effects of these events are states that occur in different times or intervals.

Behavior can also be characterized by two kinds of behavior variables.

- Structural (Direct): A behavior is derived from the structure itself without any external effect. For example, the floor of a room has an area as a behavior variable, which is directly derived from the room's width and length.
- Exogenous (Indirect): A specific kind of behavior is shown when an external object is applied to a structure. For example, the water tap or faucet has an area that water can pass through, but water is not part of the water tap design, it is only related to the design. The water flow is an indirect behavior variable controlled by the diameter of the water tap.

4.2. External effects

An external effect can be defined as an object imposed onto the design (e.g., water flowing in a faucet), an environment around the design (e.g., location and riverbed conditions for a bridge design), or an operation applied to the design (e.g., pushing a door open). The external effect can be a human action or any environmental effect applied to the design such as temperature or wind. Many engineering designs, such as mechanical design and electronic equipment design, require external operations to change the structure status to reach a goal.

Although the external effect does not belong to the design, it affects the design behavior. Without a correct interaction between the external effect and the design, the design could not mean anything at all. For instance, a bridge is built for a particular river. The same design of the bridge might not be useful for a different river.

The external effects can be characterized by exogenous variables in the same way as the structure variables. An exogenous variable describes a passive situation (e.g., a riverbed), or an operation (e.g., pushing a door open). An exog-

enous variable applied to a structure variable may cause a change in the structure if the structure variable is dynamic. For example, the door's position is a structure variable that is changed by imposing force upon it. In this case, a dynamic structure causes a dynamic behavior. However, a static structure can also generate a dynamic behavior. For instance, a water pipe has a static structure designed for water to pass through. Water is considered an exogenous variable. The behavior of water passing through the pipe can be characterized by a behavior variable, water flow rate.

From a design process point of view, behavior can be used for problem formulation, synthesis, analysis, evaluation, and reformulation (Gero, 1990). Descriptive terms for the behavior can explicitly address the functional requirements of a design. Behavior plays an important role in linking the structure and the functions (Tham et al., 1990). A design artefact is teleologically driven as any design is produced to meet a goal or achieve a purpose (here called function). The purpose provides a means of understanding the behavior of the design.

4.3. Qualitative causal relation

During the conceptual design phase, quantitative or detailed data about the structures might not be important, but qualitative analysis can help the designer to develop concepts without concentrating on numbers. The incremental qualitative analysis of DeKleer and Brown represents quantities according to how they change when a system input is perturbed, for example, increasing, decreasing, constant, or indeterminate (DeKleer & Brown, 1984).

Qualitative physics is involved with representing and reasoning about the physical world in qualitative terms. It proposes formalizing the commonsense knowledge about the everyday physical world in a way that is simpler than classical physics. Qualitative physics captures the tacit knowledge underlying the mathematics of continuously varying quantities and differential equations, and retains important distinctions (DeKleer & Brown, 1984; Forbus & Gentner, 1990). Research in this area has been growing rapidly over the past 20 years, though often under different names such as qualitative process theory, naive physics, or commonsense knowledge.

Qualitative physics involves two key issues.

- *Qualitative expression:*
  Continuous properties of the world are expressed through a discrete system of symbols. Qualitative descriptions are approximations by quantization and fuzzy variables, but not all approximations are useful. Each qualitative state of the system should lead to qualitatively distinct behavior (Forbus & Gentner, 1990).
- *Production of causal accounts of physical mechanisms:*
  Causal accounts of physical mechanisms are much easier to understand than the behavior of their physical environment (Bobrow, 1984). Causality plays an important role in qualitative physics. A systematic set of conventions is used to represent how information is propagated through physical artefacts to achieve the overall purpose or function. This set of conventions is called causality and can be used to analyze, design, and diagnose engineered artefacts. Causality is a link between function and structure through behavior.

The causal relations among the states or behavior variables can be represented as causal ordering or as an influence graph (Sycara & Navinchandra, 1991). While the causal ordering labels the causal link with an equation name, an influence has a qualitative differential relation between two variables where a change in one variable is dependent on the other. The influences of the problem variables in a physical situation can be used to represent the dynamic behavior of the situation. This behavioral description expresses causal interactions between the structure and behavior variables, and it is a thematic abstraction that has been used in CADET to retrieve cross-contextual designs (Sycara & Navinchandra, 1991).

To efficiently identify a structure and its potential behavior, the qualitative causal reasoning process can be codified into a design representation as a qualitative causal relation. Instead of deriving behavior from a structure, this behavior can be explicitly specified.

4.4. Behavior representation

Behavior type, behavior variable, and qualitative causal relation are used to represent a generic behavior of a structure. This representation covers the following aspects: behavior type; static and dynamic behavior of a structure; external and internal effects on changes; direction of causation or dependencies of the structure and behavior variables; qualitative measurement of the changes; and qualitatively distinct states.

The following formalism shows the qualitative causal relation between a structure variable and a behavior structure:

$$B(T)\{q_b\} \leftarrow S\{q_s, Ex|O\} \ . \tag{2}$$

where

B = a behavior variable,

Ex = an exogenous variable,

O = internal operation,

q_b = qualitative state of B,

q_s = qualitative state of S,

S = a structure variable,

T = behavior type,

$\leftarrow$ = causal direction,

{ } = optional symbol.

This expression not only reveals the dependency relation between the structure and the behavior, it also describes the qualitative change from the structure to the behavior, and relates the structure to the external effect. This behavior type is encoded into this expression to give a more comprehensive explanation about the design.

If the behavior is static, the corresponding behavior variable, B, is dependent on a structure variable S with or without the exogenous variable Ex. Therefore, the qualitative causal relation for static behavior becomes:

$$B(T) \leftarrow S\{Ex\} \ . \tag{3}$$

A static behavior does not change with time. It is derived from a static structure. For example, a cup is static in the sense that neither its handle nor shape can be changed, once it has been designed. Therefore, the volume of the cup (as a behavior variable) is fixed.

The external effect is optional, that is, a behavior might or might not be affected by an external effect.

If the behavior is dynamic, the respective qualitative states (q_s, q_b) in eq. (2) are required to describe how the changes can occur. The qualitative state describes a stable status and change tendency in terms of a qualitative measurement such as increasing (+) and decreasing (−). For instance, the hinge angle of a door can be increased or decreased. A status can have discrete values or continuous values quantized into intervals. A discrete value with an interval for continuous values or a changing tendency makes distinctive qualitative states.

For example, in a hand dryer design, there are two main structure variables that keep the air temperature constant: *coil* and *thermostat switch*, an exogenous variable: *air*, and two behavior variables: *temperature* and *thermostat status*, Table 1. If the temperature of the hot air exceeds 80°C, the thermostat switch is turned off. When the temperature of the hot air decreases below 80°C, the thermostat switch is turned on. The temperature with continuous values is then quantized into 0 and 1, where 0 corresponds to temperature below 80°C and 1 represents the status of temperature equal to or above 80°C. The thermostat status has two discrete values: 0 for *Off* and 1 for *On*.

The qualitative causal relation also describes the link between two structure variables, and between two behavior variables. If there are n related behavior variables and m structure variables where the mth structure variable has a direct effect on the first behavior variable, the causal propagation is from structure to behavior. This is referred to as **series propagation** and is expressed as:

$$\mathbf{B}_n \leftarrow \mathbf{B}_{n-1} \ldots \mathbf{B}_1 \leftarrow \mathbf{S}_m \leftarrow \mathbf{S}_{m-1} \leftarrow \ldots \leftarrow \mathbf{S}_1 \tag{4}$$

Note that **B** and **S** are shorthand for $B(T)\{q_b\}$ and $S\{q_s, Ex|O\}$, respectively.

Multiple influences from structure to structure, from structure to behavior, or from behavior to behavior represent another type of causal propagation called **parallel propagation**. If m structure variables affect one type of behavior or if m behavior variables cause another behavior change, this can be expressed, respectively, by eq. (5):

$$\mathbf{B} \leftarrow (\mathbf{S}_m, \mathbf{S}_{m-1}, \ldots, \mathbf{S}_1) \tag{5}$$

or by eq. (6):

$$\mathbf{B} \leftarrow (\mathbf{B}_m, \mathbf{B}_{m-1}, \ldots, \mathbf{B}_1) \tag{6}$$

By combining series and parallel causal propagation, structures and behaviors can be represented as behavior graphs, which are similar to those of Sycara and Navinchandra (1991). In Figure 2, the arrows linking structure to behavior variables, and behavior to behavior variables show the causation direction or their dependencies, and allow linking two variables with attached qualitative symbols (e.g., an arrow from B_5 to B_2 with a numeric symbol, 1, and a symbol, +, respectively). If a relation between two variables is static, no qualitative symbols are shown. An exogenous variable can be directly applied to a structure variable or a behavior variable.

The causal dependencies can be cyclic, that is, one variable A affects another variable B, meanwhile B also affects A. An example of a behavior graph, the hand dryer design, is shown in Figure 3. This models the standard control loop. Initially, the thermostat switch that controls the heating coil is on (e.g., 1). This causes the temperature of the blowing air to increase (e.g., +). Until the temperature reaches 80°C (e.g., 1), the switch is turned off (e.g., 0). Because the switch is off the hot air becomes cooler and its temperature drops

Table 1. *Qualitative measurement of hand dryer design*

Behavior variable	Qualitative values
Heat coil temperature	+, −, 1(≥80°C), 0(<80°C)
Thermostat status	1(On), 0(Off)

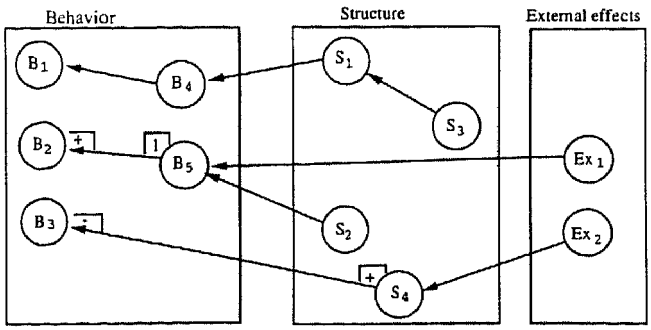


Fig. 2. Behavior graph.

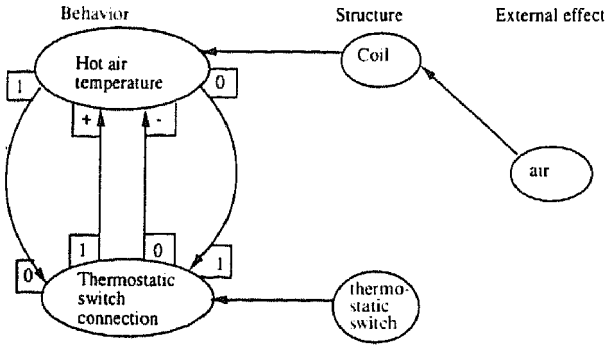


Fig. 3. Behavior graph for hand dryer.

(e.g., −). When the temperature decreases to below 80°C, the switch is turned on again.

In the behavior graph, the arrow indicates the causal direction, while the small rectangle attached to the end of the arrow represents the qualitative state. The change from one behavior state to another might be described by more than one qualitative causal relation. For instance, *temperature* $\overset{+1}{\leftarrow}$ *connection* and *temperature* $\overset{-0}{\leftarrow}$ *connection* describe two distinctive qualitative state changes. These changes are then combined into *temperature* $\overset{+1,-0}{\leftarrow}$ *connection*.

Unlike the influence graph where a monotonic expression is used by marking the change tendency on the causation arrow (e.g., $a \overset{+}{\leftarrow} b$ means a increases while b increases), the above behavior graph needs to describe the qualitative measurement at both ends of the causation arrow. This representation extends the capabilities of expressing the changes caused not only by other changes, but also by a certain stable state, for example, temperature above 80°C. Behavior variables and behavior graphs can be used to retrieve analogous designs.

5. FUNCTION

Functions are usually shown as labels representing the purposes of an artefact. Existing design systems tend to limit functional descriptions to a few words due to the difficulties of interpreting natural language. The function label then can be used to identify behaviors that achieve the corresponding function.

Because a function label is used as a key word, or an index, to retrieve corresponding behavior, it is necessary that the functional requirements are represented in a consistent way. Spelling mistakes or incorrect word order can impede the recognition of associated behaviors. Thus, a sophisticated parser with the capability of partial matching, and reordering of words, is required to process function labels.

Another problem in using function labels is that any specific meaning of a label may be limited in a search for an analogous function. One word can be used in one domain and another used in a different domain, but the two words might be synonymous and generalizable into a more abstract word. In analogy-based design, abstract function is a key representation for discovering analogous designs. For example, an umbrella's function of blocking raindrops and a door's function of blocking access (e.g., people, animals, etc.) can be expressed as the generalized function of blocking an object.

5.1. Taxonomies and function variable

There is no uniform definition or representation of function. Function is associated with a design's teleology. A widely accepted definition of function in design research is that a function is a relationship between the input and output of energy, material and information, or the manipulation of the fluxes of energy, material and information (Wirth & O'Rorke, 1993). Manipulation of fluxes involves associated actions. Function has also been defined as *to do something*, which is represented by *to verb objectives*, where the *verb* describes actions like storing, transmitting, transforming, changing, etc., and the *objectives* are fluxes.

Function can be characterized by two kinds of variables—action and flux. Their values correspond to the nodes appearing in two taxonomy hierarchies: action taxonomy and flux taxonomy.

The action taxonomy represents a hierarchy of verbs with associated parameters. For example, an action can store or transmit fluxes, transform one flux into another, change the amount of flux, and so on. A function of a curtain is to *control light intensity*. Here, *control* is a special kind of *change* action, whereas *light intensity* is the amount of the *light* flux.

The flux taxonomy has a flux root node that is subcategorized as material, energy, and information. Under the material node, are fluid, solid objects, etc. The fluid may be any of water, oil, and so on.

5.2. Function decomposition

A list of functions can be specified for a design. Any function that is directly associated with a set of behaviors, is called a **primitive function**. An abstracted function needs to be decomposed into several more specific primitive functions. For example, the function of a house is the provision of a living area, this in turn can be decomposed into sleeping area, eating area, etc. This information is available only through domain knowledge. Eating area, for example, requires light potentially provided by windows in the room, and heat insulation is potentially provided by the thickness of the walls and the materials out of which they are made. Thus, provision of light and heat insulation are two primitive functions.

Some functions of the design product are primary, others are secondary. **Primary** functions determine the name and the concept of the design product while **secondary** functions are supportive, realized or user interpreted.

5.3. Function representation

In Wirth & O'Rorke (1993), function is characterized by five attributes: input flux, output flux, source component, destination component, and function carrier. In the case of the hot–cold water faucet example, hot and cold water are two kinds of input fluxes, and the mixture of both is the output flux. The person who controls the water flow is the source component. If the output water mixture is to be used for washing hands, the hands are considered to be the destination component. The hot–cold water faucet is the function carrier. The source and destination components defined here should not be confused with the component defined in the structure of the design prototype.

Taking a different approach, the FBS model (Sembugamoorthy & Chandrasekaran, 1986) treats function, behavior, and structure as individual concepts, and relates them to causal relationship, or a dependency network (Gero, 1990; Zhao & Maher, 1992).

The function representation given here is defined according to function variables, and the relations between function and their corresponding behaviors.

Although design products differ from one domain to another, the way in which the structure or substructure achieves the design goal or function through behavior can be classified across design domains. A function can be accomplished in the following generalized FBS path types or FBS types:

- FBS type I: achieved by a static behavior ($F \leftarrow \mathbf{B}^s$).
- FBS type II: achieved by a state of a dynamic behavior ($F \leftarrow \mathbf{B}^d$).
- FBS type III: achieved by a set of behaviors occurring at same time ($F \leftarrow \{\mathbf{B}\} \cup \{\mathbf{B}\}$).
- FBS type IV: achieved by a set of behaviors occurring in sequence ($F \leftarrow (\{\mathbf{B}\},\{\mathbf{B}\})^+$).

$\mathbf{B}^s$ and $\mathbf{B}^d$ represent static and dynamic behavior states, respectively. Here, $\mathbf{B}$ without any superscript denotes either $\mathbf{B}^s$ or $\mathbf{B}^d$, that is, $\mathbf{B}^s|\mathbf{B}^d$. $\mathbf{B}^d$ can be a value or a range of values of the behavior variable. $(\{\mathbf{B}\} \cup \{\mathbf{B}\})$ represents the coexistence of two sets of behavior states referred to as parallel states, while $(\{\mathbf{B}\},\{\mathbf{B}\})$ describes two sets of behavior states occurring in sequence, referred to as series states. The plus ($^+$) sign represents state repetition. The $(\{\mathbf{B}\},\{\mathbf{B}\})^+$ describes two sets of behavior, one of which repeats after the other at least once.

FBS types I and II are considered to be primitive function achievement because each function associates with one behavior variable (Fig. 4). A direct behavior variable can be derived from other indirect behavior variables either in parallel or in series.

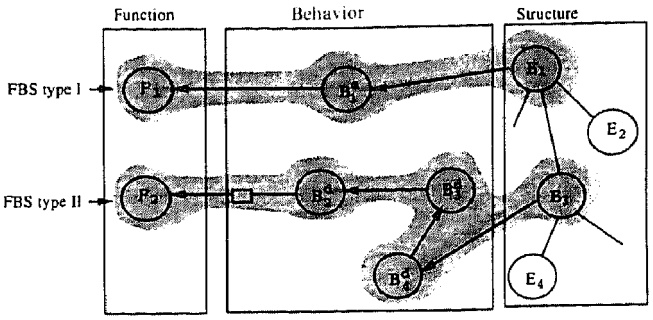


Fig. 4. FBS types I and II marked with hatching.

In FBS type I one function is realized by static behavior. The FBS type II design has a function associated with a dynamic behavior state. The behavior state (with a qualitative measurement) is represented by a small rectangle in Figure 4.

However, in many designs, a function depends on more than one behavior variable. The set of variables can be composed in parallel (FBS type III), in sequence (FBS type IV), or both (Fig. 5). A function achieved by repetition of several behavior variables is seen as a special case of sequential behavior. The number marked on the arrows for FBS type IV function indicates the order of the states.

In general, the function representation can be described as below:

$$F \leftarrow \mathbf{B}^s|\mathbf{B}^d|(\{\mathbf{B}\} \cup \{\mathbf{B}\})|(\{\mathbf{B}\},\{\mathbf{B}\})^+ \tag{7}$$

Knowing the FBS type establishes a basis for analogy-based design in between-domain designs. If two designs have the same FBS type, they are considered for analogical mapping because they have the same way of behaving to achieve a function. To some extent, the FBS type sets constraints

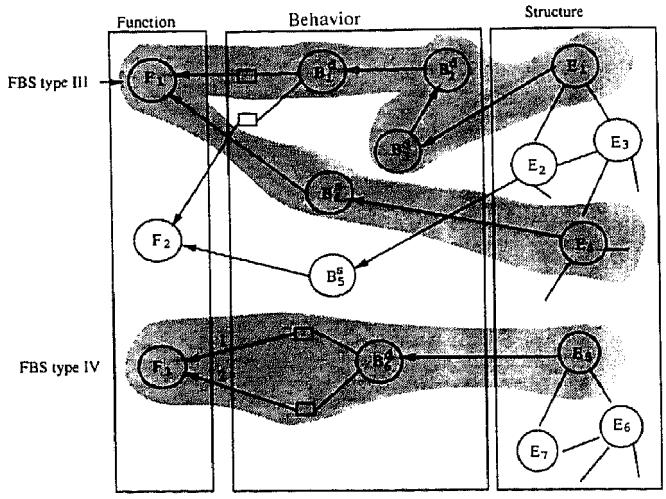


Fig. 5. FBS types III and IV marked with hatching.

upon retrieving analogous designs and elaborating analogical mapping.

5.3.1. FBS type I

The FBS type I design involves functions achieved purely by the existence of a structure: no operation is required. This is a primitive description of the relation between a function variable and a behavior variable. The behavior variable might be generated from one or more structure variables. Both structure and behavior variables are static.

Static structure variables such as elements, attributes, and relationships (not processes) can behave statically. The structure may also relate to external effects. If external effects are involved, they tend to be environmental, that is, they relate to the design without the intention to change the structure. For example, in architectural design, a view can exist due to the strategically placed windows. The static relationship between the windows (i.e., the design structure) and the surroundings (i.e., the environment effects) enables the *view exists* functionality.

5.3.2. FBS type II

The FBS type II design deals with dynamic structure and behavior. A dynamic structure variable has a valid range of values that can be changed by external effects or by a change in other structures. A specific value or a range of values determines a qualitatively distinct state of the structure and the behavior, and in turn provides a function. The function is achieved by a state or a qualitative value of a behavior variable.

For example, opening a door allows access while closing the door impedes access and provides privacy. When the behavior variable *access area* is zero, the door is closed, otherwise it is open. The physical position of the door is a dynamic structure variable. Depending on the operation applied to the door, different behaviors occur and different functions can be achieved.

5.3.3. FBS type III

The FBS type III design composes primitive FBS paths conjunctively. In many complex designs, a function is affected by more than one behavior variable. A set of static behavior and dynamic behavior variables or a combination of these are present simultaneously.

For instance, the function *carry-load* in a rigid frame design is achieved by two static behavior variables—*mechanism bending* and *mechanism compression*, each of which is deduced from other behaviour variables (Zhao & Maher, 1992).

5.3.4. FBS type IV

The FBS type IV design has functions attained by a sequence of behavior states. The function not only depends upon several behavior states, but also upon their sequence, and some states may be required to repeat one after another. Repeat behavior variables are special sequences that allow the same behavior states to reoccur several times after they have changed.

In the door example, although the *opened* and *closed* states provide the *access* and *private* functions, there is no incorporated information as to how the *opened* state is reached from the *closed* state. There should be two functions, the first of which allows the door to open (if it is closed) and the second allows the door to close (if it is open). The opening function involves state change from closed to opened, while the closing function does the reverse.

Figure 6 shows an example that has a behavior sequence expressed as follows:

$$((B_1 \cup B_2), B_3, ((B_4, B_5)^+ \cup B_6), B_7, B_8) \ .$$

First, behavior states B_1 and B_2 should be presented simultaneously, followed by B_3. B_4 and B_5 have a repeat sequence that exists simultaneously with B_6. When B_6 is no longer true, the repetition of B_4 and B_5 is terminated before B_7 starts. B_8 is the final state in the sequence. The sequence of the behavior contributes to the function, F.

6. DESIGN PROTOTYPES

Design prototypes are chunks of knowledge representing a class of generalized design experiences (Gero, 1990). The content of a design prototype includes pertinent descriptions of the function, behavior, structure and external effect, and various forms of knowledge that support the commencement and continuation of a design process.

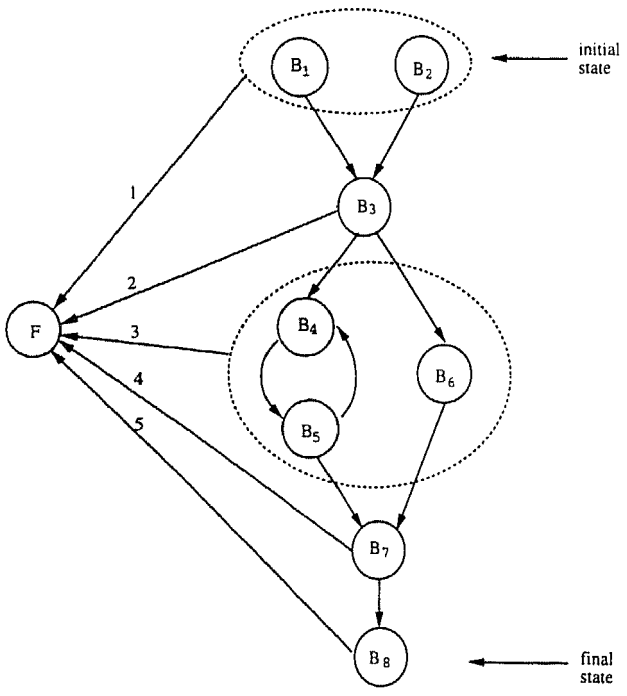


Fig. 6. Behavior sequence.

Design prototypes have been used successfully in PROBER, a routine design system in the building envelope domain (Tham et al., 1990). Through the dependency network that relates the function, behavior, and structure variables, PROBER identifies the structure element for design instantiation.

The notion of design prototypes has also been developed for creative design (Zhao & Maher, 1992) in a structural design domain. The dependency network is extended with generic causal relations such as data dependency relations, existence dependency relations, and component-system relations, so that mutation and analogy techniques can be applied.

However, both design systems deal with static structures like building and structural designs. To overcome the shortage of knowledge representation for designs with dynamic structures and to provide a uniform representation for between-domain analogies, this research enriches the notion of the design prototype by adding a new kind of knowledge called qualitative causal knowledge.

6.1. Design prototype representation

Design prototypes include shallow, operational knowledge as well as deep causal knowledge. While the shallow-level knowledge can be seen as knowledge that is sufficient for performing a task itself, the deep knowledge captures an underlying causal relationship between the function, behavior, and structure.

The design prototype is defined by a set of design variables and various kinds of knowledge described in the following formalism:

$$P = \{F, B, S, Ex, K_r, K_q, K_{cnt}, K_{cmp}, K_{qc}\} \ , \qquad (8)$$

where

B = a finite set of behavior variables,
Ex = a finite set of exogenous variables,
F = a finite set of function variables,
K_{cmp} = computational knowledge,
K_{cnt} = context knowledge,
K_q = qualitative knowledge,
K_{qc} = qualitative causal knowledge,
K_r = relational knowledge,
S = a finite set of structure variables.

While function describes goals or intentions of the design artifact, structure specifies individual elements, their attributes, and relationships between them. The exogenous variables describe the environment affecting the structure of an artifact. The link between the structure, exogenous, and function is established by behavior and represented by qualitative causal knowledge and relational knowledge.

The relational knowledge, K_r, represents the dependencies between function, behavior, and structure, while qualitative knowledge, K_q, describes the directional effect on other variables of changing a variable. The design prototype also contains computational knowledge, K_{cmp}, which is the quantitative counterpart of qualitative knowledge, and context knowledge, K_{cnt}, which identifies effects from external agents on the design in relation to exogenous variables. Distinguished from K_r, which focuses on the control of variables during the design process, K_{qc} concerns the dynamic characteristics of the design artefact once the design is made.

6.2. Qualitative causal knowledge

Qualitative causal knowledge summarizes causal relations between the function and the behavior qualitatively and between the behavior and the structures. This is summarized by the following expressions (from 7, 2, 4, 5, 6):

$$\begin{pmatrix} F \leftarrow \mathbf{B}^s|\mathbf{B}^d|(\{\mathbf{B}\} \cup \{\mathbf{B}\})|(\{\mathbf{B}\}, \{\mathbf{B}\})^+ \\ B(T)\{q_b\} \leftarrow \mathrm{S}\{q_s, Ex|O\} \\ \mathbf{B}_n \leftarrow \mathbf{B}_{n-1} \ldots \mathbf{B}_1 \leftarrow \mathbf{S}_m \leftarrow \mathbf{S}_{m-1} \leftarrow \ldots \leftarrow \mathbf{S}_1 \\ \mathbf{B} \leftarrow (\mathbf{S}_m, \mathbf{S}_{m-1}, \ldots, \mathbf{S}_1) \\ \mathbf{B} \leftarrow (\mathbf{B}_m, \mathbf{B}_{m-1}, \ldots, \mathbf{B}_1) \end{pmatrix} .$$

Qualitative causal knowledge provides a basis for analogy-based design as it includes the deep knowledge about what and how a behavior achieves a function, and what and how a structure derives the relevant behavior. This is especially useful during the analogy retrieval and analogy elaboration processes.

7. AN ANALOGY-BASED DESIGN MODEL

Usually, an analogical problem-solving process is applied when a routine design fails to yield a design solution. An analogous design (or source) with similar requirements could be retrieved, then adapted or transformed for the current design with the assistance of domain knowledge in the form of heuristic rules and transformation operators. If a design plan is stored with the source design solution, it provides more information about adaptation and transformation (Bhansali & Harandi, 1992).

Design is also an exploration process that differs from search. While a search process locates values of design variables in a defined state space, an exploration process produces state spaces (Gero, 1994). A design state space is defined by a set of variables and processes operating within a bounded context. By producing state spaces, is meant that either new design state spaces are created or existing design state spaces are modified.

While most design systems use analogy to search for solutions to design problems, this research concentrates on the

exploration aspect. An existing design is studied, that is, design variables such as function, behavior, and structure variables are obtained from the existing design. From this known design state space an analogous design can be retrieved. By mapping the current existing design and the analogous or source design, useful information is transferred by analogy to the current design to modify the existing state space. This design process bases itself on an existing design solution for a new design problem and explores alternative solutions to generate a creative design.

The intention of the remainder of this paper is to explore the potential of having a computational model that can work with between-domain analogies using the design prototype knowledge representation driven by function and behavior. This approach proposes a model combining analogical problem-solving and analogical exploration in the design process (Fig. 7). The upper section shows the exploration process while the lower section shows the problem solving process. Both processes have retrieval and elaboration tasks, but their tasks differ.

The design retriever is divided into two tasks: a concept retriever and an analogy retriever. While the concept retriever retrieves a target concept design with the given design name, the analogy retriever finds designs different from the new design, but with a similar function or behavior. The primary functions, associated causal explanation or behavior, and deduced features can then be used to retrieve analogous designs. In the analogy elaboration process, mapping between the target and the source can be found. The correspondences are considered to be learned features, which can be transferred and transformed from the source to the target as a guide in the problem-solving process. The result of the analogy elaboration is a design structure conjecture. This is the proposed new design and needs to be evaluated.

The design model proposed here aims to explore new design solutions from analogous designs. This model differs from the purely problem-solving approach in two respects: (1) useful features are derived from a target design concept and used to retrieve between-domain analogies; (2) adaptation and transformation of the source analogy solution is guided by mapping between the source and the target.

This model compiles design prototypes into the knowledge base. Each design prototype contains the qualitative causal knowledge that provides a platform to explain how an existing design achieves its functionality. It explores the solution space by using a target concept design, which increases the possibility of retrieving between-domain analogies and guides the adaptation process. In contrast to failure triggered creative design models, this model uses a target concept design as a starting point.

7.1. Design retrieval process

There are two distinct tasks involved in the **design retriever**. The first is to find a target design concept and the second is to retrieve an analogy source design.

Design requirements can be decomposed into function, behavior, and/or structure, and usually they provide a design concept name (i.e., what is to be designed). The **concept retriever** uses the concept name to find an existing design and deduce its primary function and corresponding behavior. The deduced function and behavior then act as analogical retrieval cues.

The **analogy retriever** receives input from design requirements and/or from the target concept design. These can be generalized as label indexes or graph indexes that retrieve analogous designs with the same function or behavior.

A target concept design can be found according to the design requirements. This target concept design is used to explain how an existing design works. The primitive functions and associated behaviors of the existing design are then used as analogy retrieval cues to search for analogous designs in different domains. Although an analogy can be retrieved from the same design domain, this paper focuses on between-domain analogies and the issues related to them.

7.2. Analogy elaboration process

As part of an exploration process, mapping is used to identify correspondences between the source and target,

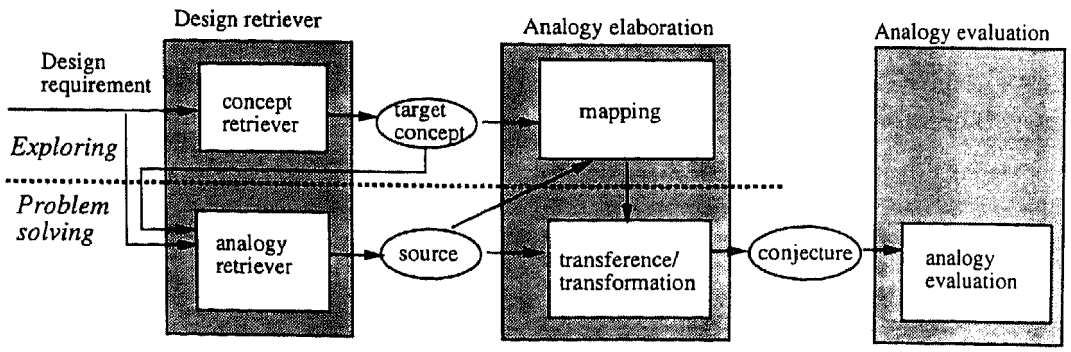


Fig. 7. Analogical problem solving and exploring processes.

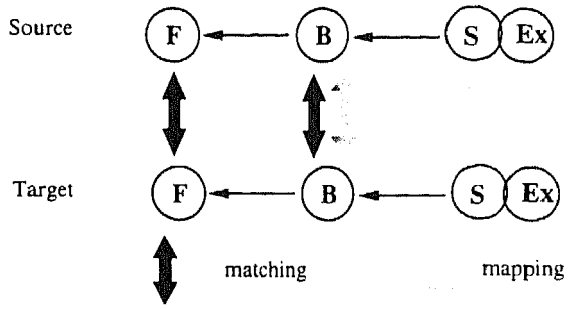


Fig. 8. Matching via function and behavior, mapping via structure.

while transferring analogy inferences indicates what can be used from the source design (Kedar-Cabelli, 1988). The aim of analogy-based design is to find structures and external effect mappings (in terms of design variables) through function or behavior matching (Fig. 8). Matching means that two function variables, two behavior variables, or two behavior graphs are identical at an abstract level. Mapping means that two structure variables or two exogenous variables have the same associated functions or behaviors.

If two functions match, their associated mapping for behavior and structure, plus external effect, can be found. If two behaviors (two behavior variables or behavior graphs) match, only structure plus external effect mapping can be obtained. In the case of matching behaviors, different structures might be found to attain different functions.

As part of the problem-solving process, the source design is transferred partially, and may be transformed, to the target resulting in a new design. This adaptation process is guided by the structure mapping process. In other words, structure mapping identifies potential design variables that can be transferred from the source to the target.

7.2.1. Mapping process

From a design point of view, the mapping process aims to find structure correspondences between the source and target designs. As mentioned in Section 3, a structure has four kinds of design variables (refer to Fig. 1): (1) element; (2) attribute; (3) relation; and (4) process.

Elements, attributes, and relations can be seen as data structure while the process is referred to as process structure. Data structure is a formalism that represents the ordered and accessible relationships in the data (i.e., elements and their attributes). The process represents a unique and finite set of events that is defined by its purpose or effect, which is achieved under certain conditions. The process performs operations on the data.

Data structures and process structures have significant differences. A data structure is passive, but accessible. Its elements are ordered spatially, causally, or logically. A process structure, on the other hand, is active and can act on the data, that is, it can change the data (if the data has a dynamic structure). The process structure involves temporal knowledge: the sequence of operations affects the sequence of the behavior states. This difference between data and process structures suggests that mapping occurs from data-to-data structure or from process-to-process structure. The process structure needs to operate on the data structure and the process mapping is performed before the data mapping.

A mapping unit (MU) is the structure (including data and process) identified through the FBS path or qualitative causal knowledge, and contains one or more structure variables. A data structure MU appears in designs of FBS type I, II, or III, while a process structure MU is found in designs of FBS type IV. The mapping rules are domain-independent and defined for these four types of FBS paths.

At the end of any mapping process, a list of correspondences for elements, attributes, relations, and operations are recorded. Some will be selected for transference from the source to the target.

This structure mapping process, which guides the analogy elaboration, occurs at different levels, for example, the level of the individual variable (element, attribute, relation, or process) and the level of the groups (behavior or structure). Due to the nature of between-domain analogy, mapping rules are defined without recourse to domain-dependent knowledge. A summary of the mapping and transferring rules is shown in Table 2.

Three rules have been defined for one-to-one mapping. **Rule 1** applies when an attribute variable maps another attribute variable. **Rule 2** applies when an attribute variable and a relation variable are mapped. **Rule 3** is for mapping two relation variables. These three rules are primitive rules, which can be applied to the static and dynamic structure mapping processes invoked by **Rule 4** and **Rule 5**.

The static structure mapping process in **Rule 4** takes every pair of the structure variables from the source and target and compares them in a generalized form. General knowledge is more successful if the pair has the same kind of structure, for example, both are attribute structures or both are relation structures. However, a pair in which one is an attribute and the other is a relation is more difficult to match using only structure information, because attributes and relations are different types of design features.

The dynamic mapping process in **Rule 5** is based on structure mapping theory (Falkenhainer et al., 1989/90), in which structural properties are mapped from a source domain to a target domain. This mapping theory suggests that a system of relations known to hold in the source also holds in the target, through causal links. The preservation of the behavior, which is generated from relations, enables the detailed mapping between two designs to be traced.

Furthermore, the composite mapping rule **Rule 6** applies **Rule 4** and **Rule 5** for static and dynamic structures, respectively. **Rule 5** consists of a complicated process that finds

Table 2. *Summary of mapping and transferring rules*

Rule	Condition	Mapping	Transferring
Rule 1	$a_i \Leftrightarrow a_j$	$e_i \Leftrightarrow e_j$	a_s
Rule 2	$a_i \Leftrightarrow r_j$	$e_i \Leftrightarrow \{e_{j_1}, \ldots, e_{j_n}\}$	a_s and e_s or r_s and $\{e_{s_1}, \ldots, e_{s_n}\}$
Rule 3	$r_i \Leftrightarrow r_j$	$\{e_{i_1}, \ldots, e_{i_m}\} \Leftrightarrow \{e_{j_1}, \ldots, e_{j_n}\}$	r_s and some of $\{e_{s_1}, \ldots, e_{s_n}\}$
Rule 4	static structures	apply static structure mapping process	combined
Rule 5	dynamic structures	apply dynamic structure mapping process	combined
Rule 6	combined static and dynamic	apply composite behavior mapping process	combined
Rule 7	process structure for primary function	apply process structure mapping process	*Ex*, *O*, or *S*
Rule 8	process structure for secondary function	apply process structure mapping process on initial and final states	*Ex*, *O*, or *S*

the mapping at the element level. Due to the dynamic behavior of the design structure, behavior semantics offer useful hints regarding the mapping. The mapping process uses qualitative causal knowledge to generalize behavior–structure graphs constructed from FBS paths and related static structures. The mapping is executed using a subgraph isomorphism algorithm. **Rule 7** and **Rule 8** are process structure mapping rules for primary function and secondary function, respectively. The algorithms are described in more detail in Qian (1995).

Mappings at different levels provide an easy way to transfer the source structure to the target. The transference of knowledge is based on the corresponding mapping rules. The transference process merely suggests what can be passed from the source to the target for certain structure variable combinations.

Mapping rules 1 to 6 result in mappings at four levels (Fig. 9). Starting with function matching, two behavior groups are mapped using the FBS path (mapping level 1, L1), then individual behavior variables are mapped (L2), followed by the mapping of element group (L3), and finally elements are mapped (L4).

The mapping at the top is more general than the one at the bottom. Mappings L1 and L3 come directly from FBS paths while mappings L2 and L4 are obtained by mapping rules.

In process structure mapping, internal processes and external operations imposed on the structure play key roles in composing the behavior sequence. Therefore, not only design data structures, but also corresponding operations, or process structures, are required for mapping to produce a new design.

If mapping of a primary function is required, the behavior sequence must be mapped exactly. This means that not only should the number of behavior states be equal, but so should their state orders. The state values should be identi-

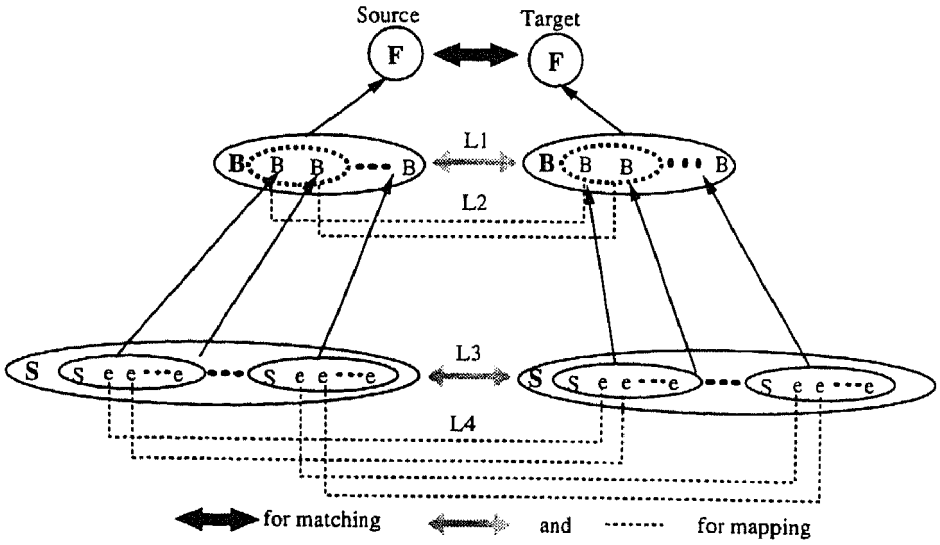


Fig. 9. Mapping levels for data structure.

cal in terms of generalized qualitative state values. If a process structure is associated with a secondary function, its mapping process is different from the one associated with a primary function. A secondary function of FBS type IV considers the initial state and final states as being more important than the intermediate states. As long as the initial state is transformed into the final state, any number of transitional states can be included. The different mapping levels contain different kinds of details about mapping. This provides a great deal of flexibility for knowledge transference in the analogy elaboration process.

7.2.2. Knowledge transference and transformation

After a mapping is found, some structure variables are identified for transference from the source to the target. Each mapping rule causes a transference of an element, an attribute, a relation, or a process, or combination of the primitive elements. The transferred variables might require transformation to adapt to the target situation. General rules are defined for knowledge transference and transformation.

7.3. Analogy evaluation process

The adapted design structure conjecture created by between-domain analogy needs to be evaluated using domain knowledge, according to the design requirements. New design variables introduced from the source to the target do not include associated domain knowledge, thus it is difficult to justify the design conjecture. In such circumstances, automatic machine checking of the structure and behavior is not realistic. Therefore, in a design exploration model, interaction between human and machine is recommended (Kolodner, 1991). Interactive evaluation processes and visual design conjectures are proposed as important factors that overcome this lack of between-domain knowledge.

8. EXAMPLES

The framework described in the previous sections for analogy-based design has been developed as **DE**sign **S**upport **S**ystem Using Analogy (DESSUA), of which the core system is implemented to demonstrate the feasibility, practicality, and potentiality of the model. DESSUA is capable of representing design prototypes (using a description language [Qian, 1995]), carrying on design processes using analogies and exploring new design ideas.

DESSUA is implemented on Sun workstations. The analogy engine is written in IBUKI Common Lisp, the requirement processor, the prototype manager, and the UI are programmed in C and XView. Part of the prototype manager is integrated with Protokit, implemented by Balachandran in the Key Centre of Design Computing, University of Sydney.

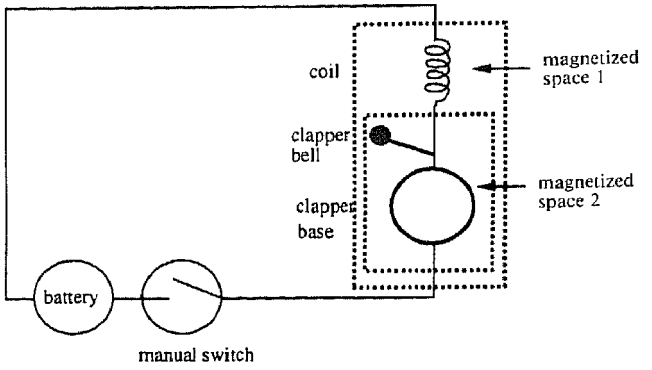


Fig. 10. Buzzer's circuit.

Two examples are presented to demonstrate the ideas and the system. Although there are target designs in the current design knowledge base, we want to produce new designs by analogy. From the existing structures, behaviors and functions, an analogical design may be retrieved. By exploring the target and source, we expect to produce new designs.

The first example shows how a buzzer design is created from a flashing cursor program. Two structure elements are replaced with a process structure through the analogical mapping and transference. The second example designs new doors from analogous behavior of a curtain design. A different structure element is introduced to produce a different kind of door.

8.1. Designing a buzzer using a serial behavior analogy

A buzzer is typically designed with a clapper bell, clapper base, coil, battery, and switch (Fig. 10). The buzzing is controlled by the magnetized clapper base repeatedly connecting and disconnecting the bell to the base. This behavior achieves the buzzing function (Fig. 11).

The relevant part of the buzzer design prototype is shown below; comments follow the semicolon.

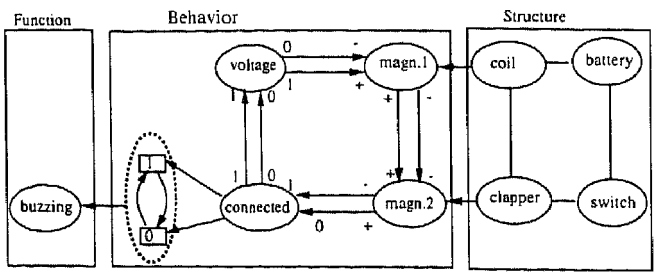


Fig. 11. Buzzer's behavior.

```
(buzzer                                                          ; design name
    (ELEMENT
        (battery)
        (switch coil (clapper (clapper-bell clapper-base)))      ; clapper is composed with
                                                                 ; clapper-bell and clapper-base
    )
    (RELATION
        (connect switch battery)
        (connect battery coil)
        (connect coil clapper-bell)
        (connect clapper-bell clapper-base D (1 0))              ; dynamic relation
        (connect clapper-base switch)
    )
    (FUNCTION
        (1 t (buzzing) iv ← (REPEAT (ii ← (connected (1)) (ii ← (connected) (0))))
    )                                                            ; 'connected' is changed between
                                                                     1 and 0
    (BEHAVIOR
        ((connected clapper) (10)                                ; dynamic behavior changes
            ← (magnetized_space2 clapper) (- +)                  ; indirect behavior variables
            ← (magnetized_space1 coil) (- +)
            ← (voltage coil) (01))
        ((voltage coil) (01) ← (connected clapper) (01))
    )
)
```

One primary function of the buzzer is *buzzing* derived from a temporal behavior. The FBS path type is IV, which means the function is achieved by a sequence of behavior states. In this example, *connected* is the dynamic behavior variable with its value as a state changed to *1* (or on) and *0* (or off) repetitively.

The retrieval is performed by function and behavior matching. Although the description of the function variables and dynamic behavior variables are different in different designs, they are turned into variable names at retrieval time. In other words, the following function and behavior representation is used as index for searching:

```
(FUNCTION
    (1 t (?F1) iv ← (REPEAT (ii ← (?B1) (1)) (ii ← (?B1) (0))))
```

?F1 and ?B1 are function and behavior variable names, respectively. With the repetition of the change in behavior states, a software program design *flashing cursor* is retrieved (Fig. 12). The relevant part of its design prototype is shown below:

```
(flashing_cursor                                                 ; design name
    (ELEMENT
        (screen cursor)                                          ; two structure elements
    )
    (RELATION
        (on cursor screen D (1 0))                               ; the relation is dynamic
    )
    (PROCESS                                                     ; the process has a while loop
        (WHILE ( ) DO                                            ; each loop has 4 actions:
                ((copy cursor screen) ⇒ (copied) (1))            ; 1: copy let the behavior
                                                                     variable
                ((wait half second))                             ; 'copied' state to 1; 2: wait;
                ((remove cursor screen) ⇒ (copied) (0))          ; 3: remove cursor;
                ((wait half second))                             ; 4: wait
        )
    )
    (FUNCTION
        (1 t (flashing) iv ← (REPEAT (ii ← (copied) (1)) (ii ← (copied) (0))))
    )                                                            ; show behavior changes
    (BEHAVIOR
    )
)
```

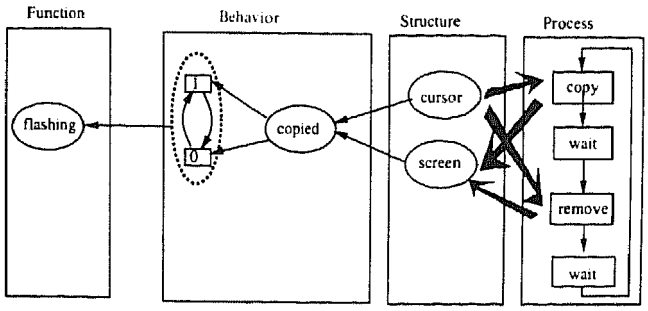


Fig. 12. Flashing cursor's behavior.

The behavior *copied* is derived from the process structure in (PROCESS ...). There is no indirect behavior, thus (BEHAVIOR ...) is empty. *Flashing* is a primary function with a kind of temporal behavior. The FBS path type is also IV and there is a dynamic behavior variable, *copied*, with its state changed from *1* (can be on or interpreted as a boolean TRUE) and *0* (off or as FALSE) repetitively.

The behavior of the two designs, buzzer and flashing cursor, are analogical because both have repetitive changes from one state to another to achieve a function. The flashing effect is accomplished by copying the cursor image on the screen for half a second, then removing it and waiting another half a second, copying it again, removing it again, and so on. This on/off behavior is exactly the same as the buzzing effect although the clapper connection and the cursor are completely different concepts in different design domains.

The behavior of both designs belong to FBS type IV, thus the process structure mapping rule, **Rule 7**, is applied. The mapping between buzzer and flashing cursor design is shown in Figure 13.

The behavior matching between the two designs occurs in the state changes. Their associated behavior variables, *connected* and *copied* are mapped. The rest of behavior in the buzzer is mapped to nothing in the flashing cursor. From the buzzer design, *coil* and *clapper* are two direct structure elements that affect the behavior, whereas in the flashing cursor design *cursor* and *screen* are the two direct structure elements. These two direct structure elements are also mapped. The last map is *battery* and *switch* in the buzzer to the process structure in the flashing cursor. They are indirect structures to derive the behavior. The identification and mapping of these key variables are performed by DESSUA by checking ELEMENT, RELATION, PROCESS, BEHAVIOR subjects in the design prototypes.

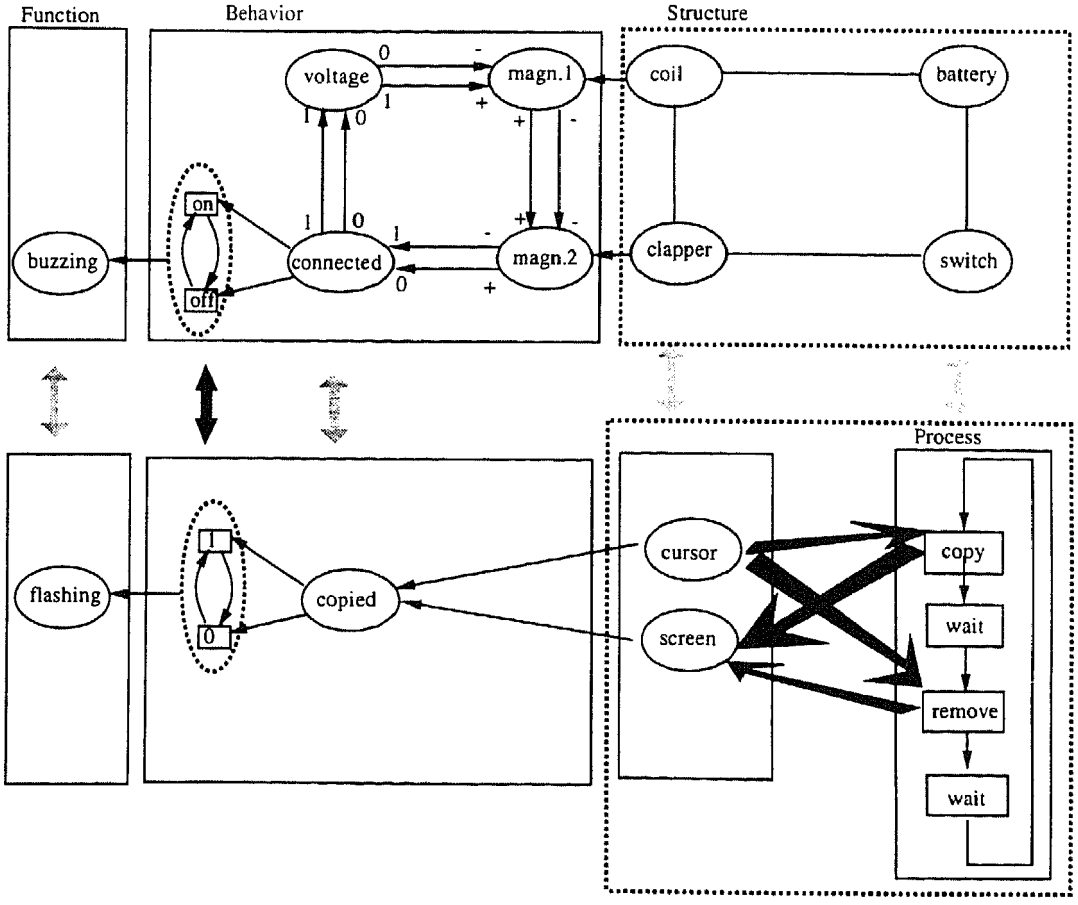


Fig. 13. Mapping between buzzer and flashing cursor designs.

Once the key variables are identified and mapped, knowledge transference and transformation are required to produce a design conjecture. Keeping the direct behavior variables *connected* and direct structure elements *coil* and *clapper*, the process structure from the flashing cursor design is transferred. The following script shows the transference and transformation process to suggest a design conjecture (capital characters are prompts by DESSUA, lower cases are typed in by the designer, and comments are started by a semicolon):

```
====================
CANDIDATE INFERENCES:
====================                                          ; structure to be transferred
STRUCTURE:
    (ELEMENT
        (COIL (CLAPPER (CLAPPER-BELL CLAPPER-BASE)))          ; clapper is composed with
                                                              ; clapper-bell and clapper-
                                                                  base
    )
    (RELATION
        (CONNECT COIL CLAPPER)
        (CONNECT CLAPPER-BELL CLAPPER-BASE D (1 0))
    )
    (PROCESS
        (WHILE ( ) DO
                  ((COPY COIL CLAPPER) ⇒ (CONNECTED) (1))     ; replaced cursor with coil
                                                                  and
                  ((WAIT HALF SECOND))                        ; screen with clapper
                  ((REMOVE COIL CLAPPER) ⇒ (CONNECTED) (0))   ; also replaced copied with
                                                                  connected
                  ((WAIT HALF SECOND))
        )
    )
================================================
FOR CONJECTURE FROM ANALOGICAL INFERENCES, ANSWER:            ; Start evaluation
    'Y' for ACCEPT;
    'M' for MODIFICATION;
    'N' for REJECTION.
================================================
(COIL (CLAPPER (CLAPPER-BELL CLAPPER-BASE))) ? y              ; accept
(CONNECT COIL CLAPPER) ? y                                    ; accept
(CONNECT CLAPPER-BELL CLAPPER-BASE D (1 0)) ? y               ; accept
(WHILE () DO
    ((COPY COIL CLAPPER) ⇒ (CONNECTED) (ON)) ? m              ; modify copy to hit
        (hit coil clapper) ⇒ (connected) (1))                 ; typed by the designer
    ((WAIT HALF SECOND)) ? y                                  ; accept
    ((REMOVE COIL CLAPPER) ⇒ (CONNECTED) (0)) ? y             ; accept
    ((WAIT HALF SECOND)) ? y                                  ; accept
)
===========
CONJECTURE:
===========
STRUCTURE:
    (ELEMENT
        (COIL (CLAPPER (CLAPPER-BELL CLAPPER-BASE)))
    )
    (RELATION
        (CONNECT COIL CLAPPER)
         (CONNECT CLAPPER-BELL CLAPPER-BASE D (1 0))
    )
    (PROCESS
        (WHILE ( ) DO
                  ((HIT COIL CLAPPER) ⇒ (CONNECTED) (1))
                  ((WAIT HALF SECOND))
                  ((REMOVE COIL CLAPPER) ⇒ (CONNECTED) (0))
                  ((WAIT HALF SECOND))
        )
    )
```

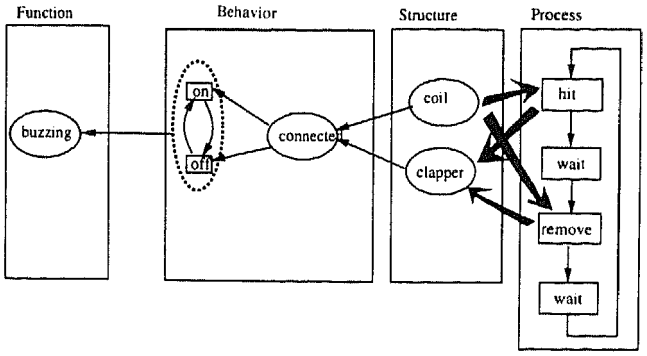


Fig. 14. New buzzer's behavior.

The candidate inference is a direct result of substitution adaptation by DESSUA. It then prompts the designer for an evaluation. By examining the process structure part (PROCESS), it can be seen that *copy* is not an appropriate action to produce a *connected* behavior, so the designer changes to another action, which produces the same behavior; for example, *hit* is one of the possible actions. Figure 14 shows the new buzzer.

The present design uses a *coil* to *hit* the *clapper.* These structure elements may not be ideal for a real design, rather than the *clapper-bell* to *hit* the *clapper-base.* However, this element substitution is not automated in this system.

Although the conjecture of the new buzzer design does not have details of how the structure elements are related and how the control signal is sent to start the hit action, the new concept is novel, in the sense of incorporating a software program that generates an electronical signal to create a force to hit the clapper.

8.2. Designing a new door using a dynamic behavior analogy

The following example demonstrates a door design using a dynamic behavior analogy. The behavior discovered through the FBS path of a target concept design's primary function provides the retrieval cue to find other analogous designs.

A door structure is dynamic in that the relation between a door and a door frame is changeable. For different positions of the door, there are corresponding functions. These functions are achieved related to the status of a dynamic behavior and are FBS type II. If a door has no internal process to control the opening and closing, this door only has a data structure.

At retrieval time, each of the behaviors associated with each of the primary functions are used as retrieval cues to search for analogy candidates. Unlike function retrieval, which requires similarity of the function description, behavior retrieval constraints specify that both design FBS paths have the same FBS type and matching behavior graphs. Behavior matching is not label matching per se, but rather a form of a semantic matching.

For demonstration purposes, the primary function *allow access* is chosen. This function is achieved by FBS type II (i.e., achieved by a state of a dynamic behavior). The following designs are retrieved for FBS type II from the designs database: curtain, flush tank, hot–cold water faucet, locker, scales, water tap. However, only curtain and water tap have the same behavior as the door's when we apply the dynamic structure mapping process (**Rule 5**) and the subgraph isomorphism algorithm (Qian, 1995).

Taking the curtain suggested from the retrieval process as an example, one of the common behaviors between the curtain design and the door design is that they both contain a behavior variable capable of changing from 0 to a constant (> 0). The two states (0 and > 0) directly contribute to two functions.

The curtain's behavior is similar to the behavior of the door, that is, they share same qualitative causal relations. The FBS paths are shown in Figure 15.

Based on these FBS paths, the script run by DESSUA showing the process of behavior matching between the door and curtain designs (with debug on), is given below:

```
============ Target concept design ============
Obtaining DOOR's behavior graphs ......

Initial BSG elements: (AREA DOOR HINGE)
causal dir = ((2 1 ((+) (+)))
              (1 0 ((+) (+))))
expand-list BSG elements (AREA DOOR HINGE)
            with (DOOR_FRAME)
Added causal dir:
causal entry = (3 2 ((S) (S)))
causal entry = (2 3 ((S) (S)))
causal entry = (2 1 ((+) (+)))
causal entry = (1 0 ((+) (+)))
Final BSG elements:
        (AREA DOOR HINGE DOOR_FRAME)
BSG:
```

```
; obtain target-matrix or behavior
; structure graphic (BSG) of main stream
      [Qian 1995]
; elements in BSG (shows behav. change)
; indexes of elements are 0, 1, 2
; 0:area, 1:door, 2:hinge
; expanded with static element
; door_frame which has index 3
; S means static relation
; causal rel from door_frame to hinge
; causal rel from hinge to door_frame
; causal rel from hinge to door
; causal rel from door to hinge

; behavior and structure element list
; generalized qualitative causal
; relation CC can be ++, --, etc.
```

```
    0       0       0       0                              ; C means continue change of
    (CC)    0       0       0                              ; a dynamic variable
    0       (CC)    0       (S)                            ; S means static
    0       0       (S)     0                              ; Dynamic change can be also discrete (D)
============ Source concept design ============           ;
Obtaining CURTAIN's behavior graphs ......                ; obtain source-matrix or BSG
Initial BSG elements: (LIGHT CURTAIN RINGS STRING)        ; element indexes,
                                                          ; 0:light, 1:curtain, 2:rings, 3:string
causal dir = ((3 2 ((+) (-)))
              (2 1 ((+) (+) (+)))                         ; 2 behavior vars related to curtain
              (1 0 ((+) (-))))
expand-list BSG elements (LIGHT CURTAIN RINGS STRING)
            with (ROD)                                    ; expanded with static element
                                                          ; rod with index 4
causal entry = (4 2 ((S) (S)))
causal entry = (2 4 ((S) (S)))
causal entry = (3 2 ((+) (-)))
causal entry = (2 1 ((+) (+) (+)))                        ; two behavior variables for
causal entry = (1 0 ((+) (-)))                            ; curtain (1)
Final BSG elements:
        (LIGHT CURTAIN RINGS STRING ROD)                  ; behavior and structure element list
BSG:
    0       0       0       0       0
    (CC)    0       0       0       0
    0       (CC)    0       0       (S)
    0       0       (CC)    0       0
    0       0       (S)     0       0
================================================
Calculating the isomorphism of two BSGs:
b-list = (AREA DOOR HINGE DOOR_FRAME)                     ; base list (short list)
t-list = (LIGHT CURTAIN RINGS STRING ROD)                 ; transformation list
tmp = (CURTAIN RINGS STRING ROD), del-elist = NIL         ; find possible delete elements
tmp = (RINGS STRING ROD), del-elist = (CURTAIN)           ; (no behav. variable nor static
                                                              struct. elem.)
tmp = (STRING ROD), del-elist = (CURTAIN)
tmp = (ROD), del-elist = (STRING CURTAIN)                 ; try delete CURTAIN or STRING
nth elem to delete = 1                                    ; curtain (1) is tried first, but
n = 1, t-list = (LIGHT RINGS STRING ROD)                  ; no isomorphism is found
nth elem to delete = 3                                    ; string (3) is tries second, and an
n = 3, t-list = (LIGHT CURTAIN RINGS ROD)                 ; isomorphism is found
                                                          ; reverse dependencies are
                                                          ; checked and results the following
                                                          ; mapping
isom res = ((DOOR CURTAIN)                                ; design name mapping
            ((AREA DOOR HINGE DOOR_FRAME)                 ; element mapping
             (LIGHT CURTAIN RINGS ROD))
             ((0     0     0     0)                       ; graph matching successful
              ((CC)  0     9     0)
              (0     (CC)  0     (S))
              (0     0     (S)   0))
             ((0     0     0     0)
              ((CC)  0     0     0)
              (0     (CC)  0     (S))
              (0     0     (S)   0))))
Mapping set is:
============================================================
             TARGET                              SOURCE
============================================================
  NAME:
             DOOR                                CURTAIN
  BEHAVIOR:
```

```
            (ACCESS AREA)                    (INTENSITY LIGHT)
  STRUCTURE:
     TARGET: ((SCREWED_IN HINGE DOOR)
              (DOOR_ANGLE DOOR DOOR_FRAME D (0 - 90 DEGREE))
              (ANGLE_ANGLE HINGE D (0 - 90 DEGREE))
              (SCREWED_IN HINGE DOOR_FRAME))
     SOURCE: ((MOVE_ALONG RINGS ROD D (0 - 10 CM))
              (IMPOSED_ON RINGS ROD
              (PASSED_THROUGH STRING RINGS
              (STITCHED (TOP CURTAIN) RINGS)))
  ELEMENT:
            R                                R
            (DOOR HINGE DOOR_FRAME)          (CURTAIN RINGS ROD)
  EXTERNAL:
            WALL                             (TOP WINDOW)
```

The graphical representation of the structure mapping is shown in Figure 16.

The transference and transformation process for FBS type II design is different from the process for FBS type IV in the previous example. It consists of the following steps (demonstrated using the example shown in Fig. 17):

Step 1: Identify the elements between the final behavior element and the static element in the target design (Fig. 17.1);

Step 2: Replace the intermediate elements, *hinge*, with the ones in the source graph, *rings* (Fig. 17.2);

Step 3: Bring in the relation if it is possible (Fig. 17.3; e.g., *stitched-top* and *imposed*);

Step 4: If a new element cannot be related with the suggested relation from the source design, there are two ways to fix this problem:

a. add the source element, for example, *rod*, to the target, resulting in a new design (Fig. 18), or

b. change the target's element to fit the relation, resulting in a new design (Fig. 19), for example, rings are imposed on the top of the door frame.

By going through every mapped relation, useful parts of the source design are transferred to the target and a new structure is gradually constructed. When the source structure variable is transferred to the target structure, and if no domain-specific knowledge is used as validation, human interaction is needed.

The following script shows the steps of the transference and transformation process:

```
  ====================
  CANDIDATE INFERENCES:                          ; structure to be transferred
  ====================
  STRUCTURE:
       (MOVE_ALONG RINGS DOOR_FRAME)             ; static struct rod is replaced by
                                                      door_frame
       (IMPOSED_ON RINGS DOOR_FRAME)
(PASSED_THROUGH STRING RINGS)
       (STITCHED (TOP DOOR) RINGS)               ; direct struct curtain is replaced
                                                      by door
  ================================================
  FOR CONJECTURE FROM ANALOGICAL INFERENCES, ANSWER: ; Start evaluation
       'Y' for ACCEPT;
       'M' for MODIFICATION;
       'N' for REJECTION.
  ================================================
  (MOVE_ALONG RINGS DOOR_FRAME) ? m              ; modify relation
       (move_along rings rod)                    ; see Fig. transfer step 4a.
       (fixed rod door_frame)                    ; add a new relation
  (IMPOSED_ON RINGS ROD) ? y                     ; take as it is,
  (PASSED_THROUGH STRING RINGS)                  ; transferred all
  (STITCHED (TOP DOOR) RINGS) ? m                ; change to folded door
       (stitched (top folded_door) rings)
```

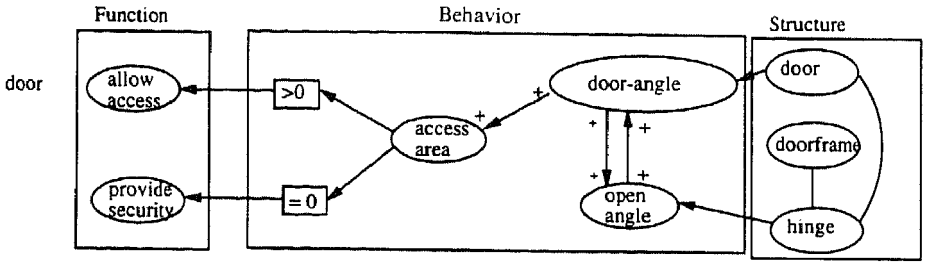
```
==========
CONJECTURES:                                    ; a new structure is created
==========
STRUCTURE:                                      ; See folding door figure
     (MOVE_ALONG RINGS ROD)
     (FIXED ROD DOOR_FRAME)
     (IMPOSED_ON RINGS ROD)
     (PASSED_THROUGH STRING RINGS)
     (STITCHED (TOP FOLDED_DOOR) RINGS)
```

Alternatively, the designer can change the door frame to produce a sliding door (Figure 19) in the following adaptation process:

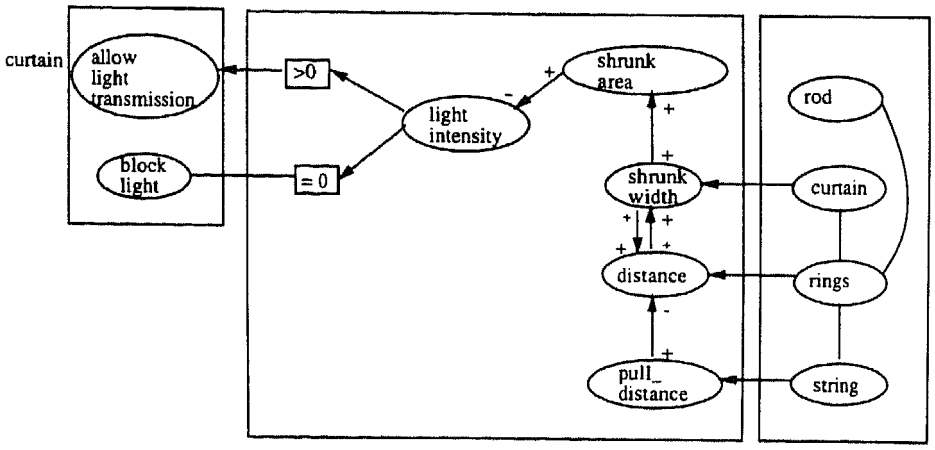
```
=============================================
(MOVE_ALONG RINGS DOOR_FRAME) ? m               ; modify relation
=============================================
     (move_along rings changed_door_frame)      ; see Fig. transfer step 4b.
(IMPOSED_ON RINGS CHANGED_DOOR_FRAME) ? y       ; accept
(PASSED_THROUGH STRING RINGS) ? n               ; discard the relation
(STITCHED (TOP DOOR) RINGS) ? y                 ; accepted although transference
CONJECTURES:                                    ; a new structure is created
STRUCTURE:                                      ; See folding door figure
     (MOVE_ALONG RINGS CHANGED_DOOR_FRAME)
     (IMPOSED_ON RINGS (TOP CHANGED_DOOR_FRAME))
     (STITCHED (TOP DOOR) RINGS)
```

9. CONCLUSION

DESSUA aims at assisting creative design rather than automating it. Interaction between the system and the human designer is important. It is claimed that the feedback from analogy retrieval and analogical mapping and transferring can stimulate the human's creative thinking. DESSUA is an interactive system with capabilities of helping creative design.



a. FBS path for door design



b. FBS for curtain design

Fig. 15. FBS paths for door and curtain.

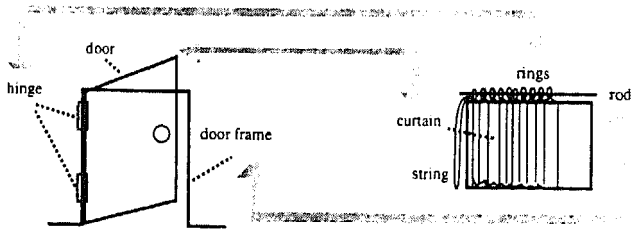


Fig. 16. Structure mapping between door and curtain.

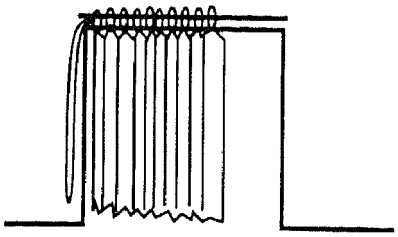


Fig. 18. "Folding" door design.

Between-domain analogy-based design requires design knowledge represented by: (1) qualitative causal relations of how design structures achieve functions through relevant behaviors; (2) generalized design knowledge; and (3) three distinct categories—function, behavior, and structure. As analogous designs do not share superficial commonalities, the causal relations provide the deep understanding of the design and provide a platform for analogical reasoning. Generalized knowledge provides sufficient information to carry out the reasoning task. Function, behavior, and structure are characterized as design variables and play different roles in understanding the design.

A formalism that represents structure, behavior, and function has been presented. A design structure is defined using a data structure and process structure. While the data structure is composed of elements, attributes and relations, the process structure includes operations and processes. Any of the data and process structures is a structure variable.

Behavior is represented by behavior type, behavior variable, and qualitative causal relations. Generalized behavior types are spatial, temporal, and aesthetic. The value of a behavior variable is either deduced from a structure variable or an exogenous variable. Qualitative causal relations describe static and dynamic characteristics of behavior.

Function consists of two parts: variable and FBS path. While the function variable expresses intentions of a design using action and flux taxonomies, FBS path type generalizes a design goal achievement path. Although designs are different from one to another, the FBS path type is limited to one of four kinds: a function can be achieved by (1) a static behavior, (2) a dynamic behavior state, (3) parallel behavior, or (4) sequential behavior.

This formalism incorporates different kinds of designs from different domains. Generalized behavior and function representation is taken into account to facilitate the search for analogous designs in different domains. Qualitative causal relations give generic causal explanations about the behavior of a design while the FBS type is concluded from the goal achievement path.

The current design prototype has capabilities to meet most requirements of knowledge representations for analogy-based design, but it lacks the knowledge required for designs with dynamic structures. This paper extends the notion of design prototype by adding qualitative causal knowledge. This is a unified knowledge representation for structures with static and dynamic characteristics, and describes generic behavior–structure and function–behavior relations.

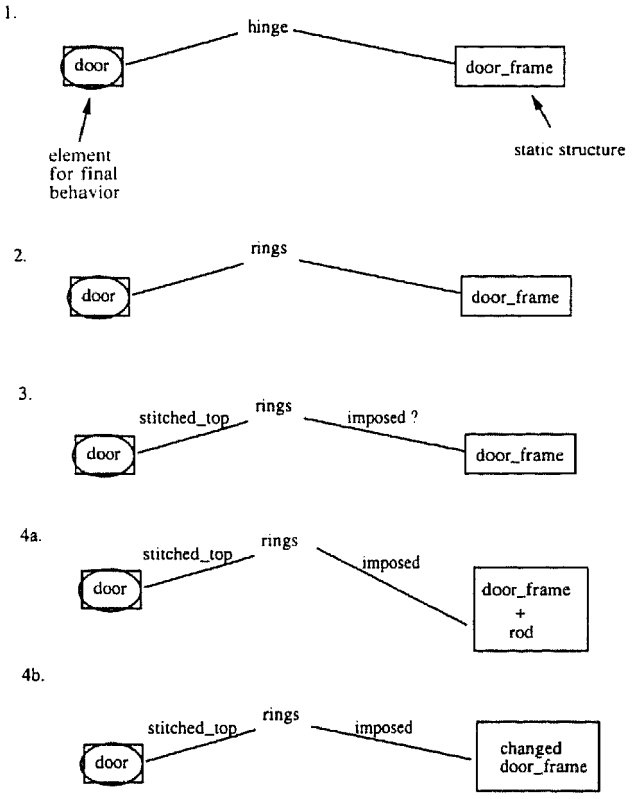


Fig. 17. Transference process from the curtain design to the door design.

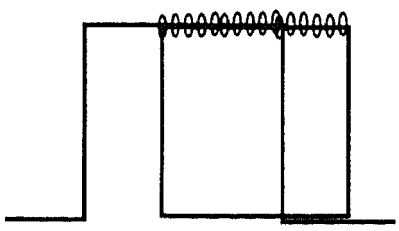


Fig. 19. "Sliding" door design.

ACKNOWLEDGMENTS

This work was executed at the Key Centre of Design Computing, University of Sydney.

REFERENCES

Bhansali, S., & Harandi, M. (1992). Synthesizing UNIX programming using derivational analogy. No. KSL 92-02. *Knowledge Systems Laboratory*, Stanford University, Stanford.

Bobrow, D.G. (Ed.) (1984). *Qualitative reasoning about physical systems*. Elsevier, North-Holland, Amsterdam.

Carbonell, J.G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In *Machine Learning II: An Artificial Intelligence Approach*, (Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., Eds.), pp. 371–392. Morgan Kaufmann, Los Altos, California.

DeKleer, J., & Brown, J.S. (1984). A qualitative physics based on confluences. *Artif. Intell. 24*, 7–83.

Falkenhainer, B., Forbus, K.D., & Gentner, D. (1989/90). The structure-mapping engine: Algorithm and examples. *Artif. Intell. 41*, 1–63.

Forbus, D.K., & Gentner, D. (1990). Causal reasoning about quantities. In *Readings in Qualitative Reasoning About Physical Systems*, (Weld, D.S., and DeKleer, J., Eds.), pp. 666–677. Morgan Kaufmann, Los Altos, California.

Gero, J.S. (1990). Design prototypes: A knowledge representation schema for design. *AI Magazine 11(4)*, 26–36.

Gero, J.S. (1994). Towards a model of exploration in computer-aided design. In *Formal Design Methods for CAD*, (Gero, J.S., and Tyugu, E., Eds.), pp. 315–336. North-Holland, Amsterdam.

Iwasaki, Y., & Chandrasekaran, B. (1992). Design verification through function- and behaviour-oriented representations: Bridging the gap between function and behaviour. In *Artificial Intelligence in Design '92*, (Gero, J.S., Ed.), pp. 597–616. Kluwer, Dordrecht.

Kedar-Cabelli, S.T. (1988). Toward a computational model of purpose-directed analogy. In *Analogica*, (Prieditis, A., Ed.), pp. 89–107. Morgan Kaufmann, Pitman, London.

Kolodner, J. (1991). Improving human decision making through case-based decision making. *AI Magazine 12*, 52–68.

Maher, M.L., Balachandran, B., & Zhang, C.M. (1995). *Case-based reasoning in design*. Lawrence Erlbaum, Hillsdale, NJ.

Maher, M.L., Zhao, F., & Gero, J.S. (1989). Creativity in humans and computers. In *Knowledge-Based Systems in Architecture*, (Gero, J.S., and Oksala, T., Eds.), pp. 129–141. Acta Polytechnica Scaondinavica, Helsinki.

Qian, L. & Gero, J.S. (1993). Design part classification by goal achievement. In *Reasoning About Function*, (Kumar, A.N., Ed.), pp. 121–125. Workshop Preprints, AAAI-93, Washington, DC.

Qian, L. (1995). *Creative Design by Analogy*, Ph.D. Thesis, Department of Architectural and Design Science, University of Sydney, Australia.

Rosenman, M.A., Gero, J.S., & Oxman, R.E. (1991). What's in a case: The use of case based, knowledge based and databases in design. In *CAAD Futures '91*, (Schmitt, G.N., Ed.), pp. 263–277. ETH, Zurich.

Sembugamoorthy, V., & Chandrasekaran, B. (1986). Functional representation of devices and compilation of diagnostic problem-solving systems. In *Experience, Memory and Reasoning*, (Kolodner, J., and Riesbeck, C., Eds.), pp. 47–73. Erlbaum, Hillsdale, NJ.

Sycara, K., & Navinchandra, D. (1991). Index transformation techniques for facilitating creative use of multiple cases. In *IJCAI-91 Workshop on AI in Design*, (Gero, J.S., Ed.), pp. 15–20. University of Sydney, Australia.

Tham, K.W., Lee, H.S., & Gero, J.S. (1990). Building envelope design using design prototypes. *ASHRAE Trans. 96(2)*, 508–520.

Wirth, R., & O'Rorke, P. (1993). Representing and reasoning about function for failure modes and effects analysis. In *Reasoning about Function*, pp. 188–194. Workshop Preprints, AAAI-93, Washington, DC.

Zhao, F., & Maher, M. (1992). Using network-based prototypes to support creative design by analogy and mutation. In *Artificial Intelligence in Design '92*, (Gero, J.S., Ed.), pp. 773–793. Kluwer, Dordrecht.

Lena Qian received a master's degree in computer engineering in Mexico and a Ph.D. from the Key Centre of Design Computing at the University of Sydney in 1995. Her research area is creative design using analogy. She is the author of a number of papers on that topic. Currently she is a development engineer at Canon Information Systems Research Australia.

John Gero is Professor of Design Science and Co-Director of the Key Centre of Design Computing at the University of Sydney. He has been a visiting professor of architecture, civil engineering, mechanical engineering, and computer science in the United States, United Kingdom, France, and Switzerland. He is the author/editor of 23 books and has published over 300 research papers. His research areas cover computational and cognitive models of design with a particular focus on creative design.